External Application Load Balancer overview

This document introduces the concepts that you need to understand how to configure an external Application Load Balancer.

An external Application Load Balancer is a proxy-based Layer 7 load balancer that enables you to run and scale your services behind a single external IP address. The external Application Load Balancer distributes HTTP and HTTPS traffic to backends hosted on a variety of Google Cloud platforms (such as Compute Engine, Google Kubernetes Engine (GKE), Cloud Storage, and so on), as well as external backends connected over the internet or via hybrid connectivity. For details, see Application Load Balancer overview: Use cases (/load-balancing/docs/application-load-balancer#use-cases).

Modes of operation

You can configure an external Application Load Balancer in the following modes:

- **Global external Application Load Balancer.** This is a global load balancer that is implemented as a managed service on Google Front Ends (GFEs) (/security/infrastructure/design#google-frontend-service). It uses the open-source Envoy proxy (https://www.envoyproxy.io/) to support advanced traffic management (/load-balancing/docs/https/traffic-management-global) capabilities such as traffic mirroring, weight-based traffic splitting, request/response-based header transformations, and more.
- **Classic Application Load Balancer.** This is the classic external Application Load Balancer that is global in Premium Tier but can be configured to be regional in Standard Tier. This load balancer is implemented on Google Front Ends (GFEs) (/security/infrastructure/design#google_front_end_service). GFEs are distributed globally and operate together using Google's global network and control plane.
- **Regional external Application Load Balancer.** This is a regional load balancer that is implemented as a managed service on the open-source Envoy proxy (https://www.envoyproxy.io/). It includes advanced traffic management (/load-balancing/docs/https/traffic-management-regional) capabilities such as traffic mirroring, weight-based traffic splitting, request/response-based header transformations, and more.

Load balancer mode	Recommended use cases	Capabilities
Global external Application Load Balancer	Use this load balancer for external HTTP(S) workloads with globally dispersed users or backend services in multiple regions.	• Compatible with GKE using Gateway (/kubernetes-engine/docs/concepts/gateway-api#gatewayclass) (fully orchestrated) or Standalone NEGs (/kubernetes-engine/docs/how-to/standalone-neg) (manual orchestration) • Supports advanced traffic management (/load-balancing/docs/https/traffic-management-global) • Global Anycast external IP addresses over Premium Tier (/network-tiers#tab1) • Can access backends across multiple regions • Supports Cloud CDN (/cdn) • Supports Google Cloud Armor (/armor)
Classic Application Load Balancer	This load balancer is global in Premium Tier but can be configured to be effectively regional in Standard Tier. In the Premium Network Service Tier (/network-tiers/docs/overview), this load balancer offers multi-region load balancing, attempts to direct traffic to the closest healthy backend that has capacity, and terminates HTTP(S) traffic as close as possible to your users. For details about the request distribution process, see Traffic distribution (/load-balancing/docs/https#load_distribution_algorithm). In the Standard Network Service Tier (/network-tiers/docs/overview), load balancing is handled regionally.	• Compatible with GKE using Gateway (/kubernetes-engine/docs/concepts/gateway-api#gatewayclass) (fully orchestrated), Ingress (/kubernetes-engine/docs/concepts/ingress-xlb) (fully orchestrated), or Standalone NEGs (/kubernetes-engine/docs/how-to/standalone-neg) (manual orchestration) • Supports Google Cloud Armor • Fewer traffic routing features

Load balancer mode	Recommended use cases	Capabilities
		See the Load balancing features (/load-balancing/docs/features) page for the full list of capabilities.
Regional external Application Load Balancer	This load balancer contains many of the features of the existing classic Application Load Balancer, along with advanced traffic management (/load-balancing/docs/https/traffic-management-regional) capabilities. Use this load balancer if you want to serve content from only one geolocation (for example, to meet compliance regulations) or if the Standard Network Service Tier (/network-tiers/docs/overview) is desired.	• Compatible with GKE using Standalone NEGs (/kubernetes-engine/docs/how-to/standalone-neg) (manual orchestration) • Supports advanced traffic management capabilities • Regional VIPs using Standard Network Tier (/network-tiers#tab2) • Terminates TLS in a single region that you configure • Serves content from the configured region only For the complete list, see Load balancing features (/load-balancing/docs/features).

Identifying the mode

Cloud consolegcloud (#gcloud)
(#cloud-console)

1. In the Google Cloud console, go to the **Load balancing** page.

   Go to Load balancing (https://console.cloud.google.com/networking/loadbalancing)

2. In the **Load Balancers** tab, the load balancer type, protocol, and region are displayed. If the region is blank, then the load balancer is global. The following table summarizes how to identify the mode of the load balancer.

Load balancer mode	Load balancing type	Region	Network tier
Global external Application Load Balancer	HTTP(S)		PREMIUM

Load balancer mode	Load balancing type	Region	Network tier
Classic Application Load Balancer	HTTP(S)(Classic)		STANDARD or PREMIUM
Regional external Application Load Balancer	HTTP(S)	Specifies a region	STANDARD

Important: After you create a load balancer, you can't edit its mode. Instead, you must delete the load balancer and create a new one.

Architecture

The following resources are required for an external Application Load Balancer deployment:

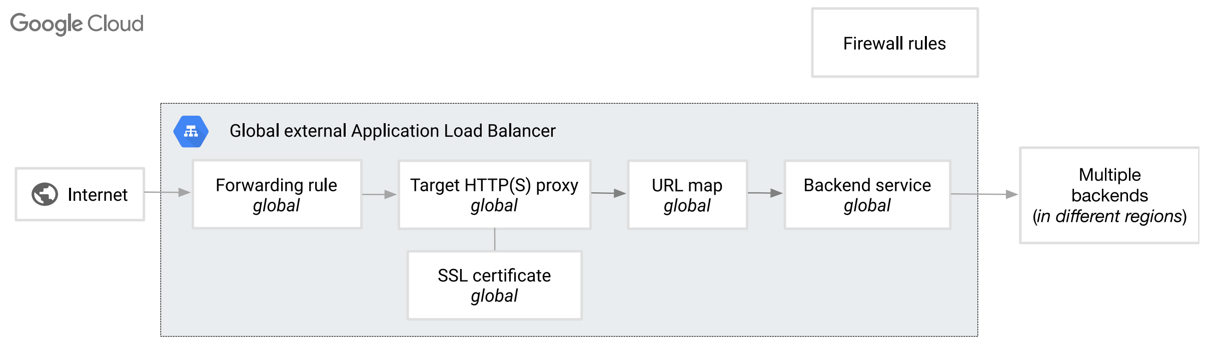
- *For regional external Application Load Balancers only*, a **proxy-only subnet** (#proxy-only-subnet) is used to send connections from the load balancer to the backends.
- An **external forwarding rule** (#forwarding-rule) specifies an external IP address, port, and target HTTP(S) proxy. Clients use the IP address and port to connect to the load balancer.
- A **target HTTP(S) proxy** (#target-proxies) 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.
  - For HTTPS load balancing, the target HTTPS proxy uses **SSL certificates** (#ssl-certificates) to prove its identity to clients. A target HTTPS proxy supports up to the documented number (/load-balancing/docs/quotas#target_pools_and_target_proxies) of SSL certificates.
- The HTTP(S) proxy uses a **URL map** (#url-maps) 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** (#backend-service) distributes requests to healthy backends (/load-balancing/docs/features#backends). The global external Application Load Balancers also support **backend buckets**.

- One or more **backends** must be connected to the backend service or backend bucket.

- A **health check** (#health-checks) periodically monitors the readiness of your backends. This reduces the risk that requests might be sent to backends that can't service the request.
- **Firewall rules** (#firewall-rules) for your backends to accept health check probes. Regional external Application Load Balancers require an additional firewall rule to allow traffic from the proxy-only subnet to reach the backends.

GlobalRegional (#regional)
(#global)

This diagram shows the components of a global external Application Load Balancer deployment. This architecture applies to both, the global external Application Load Balancer, and the classic Application Load Balancer in Premium Tier.



(/static/load-balancing/images/global-ext-https-lb.svg)
Global external Application Load Balancer components

Proxy-only subnet

Proxy-only subnets are only required for regional external Application Load Balancers.

The proxy-only subnet provides a set of IP addresses that Google uses to run Envoy proxies on your behalf. You must create *one* proxy-only subnet in each region of a VPC network where you use regional external Application Load Balancers. The `--purpose` flag for this proxy-only subnet is set to `REGIONAL_MANAGED_PROXY`. All regional Envoy-based load balancers (/load-balancing/docs/proxy-only-subnets#envoy-lb) in the same region and VPC network share a pool of Envoy proxies from the same proxy-only subnet. Further:

- Proxy-only subnets are *only* used for Envoy proxies, not your backends.

- Backend VMs or endpoints of all regional external Application Load Balancers in a region and VPC network receive connections from the proxy-only subnet.
- The IP address of the regional external Application Load Balancer is *not* located in the proxy-only subnet. The load balancer's IP address is defined by its external managed forwarding rule, which is described below.

If you previously created a proxy-only subnet with `--purpose=INTERNAL_HTTPS_LOAD_BALANCER`, you need to migrate the subnet's purpose (/load-balancing/docs/proxy-only-subnets#migrate-purpose) to `REGIONAL_MANAGED_PROXY` before you can create other Envoy-based load balancers in the same region of the VPC network.

Forwarding rules and addresses

Forwarding rules (/load-balancing/docs/forwarding-rule-concepts) 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.

Each forwarding rule for an Application Load Balancer can reference a single port from 1-65535 (/load-balancing/docs/forwarding-rule-concepts#port_specifications). To support multiple ports, you must configure multiple forwarding rules. You can configure multiple forwarding rules to use the same external IP address (VIP) and to reference the same target HTTP(S) proxy as long as the overall *combination* of IP address, port, and protocol is unique for each forwarding rule. This way, you can use a single load balancer with a shared URL map as a proxy for multiple applications.

The type of forwarding rule, IP address, and load balancing scheme used by external Application Load Balancers depends on the mode of the load balancer and which Network Service Tier (/network-tiers/docs/overview) the load balancer is in.

Load balancer mode	Network Service Tier	Forwarding rule, IP address, and Load balancing scheme	Routing from the internet to the load balancer frontend
Global external Application	Premium Tier	Global external forwarding rule (/compute/docs/reference/rest/v1/globalForwardingRules/insert)	Requests routed to the GFE that is closest to the

Load balancer mode	Network Service Tier	Forwarding rule, IP address, and Load balancing scheme	Routing from the internet to the load balancer frontend
Load Balancer		Global external IP address (/compute/docs/reference/rest/v1/globalAddresses/insert) Load balancing scheme: EXTERNAL_MANAGED	client on the internet.
Classic Application Load Balancer	Premium Tier	Global external forwarding rule (/compute/docs/reference/rest/v1/globalForwardingRules/insert) Global external IP address (/compute/docs/reference/rest/v1/globalAddresses/insert) Load balancing scheme: `EXTERNAL`	Requests routed to the GFE that is closest to the client on the internet.
	Standard Tier	Regional external forwarding rule (/compute/docs/reference/rest/v1/forwardingRules/insert) Regional external IP address (/compute/docs/reference/rest/v1/addresses/insert) Load balancing scheme: `EXTERNAL`	Requests routed to a GFE in the load balancer's region.
Regional external Application Load Balancer	Standard Tier	Regional external forwarding rule (/compute/docs/reference/rest/v1/forwardingRules/insert) Regional external IP address (/compute/docs/reference/rest/v1/addresses/insert) Load balancing scheme: `EXTERNAL_MANAGED`	Requests routed to the Envoy proxies in the same region as the load balancer.

For the complete list of protocols supported by external Application Load Balancer forwarding rules in each mode, see Load balancer features (/load-balancing/docs/features#protocols_from_the_clients_to_the_load_balancer).

Target proxies

Target proxies (/load-balancing/docs/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.

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 might appear at the client as `server: Apache/1.0`.

Note: For internet Network Endpoint Groups (NEGs), requests from the load balancer come from different IP ranges depending on the type of NEG (global or regional). For more information, see Authenticating requests (/load-balancing/docs/negs/internet-neg-concepts#authenticating_requests).

The following table specifies the type of target proxy required by external Application Load Balancers in each mode.

Load balancer mode	Target proxy types	Proxy-added headers	Custom heac (/load-balancing/dc headers) supported
Global external Application Load Balancer	Global HTTP (/compute/docs/reference/rest/v1/targetHttpProxies), Global HTTPS (/compute/docs/reference/rest/v1/targetHttpsProxies)	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	✓ Configu backend serv bucket (/load-balancing/dc headers) Not supporte

		for Cloud Trace. • **X-Forwarded-For**: [<supplied-value>,] <client-ip>, <load-balancer-ip> (see X-Forwarded-For header (#x-forwarded-for_header)) (requests only)	
Classic Application Load Balancer	Global HTTP (/compute/docs/reference/rest/v1/targetHttpProxies), Global HTTPS (/compute/docs/reference/rest/v1/targetHttpsProxies)	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.	✓ Configu backend serv bucket (/load-balancing/dc headers)

		• **`X-Forwarded-For`**: [<supplied-value>,] <client-ip>, <load-balancer-ip> (see X-Forwarded-For header (#x-forwarded-for_header)) (requests only)	
Regional external Application Load Balancer	Regional HTTP (/compute/docs/reference/rest/v1/regionTargetHttpProxies), Regional HTTPS (/compute/docs/reference/rest/v1/regionTargetHttpsProxies)	• **`X-Forwarded-Proto`**: [http | https] (requests only) • **`Via: 1.1 google`** (requests and responses) • **`X-Forwarded-For`**: [<supplied-value>,] <client-ip>, <load-balancer-ip> (see X-Forwarded-For header (#x-forwarded-for_header)) (requests only)	🚫

Note: In accordance with RFC 2616 (https://datatracker.ietf.org/doc/html/rfc2616#page-92), the following hop-by-hop headers are not propagated by the target proxy: `Connection`, `Keep-Alive`, `Proxy-Authenticate`, `Proxy-Authorization`, `TE`, `Trailers`, `Transfer-Encoding`, and `Upgrade`.

In addition to headers added by the target proxy, the load balancer adjusts other HTTP headers in the following ways:

- **For the global external Application Load Balancer,** both request and response headers are always converted to lowercase. For example, `Host` becomes `host`, and `Keep-ALIVE` becomes `keep-alive`.

  The only exception to this is when you use global internet NEG backends with HTTP/1.1. For details about how HTTP/1.1 headers are processed with global internet NEGs, see the Internet NEGs overview (/load-balancing/docs/negs/internet-neg-concepts#header-processing).

- **For the classic Application Load Balancer,** request and response headers are converted to lowercase except when you use HTTP/1.1. With HTTP/1.1, headers are proper-cased instead. The first letter of the header's key and every letter following a hyphen (`-`) is capitalized to preserve compatibility with HTTP/1.1 clients. For example, `user-agent` is changed to `User-Agent`, and `content-encoding` is changed to `Content-Encoding`.

- Some headers are coalesced. When there are multiple instances of the same header key (for example, `Via`), the load balancer combines their values into a single comma-separated list for a single header key. Only the headers whose values can be represented as a comma-separated list are coalesced. Other headers, such as `Set-Cookie`, are never coalesced.

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 separated by a single comma to the `X-Forwarded-For` header in the following order:

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

If there is no `X-Forwarded-For` header on the incoming request, these two IP addresses are the entire header value:

```
X-Forwarded-For: <client-ip>,<load-balancer-ip>
```

If the request includes an `X-Forwarded-For` header, the load balancer preserves the supplied value *before* the `<client-ip>,<load-balancer-ip>`:

```
X-Forwarded-For: <supplied-value>,<client-ip>,<load-balancer-ip>
```

Caution: The load balancer does not verify any IP addresses that precede `<client-ip>,<load-balancer-ip>` in this header. The preceding IP addresses might contain other characters, including spaces.

When running HTTP reverse proxy software on the load balancer's backends, the software might append one or both of the following IP addresses to the end of the `X-Forwarded-For` header:

- The IP address of the Google Front End (GFE) that connected to the backend. These IP addresses are in the `130.211.0.0/22` and `35.191.0.0/16` ranges.
- The IP address of the backend system itself.

Thus, an upstream process *after* your load balancer's backend might receive an `X-Forwarded-For` header of the form:

```
<existing-values>,<client-ip>,<load-balancer-ip><GFE-IP><backend-IP>
```

URL maps

URL maps (/load-balancing/docs/url-map-concepts) 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 multi-region load balancing example (/load-balancing/docs/https/setting-up-https), you might not define any URL rules and rely only on the default service. For request routing, the URL map allows you to divide your traffic by examining the URL components to send requests to different sets of backends.

URL maps used with global external Application Load Balancers and regional external Application Load Balancer support several advanced traffic management features such as header-based traffic steering, weight-based traffic splitting, and request mirroring. For more information, see the following:

- Traffic management overview for global external Application Load Balancer (/load-balancing/docs/https/traffic-management-global).
- Traffic management overview for regional external Application Load Balancer (/load-balancing/docs/https/traffic-management-regional).

The following table specifies the type of URL map required by external Application Load Balancers in each mode.

Load balancer mode	**URL map type**
Global external Application Load Balancer	Global (/compute/docs/reference/rest/v1/urlMaps)
Classic Application Load Balancer	Global (/compute/docs/reference/rest/v1/urlMaps) (with only a subset of the features supported (/load-balancing/docs/features#routing-traffic-management))
Regional external Application Load Balancer	Regional (/compute/docs/reference/rest/v1/regionUrlMaps)

SSL certificates

External Application Load Balancers using target HTTPS proxies require private keys and SSL certificates as part of the load balancer configuration:

- Google Cloud provides two *configuration methods* for assigning private keys and SSL certificates to target HTTPS proxies: Compute Engine SSL certificates and Certificate Manager. For a description of each configuration, see Certificate configuration methods (/load-balancing/docs/ssl-certificates#config-tech) in the SSL certificates overview.
- Google Cloud provides two *certificate types*: Self-managed and Google-managed. For a description of each type, see Certificate types (/load-balancing/docs/ssl-certificates#certificate-types) in the SSL certificates overview.

- The type of external Application Load Balancer (global, regional, or classic) determines which configuration methods and certificate types are supported. For details, see Certificates and Google Cloud load balancers (/load-balancing/docs/ssl-certificates#certificate-summary) in the SSL certificates overview.

Mutual TLS

Typically with HTTPS communication, the authentication works only one way: the client verifies the identity of the server. For applications that require the load balancer to authenticate the identity of clients that connect to it, both a global external Application Load Balancer and a classic Application Load Balancer support mutual TLS (mTLS).

With mTLS, the load balancer requests that the client send a certificate to authenticate itself during the TLS handshake with the load balancer. You can configure a trust store that the load balancer uses to validate the client certificate's chain of trust.

Google Cloud uses the `TrustConfig` resource in Certificate Manager to store certificates that the server uses to verify the certificate presented by the client. If you are using a classic Application Load Balancer on the Premium Network Service Tier (/network-tiers/docs/overview) or a global external Application Load Balancer, you can use Certificate Manager to provision and manage your SSL certificates or `TrustConfig` across multiple instances of the load balancer at scale. For more information, see the Certificate Manager overview (/certificate-manager/docs/overview).

For more information about mTLS, see Mutual TLS authentication (/load-balancing/docs/mtls).

SSL policies

SSL policies (/load-balancing/docs/ssl-policies-concepts) specify the set of SSL features that Google Cloud load balancers use when negotiating SSL with 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 an SSL policy to specify the set of SSL features that your load balancer uses when negotiating SSL with clients. In addition, you can apply that SSL policy to your target HTTPS proxy.

The following table specifies the SSL policy support for load balancers in each mode.

Load balancer mode	SSL policies supported

Global external Application Load Balancer	✓
Classic Application Load Balancer	✓
Regional external Application Load Balancer	✓

Backend services and buckets

Backend services provide configuration information to the load balancer. Load balancers use the information in a backend service to direct incoming traffic to one or more attached backends. For an example showing how to set up a load balancer with a backend service and a Compute Engine backend, see Setting up an external Application Load Balancer with a Compute Engine backend (/load-balancing/docs/https/ext-https-lb-simple).

Backend buckets direct incoming traffic to Cloud Storage buckets. For an example showing how to add a bucket to a external Application Load Balancer, see Setting up a load balancer with backend buckets (/load-balancing/docs/https/ext-load-balancer-backend-buckets).

The following table specifies the backend features supported by external Application Load Balancers in each mode.

Load balancer mode	Supported backends on a backend service						Supports backend buckets	Supports Google Cloud Armor	Supports Cloud CDN	Supports IAP	Suppor Service Extensi
	Instance groups	Zonal NEGs	Internet NEGs	Serverless NEGs	Hybrid NEGs	Private Service Connect NEGs					
Global external Application Load Balancer	✓	✓ [2]	✓	✓	✓	✓	✓	✓	✓	✓	✓
Classic Application Load Balancer	✓	✓ [1]	✓	✓	✓	🚫	✓	✓	✓ when using Premium Tier	✓	🚫
Regional external Application	✓	✓ [1]	✓	✓	✓	✓	🚫	🚫	🚫	✓	✓

Load balancer mode	Supported backends on a backend service						Supports backend buckets	Supports Google Cloud Armor	Supports Cloud CDN	Supports IAP	Suppor Service Extensi
Load Balancer											

[1] Use `GCE_VM_IP_PORT` type endpoints with GKE: Use standalone zonal NEGs (/kubernetes-engine/docs/how-to/standalone-neg) or use Ingress (/kubernetes-engine/docs/concepts/ingress-ilb)

[2] Use `GCE_VM_IP_PORT` type endpoints with GKE: Use standalone zonal NEGs (/kubernetes-engine/docs/how-to/standalone-neg)

[3] Supported only with the GKE Gateway controller (/kubernetes-engine/docs/concepts/gateway-api)

For more information, see the following:

- Backend services overview (/load-balancing/docs/backend-service)
- What is Cloud Storage? (/storage/docs/introduction)

Backends and VPC networks

The restrictions on where backends can be located depend on the type of load balancer.

- For the global external Application Load Balancer and the classic Application Load Balancer, all backend instances from instance group backends and all backend endpoints from NEG backends must be located in the same project. However, an instance group backend or a NEG can use a different VPC network in that project. The different VPC networks do not need to be connected using VPC Network Peering because GFEs communicate directly with backends in their respective VPC networks.
- For the regional external Application Load Balancer, all backends must be located in the same VPC network and region.

Protocol to the backends

When you configure a backend service for the 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.

For the complete list of protocols supported, see Load balancing features: Protocols from the load balancer to the backends (/load-balancing/docs/features#protocol-to-backends).

WebSocket support

Google Cloud HTTP(S)-based load balancers have native support for the WebSocket protocol when you use HTTP or HTTPS as the protocol to the backend. The load balancer does not need any configuration to proxy WebSocket connections.

The WebSocket protocol provides a full-duplex communication channel between clients and servers. An HTTP(S) request initiates the channel. For detailed information about the protocol, see RFC 6455 (https://tools.ietf.org/html/rfc6455).

When the load balancer recognizes a WebSocket `Upgrade` request from an HTTP(S) client 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 instance does not return a successful `Upgrade` response, the load balancer closes the connection.

The timeout for a WebSocket connection depends on the configurable *backend service timeout* of the load balancer, which is 30 seconds by default. This timeout applies to WebSocket connections regardless of whether they are in use.

Session affinity for WebSockets works the same as for any other request. For information, see Session affinity (/load-balancing/docs/backend-service#session_affinity).

Using gRPC with your Google Cloud applications

gRPC (https://grpc.io/docs/) 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:

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 a 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.

You must enable TLS on your backends. For more information, see Encryption from the load balancer to the backends (/load-balancing/docs/ssl-certificates/encryption-to-the-backends).

If you want to configure an external Application 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 (/kubernetes-engine/docs/how-to/ingress-http2).

For information about troubleshooting problems with HTTP/2, see Troubleshooting issues with HTTP/2 to the backends (/load-balancing/docs/https/troubleshooting-ext-https-lbs).

For information about HTTP/2 limitations, see HTTP/2 limitations (/load-balancing/docs/https#HTTP2-limitations).

Health checks

Each backend service specifies a health check that periodically monitors the backends' readiness to receive a connection from the load balancer. This reduces the risk that requests might be sent to backends that can't service the request. Health checks do not check if the application itself is working.

For the health check probes, you must create an ingress allow firewall rule (#firewall-rules) that allows health check probes to reach your backend instances. Typically, health check probes originate from Google's centralized health checking mechanism.

Regional external Application Load Balancers that use hybrid NEG backends are an exception to this rule because their health checks originate from the proxy-only subnet instead. For details, see the Hybrid NEGs overview (/load-balancing/docs/negs/hybrid-neg-concepts#envoy-health-checks).

Health check protocol

Although it is not required and not always possible, it is a best practice to use a health check whose protocol matches the protocol of the backend service (/load-balancing/docs/features#protocol-to-backends). For example, an HTTP/2 health check most accurately tests HTTP/2 connectivity to backends. In contrast, regional external Application Load Balancers that use hybrid NEG backends do not support gRPC health checks. For the list of supported health check protocols, see Load balancing features (/load-balancing/docs/features#health-checks).

The following table specifies the scope of health checks supported by external Application Load Balancers in each mode.

Load balancer mode	Health check type
Global external Application Load Balancer	Global (/compute/docs/reference/rest/v1/healthChecks)
Classic Application Load Balancer	Global (/compute/docs/reference/rest/v1/healthChecks)
Regional external Application Load Balancer	Regional (/compute/docs/reference/rest/v1/regionHealthChecks)

For more information about health checks, see the following:

- Health checks overview (/load-balancing/docs/health-check-concepts)
- Creating health checks (/load-balancing/docs/health-checks)

Firewall rules

The load balancer requires the following firewall rules:

- For the global external Application Load Balancers, an ingress allow rule to permit traffic from Google Front Ends (GFEs) to reach your backends.
  For the regional external Application Load Balancer, an ingress allow rule to permit traffic from the proxy-only subnet (#proxy-only-subnet) to reach your backends.
- An ingress allow rule to permit traffic from the health check probes ranges. For more information about health check probes and why it's necessary to allow traffic from them, see Probe IP ranges and firewall rules (/load-balancing/docs/health-check-concepts#ip-ranges).

Firewall rules are implemented at the VM instance level, not on GFE proxies. You cannot use Google Cloud firewall rules to prevent traffic from reaching the load balancer. For the global external Application Load Balancer and the classic Application Load Balancer, you can use Google Cloud Armor (/armor/docs) to achieve this.

The ports for these firewall rules must be configured as follows:

- Allow traffic to the destination port for each backend service's health check.
- For instance group backends: Determine the ports to be configured by the mapping between the backend service's named port (/load-balancing/docs/backend-service#named_ports) and the port numbers associated with that named port on each instance group. The port numbers can vary among instance groups assigned to the same backend service.
- For `GCE_VM_IP_PORT` NEG backends: Allow traffic to the port numbers of the endpoints.

The following table summarizes the required source IP address ranges for the firewall rules:

Load balancer mode	Health check source ranges	Request source ranges
Global external Application Load Balancer	• `35.191.0.0/16` • `130.211.0.0/22`	The source of GFE traffic depends on the backend type: • Instance groups, zonal NEGs (`GCE_VM_IP_PORT`), and hybrid connectivity NEGs (`NON_GCP_PRIVATE_IP_PORT`): • `130.211.0.0/22` • `35.191.0.0/16` • Internet NEGs (`INTERNET_FQDN_PORT` and `INTERNET_IP_PORT`): • `34.96.0.0/20` • `34.127.192.0/18` • `SERVERLESS` NEGs and backend buckets: Google's production network handles packet routing
Classic Application Load Balancer	• `35.191.0.0/16` • `130.211.0.0/22`	The source of GFE traffic depends on the backend type: • Instance groups, zonal NEGs (`GCE_VM_IP_PORT`), and hybrid connectivity NEGs (`NON_GCP_PRIVATE_IP_PORT`): • `35.191.0.0/16` • `130.211.0.0/22`

Load balancer mode	Health check source ranges	Request source ranges
		• Internet NEGs (`INTERNET_FQDN_PORT` and `INTERNET_IP_PORT`): • `34.96.0.0/20` • `34.127.192.0/18` • `SERVERLESS` NEGs and backend buckets: Google's production network handles packet routing.
Regional external Application Load Balancer	• `35.191.0.0/16` • `130.211.0.0/22` Allowlisting Google's health check probe ranges isn't required for hybrid NEGs. However, if you're using a combination of hybrid and zonal NEGs in a single backend service, you need to allowlist the Google health check probe ranges (/load-balancing/docs/health-check-concepts#ip-ranges) for the zonal NEGs.	The proxy-only subnet (/load-balancing/docs/proxy-only-subnets#proxy_only_subnet_create) that you configure.

Important: Make sure to allow packets from the full health check ranges. If your firewall rule allows packets from only a subset of the ranges, you might see health check failures because the load balancer can't communicate with your backends. This causes HTTP 502 responses.

Shared VPC architecture

External Application Load Balancers support networks that use Shared VPC. Shared VPC lets organizations connect resources from multiple projects to a common VPC network so that they can communicate with each other securely and efficiently by using internal IP addresses from that network. If you're not already familiar with Shared VPC, read the Shared VPC overview (/vpc/docs/shared-vpc).

There are many ways to configure an external Application Load Balancer within a Shared VPC network. Regardless of type of deployment, all the components of the load balancer must be in the same organization.

Load balancer	Frontend components	Backend components
Global external Application Load Balancer Classic Application Load Balancer	The global external IP address, the forwarding rule, the target HTTP(S) proxy, and the associated URL map must be defined in the same host or service project as the backends.	A global backend service must be defined in the same host or service project as the backends (/load-balancing/docs/features#backends) (instance groups or NEGs). Health checks associated with backend services must be defined in the same project as the backend service as well. Cross-project service referencing (#cross-project) is not supported.
Regional external Application Load Balancer	Create the required network and proxy-only subnet in the Shared VPC host project. The regional external IP address, the forwarding rule, the target HTTP(S) proxy, and the associated URL map must be defined in the same project. This project can be the host project or a service project.	You can do one of the following: • Create backend services and backends (/load-balancing/docs/features#backends) (instance groups, serverless NEGs, or any other supported backend types) in the same service project (#service-project) as the frontend components. • Create backend services and backends (instance groups, serverless NEGs, or any other supported backend types) in as many service projects as required. A single URL map can reference backend services across different projects. This type of deployment is known as cross-project service referencing (#cross-project). Each backend service must be defined in the same project as the backends it references. Health checks associated with backend services must be defined in the same project as the backend service as well.

While you can create all the load balancing components and backends in the Shared VPC host project, this model does not separate network administration and service development responsibilities.

Serverless backends in a Shared VPC environment

For a load balancer that is using a serverless NEG backend, the backend Cloud Run, Cloud Functions, or App Engine service must be in the same project as the serverless NEG.

Additionally, for the regional external Application Load Balancer that supports cross-project service referencing, the backend service, serverless NEG, and the Cloud Run service must always be in the same service project.

Cross-project service referencing

In this model, the load balancer's frontend and URL map are in a host or service project. The load balancer's backend services and backends can be distributed across projects in the Shared VPC environment. Cross-project backend services can be referenced in a single URL map. This is referred to as *cross-project service referencing*.

Cross-project service referencing allows organizations to configure one central load balancer and route traffic to hundreds of services distributed across multiple different projects. You can centrally manage all traffic routing rules and policies in one URL map. You can also associate the load balancer with a single set of hostnames and SSL certificates. You can therefore optimize the number of load balancers needed to deploy your application, and lower manageability, operational costs, and quota requirements.

By having different projects for each of your functional teams, you can also achieve separation of roles within your organization. Service owners can focus on building services in service projects, while network teams can provision and maintain load balancers in another project, and both can be connected by using cross-project service referencing.

Service owners can maintain autonomy over the exposure of their services and control which users can access their services by using the load balancer. This is achieved by a special IAM role called the Compute Load Balancer Services User role (/compute/docs/access/iam#compute.loadBalancerServiceUser) (`roles/compute.loadBalancerServiceUser`).

To learn how to configure Shared VPC for a regional external Application Load Balancer—with and without cross-project service referencing, see Set up a regional external Application Load Balancer with Shared VPC (/load-balancing/docs/https/setting-up-reg-ext-shared-vpc).

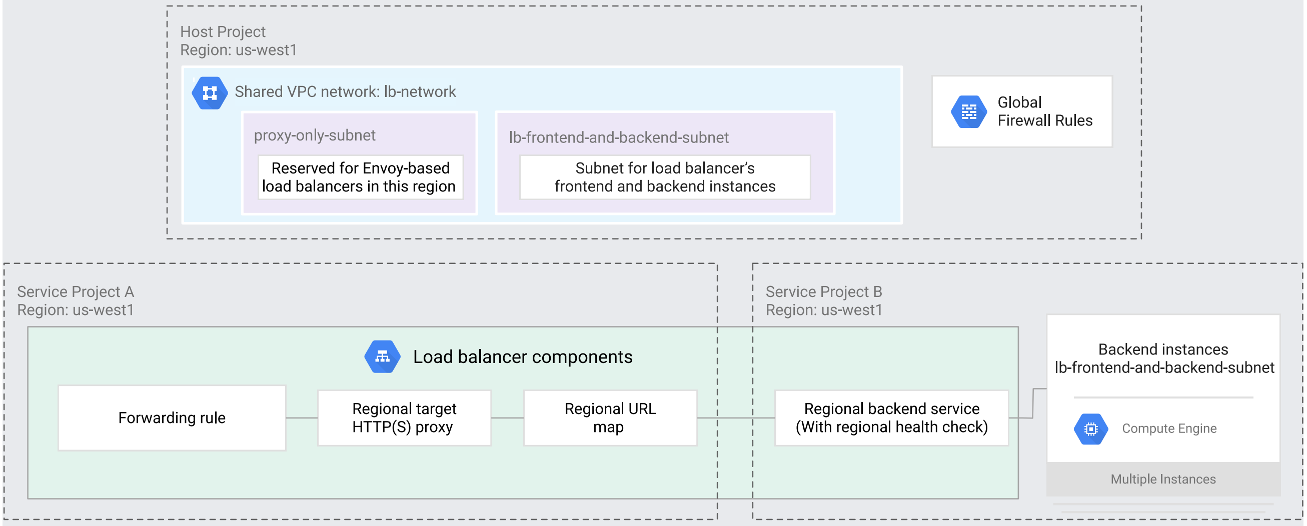
Cross-project service referencing can be used with instance groups, serverless NEGs, and most other supported backend types. However, the following limitations apply:

- With regional external Application Load Balancers, you can't reference a cross-project backend service if the backend service has regional internet NEG backends.
- Cross-project service referencing is *not supported* for the global external Application Load Balancer and the classic Application Load Balancer.

Example 1: Load balancer frontend and backend in different service projects

Here is an example of a deployment where the load balancer's frontend and URL map are created in service project A and the URL map references a backend service in service project B.

In this case, Network Admins or Load Balancer Admins in service project A will require access to backend services in service project B. Service project B admins grant the `compute.loadBalancerServiceUser` IAM role to Load Balancer Admins in service project A who want to reference the backend service in service project B.

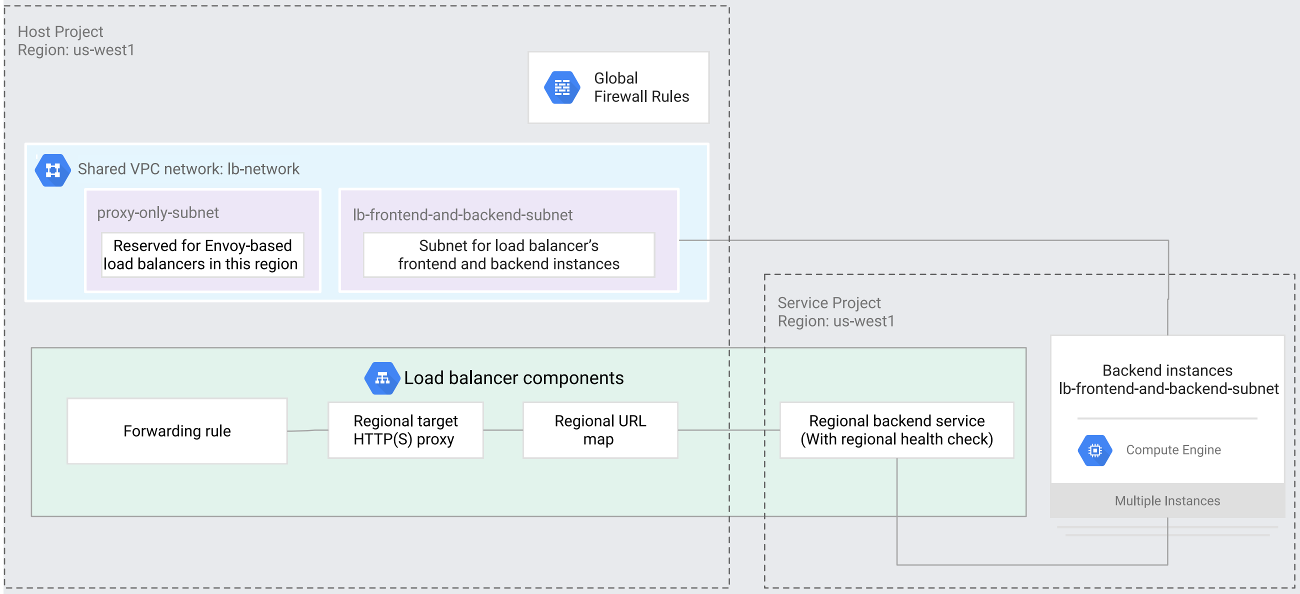

Host Project
Region: us-west1
Shared VPC network: lb-network
proxy-only-subnet
Reserved for Envoy-based load balancers in this region
lb-frontend-and-backend-subnet
Subnet for load balancer's frontend and backend instances
Global Firewall Rules
Service Project A
Region: us-west1
Load balancer components
Forwarding rule
Regional target HTTP(S) proxy
Regional URL map
Service Project B
Region: us-west1
Regional backend service (With regional health check)
Backend instances lb-frontend-and-backend-subnet
Compute Engine
Multiple Instances

(/static/load-balancing/images/reg-ext-https-lb-l7-shared-vpc-service-lb.svg)
Load balancer frontend and backend in different service projects

Example 2: Load balancer frontend in the host project and backends in service projects

In this type of deployment, the load balancer's frontend and URL map are created in the host project and the backend services (and backends) are created in service projects.

In this case, Network Admins or Load Balancer Admins in the host project will require access to backend services in the service project. Service project admins grant the `compute.loadBalancerServiceUser` IAM role to to Load Balancer Admins in the host project A who want to reference the backend service in the service project.


Host Project
Region: us-west1
Global Firewall Rules
Shared VPC network: lb-network
proxy-only-subnet
Reserved for Envoy-based load balancers in this region
lb-frontend-and-backend-subnet
Subnet for load balancer's frontend and backend instances
Service Project
Region: us-west1
Load balancer components
Forwarding rule
Regional target HTTP(S) proxy
Regional URL map
Regional backend service (With regional health check)
Backend instances lb-frontend-and-backend-subnet
Compute Engine
Multiple Instances

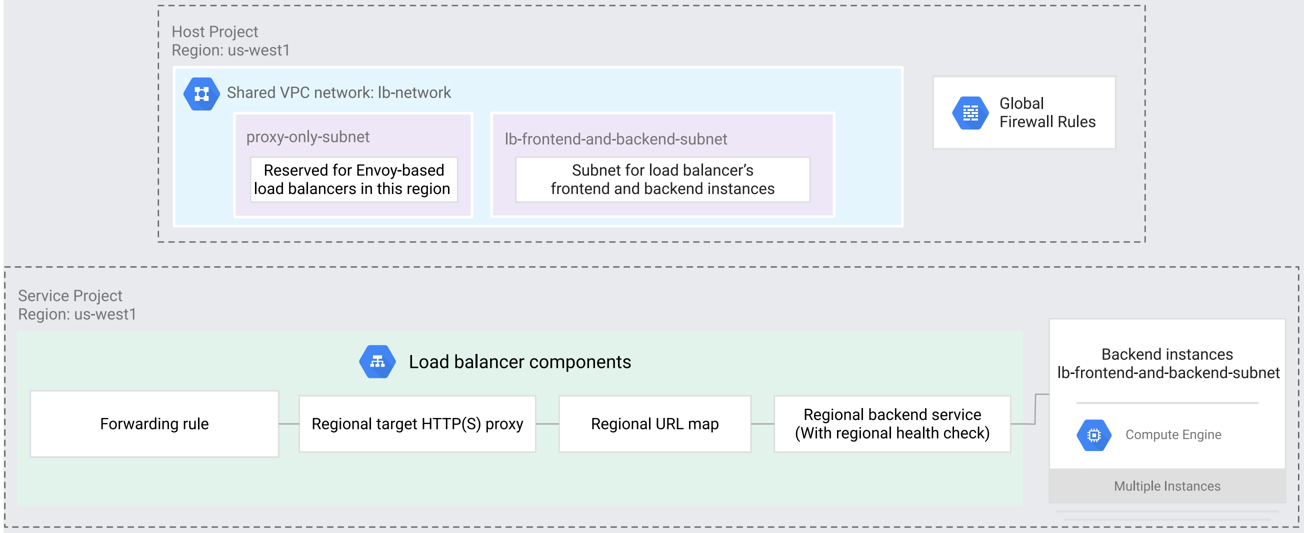
(/static/load-balancing/images/reg-ext-https-lb-l7-shared-vpc-host-lb.svg)

Load balancer frontend and URL map in host project

All load balancer components and backends in a service project

In this model, all load balancer components and backends are in a service project. This deployment model is supported by all HTTP(S) load balancers.

The load balancer components and backends must use the same VPC network.


Host Project
Region: us-west1
Shared VPC network: lb-network
proxy-only-subnet
Reserved for Envoy-based load balancers in this region
lb-frontend-and-backend-subnet
Subnet for load balancer's frontend and backend instances
Global Firewall Rules
Service Project
Region: us-west1
Load balancer components
Forwarding rule
Regional target HTTP(S) proxy
Regional URL map
Regional backend service (With regional health check)
Backend instances lb-frontend-and-backend-subnet
Compute Engine
Multiple Instances

(/static/load-balancing/images/reg-ext-https-lb-l7-shared-vpc-example.svg)

Regional external Application Load Balancer on Shared VPC network

How connections work

Global external Application Load Balancer connections

The global external Application Load Balancers are 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 client requests are 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 probers: `35.191.0.0/16` and `130.211.0.0/22`.

Note: For internet NEGs, requests from the load balancer come from different IP ranges depending on the type of NEG (global or regional). For more information, see Authenticating requests (/load-balancing/docs/negs/internet-neg-concepts#authenticating_requests).

Depending on the backend service configuration, the protocol used by each GFE to connect to your backends can be HTTP, HTTPS, or HTTP/2. For HTTP or HTTPS connections, the HTTP version used is HTTP 1.1.

HTTP keepalive is enabled by default, as specified in the HTTP 1.1 specification. HTTP keepalives attempt to efficiently use the same TCP session; however, there's no guarantee. The GFE uses a client HTTP keepalive timeout of 610 seconds and a default backend keepalive timeout value of 600 seconds. You can update the client HTTP keepalive timeout but the backend keepalive timeout value is fixed. You can configure the request/response timeout by setting the backend service timeout. Though closely related, an HTTP keepalive and a TCP idle timeout are not the same thing. For more information, see timeouts and retries (#timeouts_and_retries).

To ensure that traffic is load balanced evenly, the load balancer might cleanly close a TCP connection either by sending a `FIN ACK` packet after completing a response that included a `Connection: close` header, or it might issue an HTTP/2 `GOAWAY` frame after completing a response. This behavior does not interfere with any active requests or responses.

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 external Application Load Balancers work (/load-balancing/docs/tutorials/about-capacity-optimization-with-global-lb#how_https_load_balancing_works)
in the solutions guide: Application Capacity Optimizations with Global Load Balancing.

Regional external Application Load Balancer connections

The regional external Application Load Balancer is a managed service implemented on the Envoy proxy. The regional external Application Load Balancer uses a shared subnet called a proxy-only subnet to provision a set of IP addresses that Google uses to run Envoy proxies on your behalf. The `--purpose` flag for this proxy-only subnet is set to `REGIONAL_MANAGED_PROXY`. All regional Envoy-based load balancers (/load-balancing/docs/proxy-only-subnets#envoy-lb) in a particular network and region share this subnet.

Clients use the load balancer's IP address and port to connect to the load balancer. Client requests are directed to the proxy-only subnet in the same region as the client. The load balancer terminates clients requests and then opens new connections from the proxy-only subnet to your backends. Therefore, packets sent from the load balancer have source IP addresses from the proxy-only subnet.

Depending on the backend service configuration, the protocol used by Envoy proxies 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 Envoy proxy sets both the client HTTP keepalive timeout and the backend keepalive timeout to a default value of 600 seconds each. You can update the client HTTP keepalive timeout but the backend keepalive timeout value is fixed. You can configure the request/response timeout by setting the backend service timeout. For more information, see timeouts and retries (#timeouts_and_retries).

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.
- External Application Load Balancers support the `HTTP/1.1 100 Continue` response.

For the complete list of protocols supported by external Application Load Balancer forwarding rules in each mode, see Load balancer features (/load-balancing/docs/features#protocols_from_the_clients_to_the_load_balancer).

Source IP addresses for client packets

The source IP address for packets, as seen by the backends, is *not* the Google Cloud external IP address of the load balancer. In other words, there are two TCP connections.

For the global external Application Load Balancers:

- Connection 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 or a forward proxy).
  - **Destination IP address:** your load balancer's IP address.
- Connection 2, from the load balancer (GFE) to the backend VM or endpoint:
  - **Source IP address:** an IP address in one of the ranges specified in Firewall rules (#firewall-rules).
  - **Destination IP address:** the internal IP address of the backend VM or container in the VPC network.

Note: For internet NEGs, requests from the load balancer come from different IP ranges. For more information, see Authenticating requests (/load-balancing/docs/negs/internet-neg-concepts#authenticating_requests).

For the regional external Application Load Balancers:

- Connection 1, from original client to the load balancer (proxy-only subnet):
  - **Source IP address:** the original client (or external IP address if the client is behind NAT or a forward proxy).
  - **Destination IP address:** your load balancer's IP address.
- Connection 2, from the load balancer (proxy-only subnet) to the backend VM or endpoint:
  - **Source IP address:** an IP address in the proxy-only subnet (#proxy-only-subnet) that is shared among all the Envoy-based load balancers deployed in the same region and network as the load balancer.
  - **Destination IP address:** the internal IP address of the backend VM or container in the VPC network.

Return path

For the global external Application Load Balancers, Google Cloud uses special routes not defined in your VPC network for health checks. For more information, see Load balancer return paths (/vpc/docs/routes#special-lb-paths).

For regional external Application Load Balancers, Google Cloud uses open-source Envoy proxies to terminate client requests to the load balancer. The load balancer terminates the TCP session and opens a new TCP session from the region's proxy- only subnet to your backend. Routes defined within your VPC network facilitate communication from Envoy proxies to your backends and from your backends to the Envoy proxies.

Open ports

GFEs have several open ports to support other Google services that run on the same architecture. When you run a port scan, you might see other open ports for other Google services running on GFEs.

Both GFE-based load balancers—global external Application Load Balancers and classic Application Load Balancers—always show ports 80 and 443 as open (along with any other port you've configured in your load balancer's forwarding rules). However, if you haven't configured a forwarding rule for port 80 or for port 443, any connections sent to those ports are refused. Conversely, regional external Application Load Balancers are implemented using Envoy proxies and don't show extra open ports during a scan.

Running a port scan on the IP address of a GFE-based load balancer is not useful from an auditing perspective for the following reasons:

- A port scan (for example, with `nmap`) generally expects no response packet or a TCP RST packet when performing TCP SYN probing. GFEs will send SYN-ACK packets in response to SYN probes only for ports on which you have configured a forwarding rule. GFEs only send packets to your backends in response to packets sent to your load balancer's IP address *and* the destination port configured on its forwarding rule. Packets that are sent to a different IP address or port are not sent to your backends. GFEs implement security features such as Google Cloud Armor. Even without a Google Cloud Armor configuration, Google infrastructure and GFEs provide defense-in-depth for DDoS attacks and SYN floods.
- Packets sent to the IP address of your load balancer could be answered by any GFE in Google's fleet; however, scanning a load balancer IP address and destination port combination only interrogates a single GFE per TCP connection. The IP address of your load balancer is not assigned to a single device or system. Thus, scanning the IP address of a GFE-based load balancer does not scan all the GFEs in Google's fleet.

With that in mind, the following are some more effective ways to audit the security of your backend instances:

- A security auditor should inspect the forwarding rules configuration for the load balancer's configuration. The forwarding rules define the destination port for which your load balancer accepts packets and forwards them to the backends. For GFE-based load balancers, each external forwarding rule can only reference a single destination TCP port (/load-balancing/docs/forwarding-rule-concepts#port_specifications). For a load balancer using TCP port 443, UDP port 443 is used when the connection is upgraded to QUIC (HTTP/3).
- A security auditor should inspect the firewall rule configuration applicable to backend VMs. The firewall rules that you set block traffic from the GFEs to the backend VMs, but don't block incoming traffic to the GFEs. For best practices, see the firewall rules section (#firewall-rules).

TLS termination

The following table summarizes how TLS termination is handled by external Application Load Balancers in each mode.

Load balancer mode	TLS termination
Global external Application Load Balancer	TLS is terminated on a GFE, which can be anywhere in the world.
Classic Application Load Balancer	TLS is terminated on a GFE, which could be anywhere in the world.
Regional external Application Load Balancer	TLS is terminated on Envoy proxies located in a proxy-only subnet in a region chosen by the user. Use this load balancer mode if you need geographic control over the region where TLS is terminated.

Timeouts and retries

External Application Load Balancers support the following types of timeouts:

Timeout type and description	Default values	Supports custom values		
		Global	Classic	Regional
Backend service timeout (#timeout-bes)[1] A request and response timeout. Represents the maximum amount of time that can elapse from when the load balancer sends the first byte of the HTTP request to your backend to when your backend returns the last byte of the HTTP response. If the entire HTTP response has not been returned to the load balancer within the request or response timeout, the remaining response data is dropped. For WebSocket traffic, the maximum duration of the WebSocket, whether idle or active.	• For serverless NEGs on a backend service: 60 minutes • For all other backend types on a backend service: 30 seconds • For backend buckets: 24 hours (86,400 seconds)	✓	✓	✓
Client HTTP keepalive timeout (#http-keepalive-timeout) The amount of time that the load balancer keeps a TCP connection open for multiple HTTP requests from the same client to one of the following: • A first-layer Google Front End for a global external Application Load Balancer or a classic Application Load Balancer • A managed Envoy proxy for a regional external Application Load Balancer	• For a global external Application Load Balancer and a classic Application Load Balancer: 610 seconds • For a regional external Application Load Balancer: 600 seconds	✓ [2]	🚫	🚫
Backend HTTP keepalive timeout (#timeout-keepalive-backends) The amount of time that the load balancer keeps a TCP connection open for multiple requests to a common backend from one of the following types of proxies: • A second-layer Google Front End for a global external Application Load Balancer or a classic Application Load Balancer	• For backend services: 10 minutes (600 seconds) • For backend buckets: 6 minutes (360 seconds)	🚫	🚫	🚫

Timeout type and description	Default values	Supports custom values		
		Global	Classic	Regional
• A managed Envoy proxy for a regional external Application Load Balancer				

[1]Not configurable for serverless NEG backends. Not configurable for backend buckets.

[2]Configuration support for global external Application Load Balancers is not applicable to WebSocket.

Backend service timeout

The configurable *backend service timeout* represents the maximum amount of time that the load balancer waits for your backend to process an HTTP request and return the corresponding HTTP response. Except for serverless NEGs, the default value for the backend service timeout is 30 seconds.

For example, if you want to download a 500-MB file, and the value of the backend service timeout is 90 seconds, the load balancer expects the backend to deliver the entire 500-MB file within 90 seconds. It is possible to configure the backend service timeout to be insufficient for the backend to send its complete HTTP response. In this situation, if the load balancer has at least received HTTP response headers from the backend, the load balancer returns the complete response headers and as much of the response body as it could obtain within the backend service timeout.

Except for WebSockets, you should set the backend service timeout to the longest amount of time that you expect your backend to need in order to process an HTTP response. For WebSockets, the backend service timeout has a different meaning; it represents the maximum possible duration of a WebSocket, idle or active. You should increase the backend service timeout if the following is true:

- The software running on your backend needs more time to process an HTTP request and return its entire response.
- You see an HTTP `408` response with `jsonPayload.statusDetail client_timed_out`.
- You need WebSocket connections to persist for more time.

The backend service timeout accepts values between `1` and `2,147,483,647` seconds; however, larger values are not practical configuration options. Google Cloud does not guarantee that an underlying TCP connection can remain open for the entirety of the value of the backend service timeout. Client systems must implement retry logic instead of relying on a TCP connection to be open for long periods of time.

- For a global external Application Load Balancer or a classic Application Load Balancer, backend service timeout values larger than one day (86,400 seconds) are not recommended because Google Cloud periodically restarts Google Front Ends for software updates and other routine maintenance. Your backend service timeout value does not delay Google maintenance activities. The longer the backend service timeout value, the more likely it is that Google will terminate TCP connections for maintenance.
- For a regional external Application Load Balancer, Google Cloud periodically restarts or changes the number of serving Envoy software tasks. The longer the backend service timeout value, the more likely it is that Envoy task restarts or replacements will terminate TCP connections.

Regional external Application Load Balancers support a URL map `routeActions.timeout` (/compute/docs/reference/rest/v1/regionUrlMaps) parameter, which can override the backend service timeout. When `routeActions.timeout` is omitted, the value of the backend service timeout is used. When `routeActions.timeout` is supplied, the backend service timeout is ignored, and the value of `routeActions.timeout` is used as the request and response timeout instead.

To configure the backend service timeout, use one of the following methods:

- Google Cloud console: Modify the **Timeout** field of the load balancer's backend service.
- Google Cloud CLI: Use the `gcloud compute backend-services update` command (/sdk/gcloud/reference/compute/backend-services/update#--timeout) to modify the `--timeout` parameter of the backend service resource.
- API: For a global external Application Load Balancer or a classic Application Load Balancer, modify the `timeoutSec` parameter for the global `backendServices` resource (/compute/docs/reference/rest/v1/backendServices).
- API: For a regional external Application Load Balancer, modify the `timeoutSec` parameter for the `regionBackendServices` resource (/compute/docs/reference/rest/v1/regionBackendServices).

Client HTTP keepalive timeout

The *client HTTP keepalive timeout* represents the maximum amount of time that a TCP connection can be idle between the (downstream) client and one of the following types of proxies:

- For a global external Application Load Balancer or a classic Application Load Balancer: a first-layer Google Front End
- For a regional external Application Load Balancer: an Envoy proxy

An HTTP keepalive timeout represents the TCP idle timeout for the underlying TCP connections. The client HTTP keepalive timeout does not apply to WebSockets; instead, the backend service timeout (#timeout-bes) defines a total duration for the WebSockets.

- For a global external Application Load Balancer, the default value is 610 seconds. You can configure the client HTTP keepalive timeout with a value between 5 and 1200 seconds.
- For a classic Application Load Balancer, the client HTTP keepalive timeout is fixed at 610 seconds.
- For a regional external Application Load Balancer, the client HTTP keepalive timeout is fixed at 600 seconds.

To configure the keepalive timeout parameter, use one of the following methods:

- Google Cloud console: Modify the **HTTP keepalive timeout** field of the load balancer's frontend configuration.
- Google Cloud CLI: Use the `gcloud compute target-http-proxies update` command (/sdk/gcloud/reference/compute/target-http-proxies/update) or the `gcloud compute target-https-proxies update` command (/sdk/gcloud/reference/compute/target-https-proxies/update) to modify the `--http-keep-alive-timeout-sec` parameter of the target HTTP proxy or the target HTTPS proxy resource.
- API: Modify the `httpKeepAliveTimeoutSec` parameter for the `targetHttpProxies` resource (/compute/docs/reference/rest/v1/targetHttpProxies) or the `targetHttpsProxies` resource (/compute/docs/reference/rest/v1/targetHttpsProxies).

The load balancer's client HTTP keepalive timeout should be greater than the HTTP keepalive (TCP idle) timeout used by downstream clients or proxies. If a downstream client has a greater HTTP keepalive (TCP idle) timeout than the load balancer's client HTTP keepalive timeout, it's possible for a race condition to occur. From the perspective of a downstream client, an established TCP connection is permitted to be idle for longer than permitted by the load balancer. This means that the downstream client can send packets after the load balancer considers the TCP connection to be closed. When that happens, the load balancer responds with a TCP reset (RST) packet.

Backend HTTP keepalive timeout

External Application Load Balancers are proxies that use at least two TCP connections:

- For a global external Application Load Balancer or a classic Application Load Balancer, a first TCP connection exists between the (downstream) client and a first-layer GFE. First-layer GFEs connect to second layer GFEs, and then the second-layer GFEs open a second TCP connection to your backends.
- For a regional external Application Load Balancer, a first TCP connection exists between the (downstream) client and an Envoy proxy. The Envoy proxy then opens a second TCP connection to your backends.

The load balancer's secondary TCP connections might not get closed after each request; they can stay open to handle multiple HTTP requests and responses. The *backend HTTP keepalive timeout* defines the TCP idle timeout between the load balancer and your backends. The backend HTTP keepalive timeout does not apply to WebSockets; instead, the backend service timeout (#timeout-bes) defines a total duration for the WebSocket.

The backend keepalive timeout is fixed at 10 minutes (600 seconds) and cannot be changed. The load balancer's backend keepalive timeout should be less than the keepalive timeout used by software running on your backends. This avoids a race condition where the operating system of your backends might close TCP connections with a TCP reset (RST). Because the backend keepalive timeout for the load balancer is not configurable, you must configure your backend software so that its HTTP keepalive (TCP idle) timeout value is greater than 600 seconds.

The following table lists the changes necessary to modify keepalive timeout values for common web server software.

Web server software	Parameter
Apache (https://httpd.apache.org)	KeepAliveTimeout (https://httpd.apache.org/docs/2.4/mod/core.html#keepalivetimeout)
nginx (https://nginx.org)	keepalive_timeout (http://nginx.org/en/docs/http/ngx_http_core_module.html#keepalive_timeo

Retries

Support for retry logic depends on the mode of the external Application Load Balancer.

Load balancer mode	Retry logic
Global external Application Load Balancer	Configurable by using a retry policy (/compute/docs/reference/rest/beta/urlMaps) in the URL map. The default number of retries (`numRetries`) is 1. The maximum number of retries that can be configured by using the retry policy is 25. The default timeout for each try (`perTryTimeout`) is 30 seconds with a maximum configurable `perTryTimeout` of 24 hours. Without a retry policy, unsuccessful requests that have no HTTP body (for example, `GET` requests) that result in HTTP `502`, `503`, or `504` responses (`retryConditions=["gateway-error"]`) are retried once. HTTP `POST` requests are not retried. Retried requests only generate one log entry for the final response.
Classic Application Load Balancer	The retry policy cannot be changed for connection retries. HTTP `POST` requests are not retried. HTTP `GET` requests are always retried once as long as 80% or more of the backends are healthy. If there is a single backend instance in a group and the connection to that backend instance fails, the percentage of unhealthy backend instances is 100%, so the GFE doesn't retry the request. The load balancer retries a failed `GET` request if the first request failed before receiving response headers from the backend instance. Retried requests only generate one log entry for the final response. For more information, see External Application Load Balancer logging and monitoring (/load-balancing/docs/https/https-logging-monitoring). Unsuccessful requests result in the load balancer synthesizing an HTTP `502` response.
Regional external Application Load Balancer	Configurable by using a retry policy (/compute/docs/reference/rest/beta/regionUrlMaps) in the URL map. The default number of retries (`numRetries`) is 1. The maximum number of retries that can be configured by using the retry policy is 25. The default timeout for each try (`perTryTimeout`) is 30 seconds with a maximum configurable `perTryTimeout` of 24 hours. Without a retry policy, unsuccessful requests that have no HTTP body (for example, `GET` requests) that result in HTTP `502`, `503`, or `504` responses are retried once. HTTP `POST` requests are not retried. Retried requests only generate one log entry for the final response.

The WebSocket protocol is supported with GKE Ingress (/kubernetes-engine/docs/concepts/ingress-xlb#support_for_websocket).

Illegal request and response handling

The 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 Application Load Balancers (/load-balancing/docs/quotas#https-lb-header-limits).
- 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 Application Load Balancers.

- The HTTP version is unknown.

Traffic distribution

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 Application Load Balancers support 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.

How traffic is distributed among backends depends on the mode of the load balancer.

Global external Application Load Balancer

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 the global external Application Load Balancer, load balancing is two-tiered. The balancing mode determines the weighting or fraction of traffic that should be sent to each backend (instance group or NEG). Then, the load balancing policy (`LocalityLbPolicy`) determines how traffic is distributed to instances or endpoints within the group. For more information, see the Load balancing locality policy (backend service API documentation) (/compute/docs/reference/rest/v1/backendServices#BackendService.FIELDS.locality_lb_policy).

For the classic Application Load Balancer, the balancing mode is used to select the most favorable backend (instance group or NEG). Traffic is then distributed in a round robin fashion among instances or endpoints within the backend.

How requests are distributed

GFE-based external Application Load Balancers use the following process to distribute incoming requests:

1. **From client to first-layer GFE.** Edge routers advertise the forwarding rule's external IP address at the borders of Google's network. Each advertisement lists a next hop to a Layer 3/4 load balancing system (Maglev). The Maglev systems route traffic to a first-layer Google Front End (GFE).
   - *When using Premium Tier*, Google advertises your load balancer's IP address from all points of presence, worldwide. Each load balancer IP address is global anycast.
   - *When using Standard Tier*, Google advertises your load balancer's IP address from points of presence associated with the forwarding rule's region. The load balancer uses a regional external IP address. Using a Standard Tier forwarding rule limits you to instance group and zonal NEG backends in the same region as the load balancer's forwarding rule.
2. **From first-layer GFE to second-layer GFE.** The first-layer GFE terminates TLS if required and then routes traffic to second-layer GFEs according to the following process:
   - First-layer GFEs parse the URL map and select a backend service or backend bucket.
   - For backend services with internet NEGs, the first layer-GFEs select a second-layer external forwarding gateway colocated with the first-layer GFE. The forwarding gateway sends requests to the internet NEG endpoint. This concludes the request distribution process for internet NEGs.
   - For backend services with serverless NEGs (/load-balancing/docs/negs/serverless-neg-concepts), Private Service Connect (PSC) NEGs (/load-balancing/docs/negs#psc-neg), and backend buckets, first-layer GFEs select a second-layer GFE in the region matching the region of the NEG or bucket. For multi-region Cloud Storage buckets, first-layer GFEs select second-layer GFEs in a region as close as possible to the first-layer GFEs (defined by network round trip time).

- For backend services with instance groups, zonal NEGs with `GCE_VM_IP_PORT` endpoints (/load-balancing/docs/negs/zonal-neg-concepts), and hybrid NEGs (/load-balancing/docs/negs/hybrid-neg-concepts), Google's capacity management system informs first-layer GFEs about the used and configured capacity for each backend. The configured capacity for a backend is defined by the balancing mode (/load-balancing/docs/backend-service#balancing-mode), the target capacity (/load-balancing/docs/backend-service#target-capacity) of the balancing mode, and the capacity scaler (/load-balancing/docs/backend-service#capacity_scaler).
    - *Standard Tier:* First-layer GFEs select a second layer GFE in the region containing the backends.
    - *Premium Tier:* First-layer GFEs select second-layer GFEs from a set of applicable regions. Applicable regions are all regions where backends have been configured, excluding those regions with configured backends having zero capacity. First-layer GFEs select the closest second-layer GFE in an applicable region (defined by network round-trip time). If backends are configured in two or more regions, first-layer GFEs can spill requests over to other applicable regions if a first-choice region is full. Spillover to other regions is possible when all backends in the first-choice region are at capacity.

3. **Second layer GFEs select backends.** Second-layer GFEs are located in zones of a region. They use the following process to select a backend:
    - For backend services with serverless NEGs, Private Service Connect NEGs, and backend buckets, second-layer GFEs forward requests to Google's production systems. This concludes the request distribution process for these backends.
    - For backend services with instance groups, zonal NEGs with `GCE_VM_IP_PORT` endpoints, and hybrid NEGs, Google's health check probe systems inform second-layer GFEs about the health check status of the backend instances or endpoints.

      Premium Tier only: If the second-layer GFE has no healthy backend instances or endpoints in its region, it might send requests to another second-layer GFE in a different applicable region with configured backends. Spillover between second-layer GFEs in different regions does not exhaust all possible region-to-region combinations. If you need to direct traffic away from backends in a particular region, instead of configuring backends to fail health checks, you should set the capacity scaler of the backend to zero (/load-balancing/docs/backend-service#capacity_scaler) so the first-layer GFE excludes the region during the previous step.

The second-layer GFE then directs requests to backend instances or endpoints in zones within its region as discussed in the next step.

4. **Second layer GFE selects a zone.** By default, second-layer GFEs use the `WATERFALL_BY_REGION` algorithm where each second-layer GFE prefers to select backend instances or endpoints in the same zone as the zone that contains the second-layer GFE. Because `WATERFALL_BY_REGION` minimizes traffic between zones, at low request rates, each second-layer GFE might exclusively send requests to backends in the same zone as the second-layer GFE itself.

   For global external Application Load Balancers only, second-layer GFEs can be configured to use one of the following alternative algorithms by using a `serviceLbPolicy` (/load-balancing/docs/service-lb-policy):

   - `SPRAY_TO_REGION`: Second-layer GFEs do not prefer selecting backend instances or endpoints in the same zone as the second-layer GFE. Second-layer GFEs attempt to distribute traffic to all backend instances or endpoints in all zones of the region. This can lead to more even distribution of load at the expense of increased traffic between zones.
   - `WATERFALL_BY_ZONE`: Second-layer GFEs strongly prefer selecting backend instances or endpoints in the same zone as the second-layer GFE. Second-layer GFEs only direct requests to backends in different zones after all backends in the current zone have reached their configured capacities.

5. **Second layer GFE selects instances or endpoints within the zone.** By default, a second-layer GFE distributes requests among backends in a round-robin fashion. For global external Application Load Balancers only, you can change this by using a load balancing locality policy (`localityLbPolicy`). The load balancing locality policy applies only to backends within the selected zone discussed in the previous step.

Regional external Application Load Balancer

For regional external Application Load Balancers, traffic distribution is based on the load balancing mode and the load balancing locality policy.

The balancing mode determines the weight and fraction of traffic that should be sent to each group (instance group or NEG). The load balancing locality policy (`LocalityLbPolicy`) determines how backends within the group are load balanced.

When a backend service receives traffic, it first directs traffic to a backend (instance group or NEG) according to the backend's balancing mode. After a backend is selected, traffic is then distributed among instances or endpoints in that backend group according to the load balancing locality policy.

For more information, see the following:

- Balancing modes (/load-balancing/docs/backend-service#balancing-mode)
- Load balancing locality policy (regional backend service API documentation) (/compute/docs/reference/rest/beta/regionBackendServices).

Session affinity

Session affinity (/load-balancing/docs/backend-service#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.

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).

External Application Load Balancers offer the following types of session affinity:

- NONE. Session affinity is not set for the load balancer.
- Client IP affinity (/load-balancing/docs/backend-service#client_ip_affinity)
- Generated cookie affinity (/load-balancing/docs/backend-service#generated_cookie_affinity)
- Header field affinity (/load-balancing/docs/backend-service#header_field_affinity)
- HTTP Cookie affinity (/load-balancing/docs/backend-service#http_cookie_affinity)

The following table summarizes the supported session affinity options supported by external Application Load Balancers in each mode:

Load balancer mode	**Session affinity options**				
	None	**Client IP**	**Generated cookie**	**Header field**	**HTTP cookie**
Global external Application Load Balancer	✓	✓	✓	✓	✓
Classic Application Load Balancer	✓	✓	✓	🚫	🚫
Regional external Application Load	✓	✓	✓	✓	✓

Load balancer mode	Session affinity options
Balancer	

HTTP/2 support

HTTP/2 max concurrent streams

The HTTP/2 `SETTINGS_MAX_CONCURRENT_STREAMS` (https://tools.ietf.org/html/rfc7540#section-6.5.2) setting describes the maximum number of streams that an endpoint accepts, initiated by the peer. The value advertised by an HTTP/2 client to a Google Cloud load balancer is effectively meaningless because the load balancer doesn't initiate streams to the client.

In cases where the load balancer uses HTTP/2 to communicate with a server that is running on a VM, the load balancer respects the `SETTINGS_MAX_CONCURRENT_STREAMS` value advertised by the server. If a value of zero is advertised, the load balancer can't forward requests to the server, and this might result in errors.

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. As a result, you might see high backend latencies because backend connections are made more frequently.
- HTTP/2 between the load balancer and the backend does not support running the WebSocket Protocol over a single stream of an HTTP/2 connection (RFC 8441 (https://tools.ietf.org/html/rfc8441)).
- HTTP/2 between the load balancer and the backend does not support server push.
- The gRPC error rate and request volume aren't visible in the Google Cloud API or the Google Cloud console. If the gRPC endpoint returns an error, the load balancer logs and the monitoring data report the `OK 200` HTTP response code.

HTTP/3 support

HTTP/3 (https://datatracker.ietf.org/doc/rfc9114/) is a next-generation internet protocol. It is built on top of IETF QUIC (https://datatracker.ietf.org/doc/rfc9000/), a protocol developed from the original Google QUIC (https://www.chromium.org/quic) protocol. HTTP/3 is supported between the external Application Load Balancer, Cloud CDN, and clients.

Google QUIC (also known as gQUIC) support was removed in April 2023 in favor of HTTP/3 over IETF QUIC, which is both an IETF standard and has been shown to outperform gQUIC.

Specifically:

- IETF QUIC is a transport layer protocol that provides congestion control and reliability similar to TCP, uses TLS 1.3 for security, and improved performance.
- HTTP/3 is an application layer built on top of IETF QUIC, and it relies on QUIC to handle multiplexing, congestion control, loss detection, and retransmission.
- HTTP/3 allows faster client connection initiation, eliminates head-of-line blocking in multiplexed streams, and supports connection migration when a client's IP address changes.
- HTTP/3 is supported for connections between clients and the load balancer, not connections between the load balancer and its backends.
- HTTP/3 connections use the BBR (/blog/products/networking/tcp-bbr-congestion-control-comes-to-gcp-your-internet-just-got-faster) congestion control algorithm.

HTTP/3 on your load balancer can improve web page load times, reduce video rebuffering, and improve throughput on higher latency connections.

The following table specifies the HTTP/3 support for external Application Load Balancers in each mode.

Load balancer mode	HTTP/3 support
Global external Application Load Balancer (always Premium Tier)	✓
Classic Application Load Balancer in Premium Tier	✓
Classic Application Load Balancer in Standard Tier	🚫
Regional external Application Load Balancer (always Standard Tier)	🚫

How HTTP/3 is negotiated

When HTTP/3 is enabled, the load balancer advertises this support to clients, allowing clients that support HTTP/3 to attempt to establish HTTP/3 connections with the HTTPS load balancer.

- Properly implemented clients always fall back to HTTPS or HTTP/2 when they cannot establish an HTTP/3 connection.
- Clients that support HTTP/3 use their cached prior knowledge of HTTP/3 support to save unnecessary round-trips in the future.
- Because of this fallback, enabling or disabling HTTP/3 in the load balancer does not disrupt the load balancer's ability to connect to clients.

Support is advertised in the `Alt-Svc` (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Alt-Svc) HTTP response header. When HTTP/3 is enabled, responses from the load balancer include the following `alt-svc` header value:

```
alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000"
```

Note: The `Alt-Svc` header advertises multiple versions of HTTP/3 in order to support earlier drafts that are used by some clients (e.g. `h3-29`).

If HTTP/3 has been explicitly set to `DISABLE`, responses do not include an `alt-svc` response header.

When you have HTTP/3 enabled on your HTTPS load balancer, some circumstances can cause your client to fall back to HTTPS or HTTP/2 instead of negotiating HTTP/3. These include the following:

- When a client supports versions of HTTP/3 that are not compatible with the HTTP/3 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 HTTP/3 from working.
- The client does not support HTTP/3 at all, and thus does not attempt to negotiate an HTTP/3 connection.

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

Before you enable HTTP/3, ensure that the previously described behaviors are acceptable for your workloads.

Configure HTTP/3

Both `NONE` (the default) and `ENABLE` enable HTTP/3 support for your load balancer.

When HTTP/3 is enabled, the load balancer advertises it to clients, which allows clients that support it to negotiate an HTTP/3 version with the load balancer. Clients that do not support HTTP/3 do not negotiate an HTTP/3 connection. You do not need to explicitly disable HTTP/3 unless you have identified broken client implementations.

External Application Load Balancers provide three ways to configure HTTP/3 as shown in the following table.

quicOverride value	Behavior
`NONE`	Support for HTTP/3 is advertised to clients.
`ENABLE`	Support for HTTP/3 is advertised to clients. Note: TLS 0-RTT (also known as *TLS Early Data*) is not yet supported for HTTP/3.
`DISABLE`	Explicitly disables advertising HTTP/3 and Google QUIC to clients.

To explicitly enable (or disable) HTTP/3, follow these steps.

Console: HTTPS (#console:-https) gcloud: HTTPS ... API: HTTPS (#api:-https)

Note: Setting the HTTP/3 negotiation isn't currently supported on the target proxies page and must be configured by editing the load balancer configuration.

1. In the Google Cloud console, go to the **Load balancing** page.

   Go to Load balancing (https://console.cloud.google.com/networking/loadbalancing/add)

2. Select the load balancer that you want to edit.

3. Click **Frontend configuration**.

4. Select the frontend IP address and port that you want to edit. To edit an HTTP/3 configuration, the protocol must be HTTPS.

Enable HTTP/3

1. Select the **QUIC negotiation** drop-down.
2. To explicitly enable HTTP/3 for this frontend, select **Enabled**.
3. If you have multiple frontend rules representing IPv4 and IPv6, make sure to enable HTTP/3 on each rule.

Disable HTTP/3

1. Select the **QUIC negotiation** drop-down.
2. To explicitly disable HTTP/3 for this frontend, select **Disabled**.
3. If you have multiple frontend rules representing IPv4 and IPv6, make sure to disable HTTP/3 for each rule.

Limitations

- HTTPS load balancers do not send a `close_notify` closure alert (https://tools.ietf.org/html/rfc5246#section-7.2.1) when terminating SSL connections. That is, the load balancer closes the TCP connection instead of performing an SSL shutdown.
- HTTPS load balancers support only lowercase characters in domains in a common name (`CN`) attribute or a subject alternative name (`SAN`) attribute of the certificate. Certificates with uppercase characters in domains are returned only when set as the primary certificate (/load-balancing/docs/ssl-certificates#multiplessl) in the target proxy.
- HTTPS load balancers do not use the Server Name Indication (SNI) extension when connecting to the backend, except for load balancers with Internet NEG backends (/load-balancing/docs/negs/internet-neg-concepts#support_for_the_ssl_server_name_indication_sni_extension) . For more details, see Encryption from the load balancer to the backends (/load-balancing/docs/ssl-certificates/encryption-to-the-backends).
- When using regional external Application Load Balancers with Cloud Run in a Shared VPC environment, standalone VPC networks in service projects can send traffic to any other Cloud Run services deployed in any other service projects within the same

Shared VPC environment. This is a known issue and this form of access will be blocked in the future.

- Google Cloud doesn't guarantee that an underlying TCP connection can remain open for the entirety of the value of the backend service timeout. Client systems must implement retry logic instead of relying on a TCP connection to be open for long periods of time.

What's next

- To learn how to deploy a global external Application Load Balancer, see Setting up an external Application Load Balancer with a Compute Engine backend (/load-balancing/docs/https/ext-http-lb-simple).
- To learn how to deploy a regional external Application Load Balancer, see Setting up a regional external Application Load Balancer with a Compute Engine backend (/load-balancing/docs/https/setting-up-reg-ext-https-lb).
- If you are an existing user of the classic Application Load Balancer, make sure that you review Plan your migration to the global external Application Load Balancer (/load-balancing/docs/https/migrate-to-global) when you plan a new deployment with the global external Application Load Balancer.
- To learn how to automate your external Application Load Balancer setup with Terraform, see Terraform module examples for external Application Load Balancers (/load-balancing/docs/https/ext-http-lb-tf-module-examples).
- To learn how to configure advanced traffic management capabilities available with the global external Application Load Balancer, see Traffic management overview for global external Application Load Balancers (/load-balancing/docs/https/traffic-management-global).
- To learn how to configure advanced traffic management capabilities available with the regional external Application Load Balancer, see Traffic management overview for regional external Application Load Balancers (/load-balancing/docs/https/traffic-management-regional).
- To find the locations for Google PoPs, see GFE locations (/load-balancing/docs/locations).
- To learn about capacity management, see Capacity management with load balancing tutorial (/load-balancing/docs/tutorials/capacity-management-with-load-balancing) and

Application capacity optimizations with global load balancing (/load-balancing/docs/tutorials/about-capacity-optimization-with-global-lb).

- To learn about serving websites, see Serving websites (/solutions/web-serving-overview).
- To learn how to use Certificate Manager to provision and manage SSL certificates, see the Certificate Manager overview (/certificate-manager/docs/overview).
- To inject custom logic into the load balancing data path, configure Service Extensions callouts (/service-extensions/docs/configure-callout).

Last updated 2023-12-08 UTC.