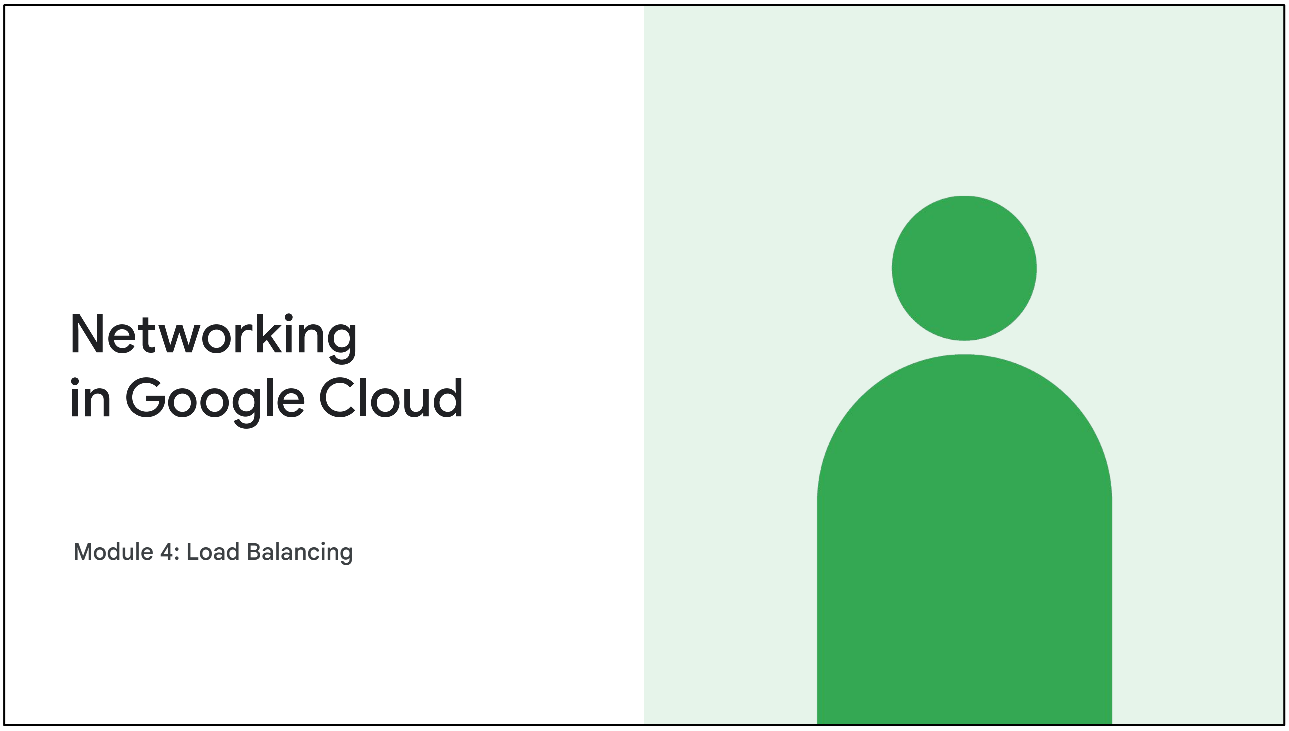
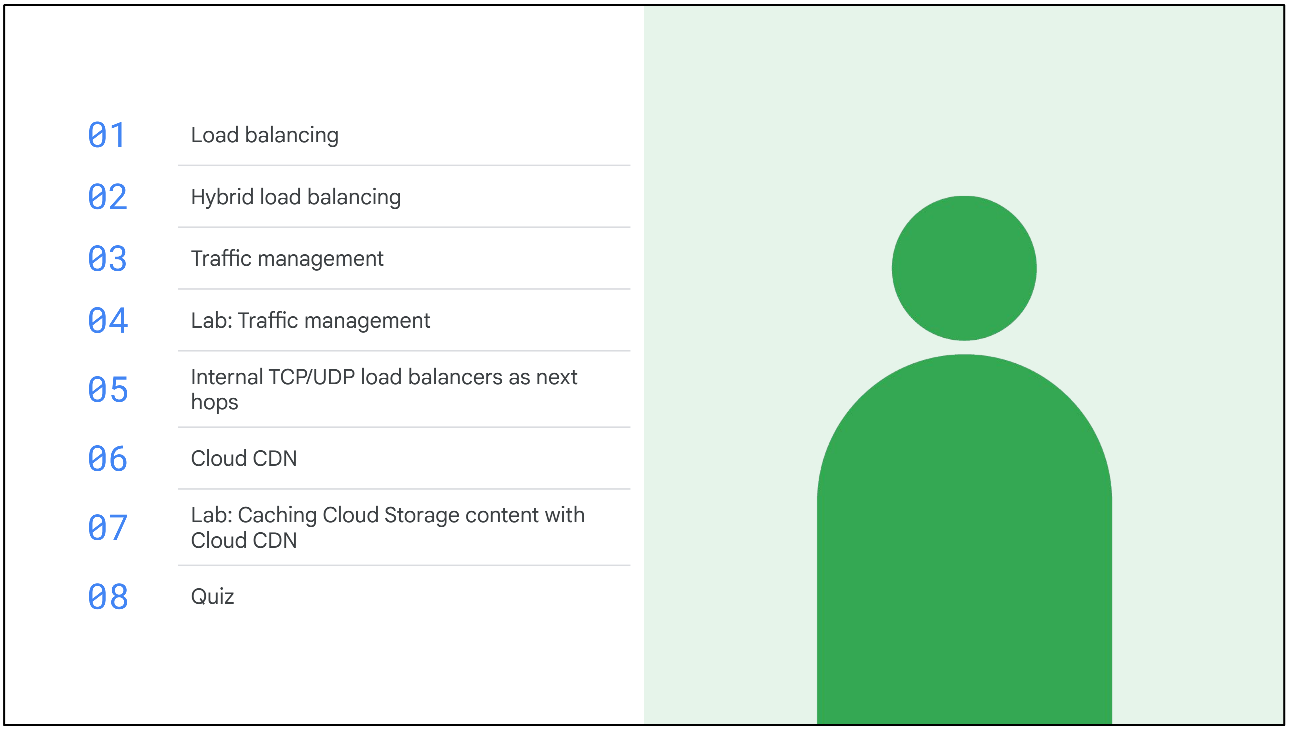


Networking in Google Cloud

Module 4: Load Balancing

Welcome to the Load Balancing module. This is the fourth and final module of the Networking in Google Cloud: Defining and Implementing Networks course.



In this module, we will cover the topics listed on the screen.

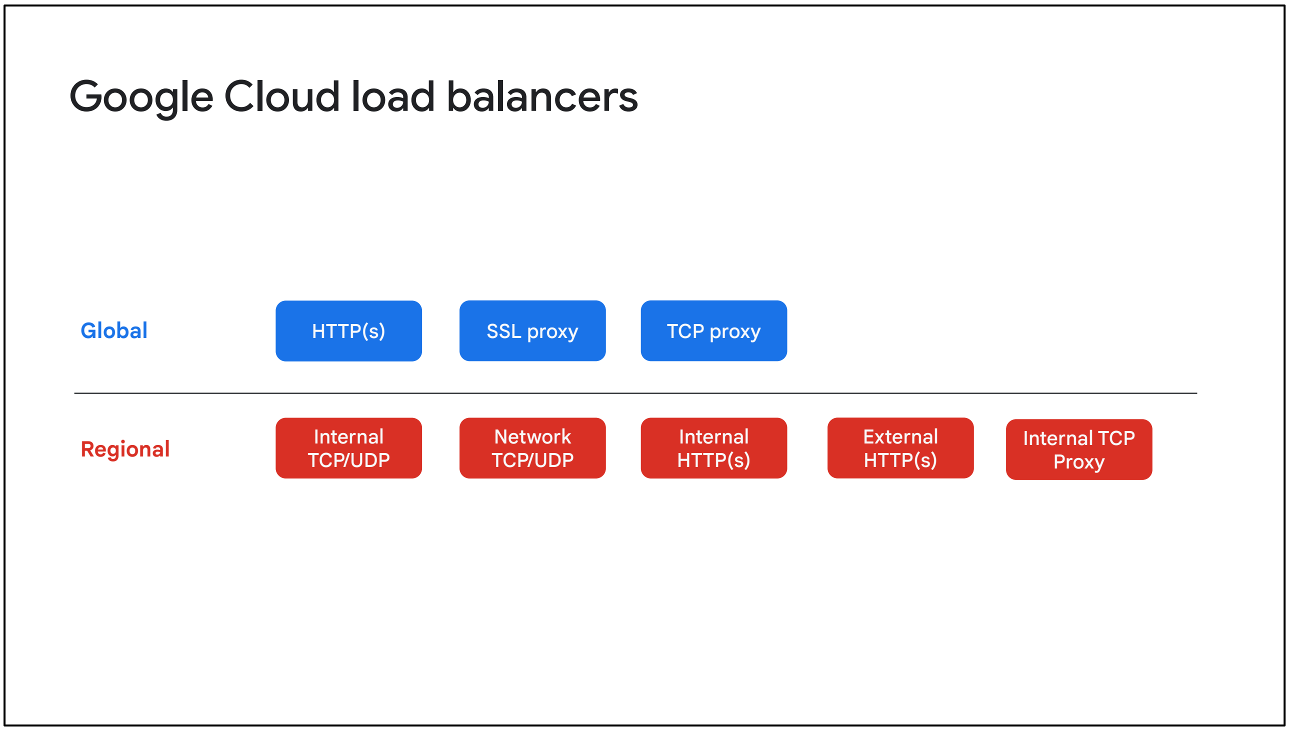
We'll begin with an overview of Google Cloud load balancing. We will continue with a discussion of hybrid load balancing - in other words, load balancing between Google Cloud, other public clouds, and on-premises environments.

We will follow with a discussion of traffic management, which provides enhanced features to route traffic based on criteria that you specify. After that, you will apply what you've learned in a traffic management lab exercise.

Next, we'll discuss using Internal TCP/UDP load balancers as next hops, including benefits, caveats, and some use cases.

We'll continue by discussing how to use the Google global edge network to serve content closer to users with the Cloud Content Delivery Network (CDN). We'll follow that with a Cloud CDN lab exercise and a brief quiz.

Let's get started!



Google Cloud load balancers

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

The global load balancers are the HTTP(S), SSL proxy, and TCP proxy load balancers. These load balancers leverage the Google Front End (GFE) service. GFE is a software-defined, distributed system that is available from Google points of presence and is distributed globally. The global load balancer manages redundancy, such as routing a request to a nearby region when the desired region is unavailable.

Use a global load balancer when your users and instances are globally distributed, you need access to the same workloads and content, and you want to use a single anycast IP address to provide access.

The regional load balancers are the internal TCP/UDP, the network TCP/UDP, the internal HTTP(S), and the external HTTP(s) load balancers. These load balancers distribute traffic to instances that are in a single Google Cloud region.

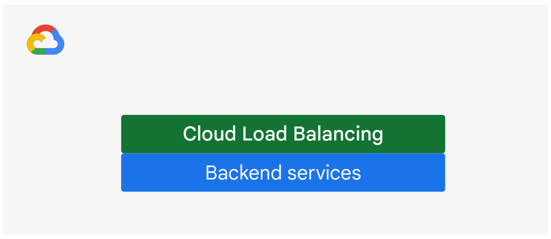
The internal load TCP/UDP balancer uses Andromeda, which is a a network virtualization stack that is software-defined and made by Google Cloud.

The network load balancer uses Maglev, which is a large, distributed software system.

The internal HTTP(S) load balancer is a proxy-based Layer 7 load balancer. This load balancer lets you run and scale your services behind a private load balancing IP address. The IP address is accessible only in the region where the load balancer is located.

Overview of Cloud Load Balancing

- Cloud Load Balancing receives client traffic.
- Backend services define:
  - How the traffic is distributed.
  - Which health check to use.
  - If session affinity is used.
  - Which other services are used (such as Cloud CDN or Identity-Aware Proxy).



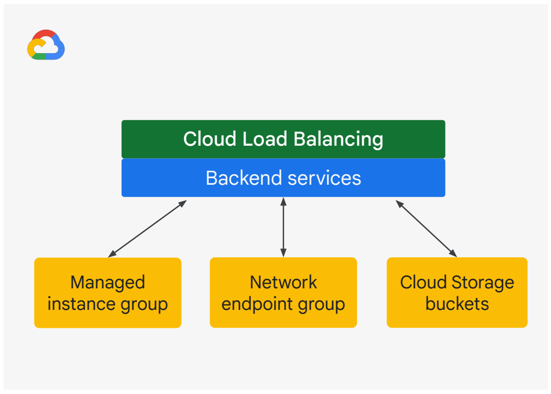
Cloud Load Balancing receives client traffic. This traffic can be external or external, depending on the load balancer you use.

The backend services define how to handle the traffic. For example, backend services define how the traffic is distributed, which health check to use, and if session affinity is used. Backend services also define which other Google Cloud services to use, such as Cloud CDN or Identity-Aware Proxy.

Overview of Cloud Load Balancing

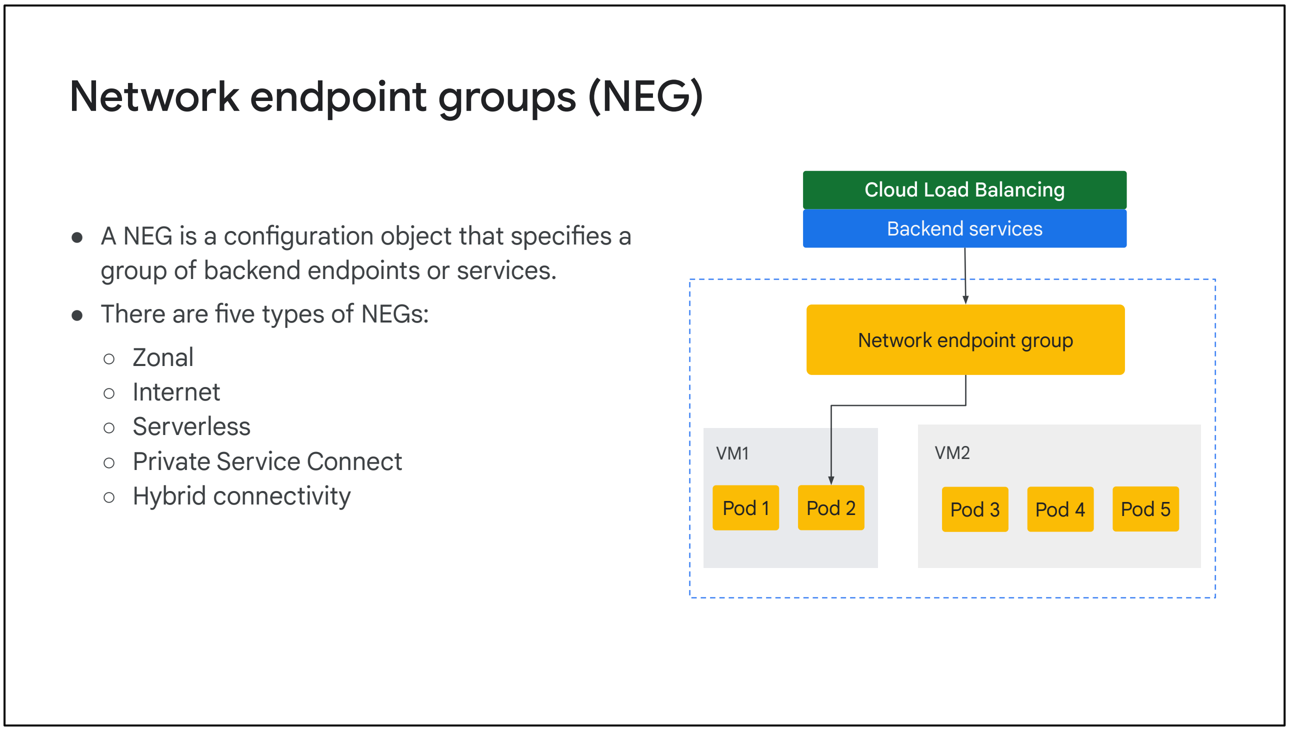
Cloud Load Balancing can route traffic to:

- Managed instance groups: a group of virtual machines created from a template.
- Network endpoint groups (NEG): a group of services or workloads.
- Cloud Storage buckets



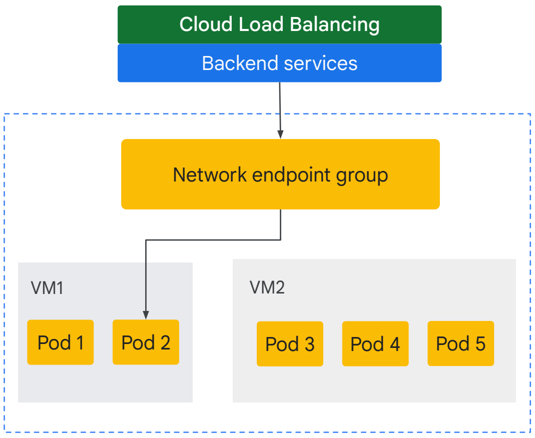
In previous training, you learned that Cloud Load Balancing can route traffic to a managed instance group, a network endpoint group (NEG), or Cloud Storage buckets.

In this module, we are going to look at some special features related to network endpoint groups.



Network endpoint groups (NEG)

- A NEG is a configuration object that specifies a group of backend endpoints or services.
- There are five types of NEGs:
  - Zonal
  - Internet
  - Serverless
  - Private Service Connect
  - Hybrid connectivity



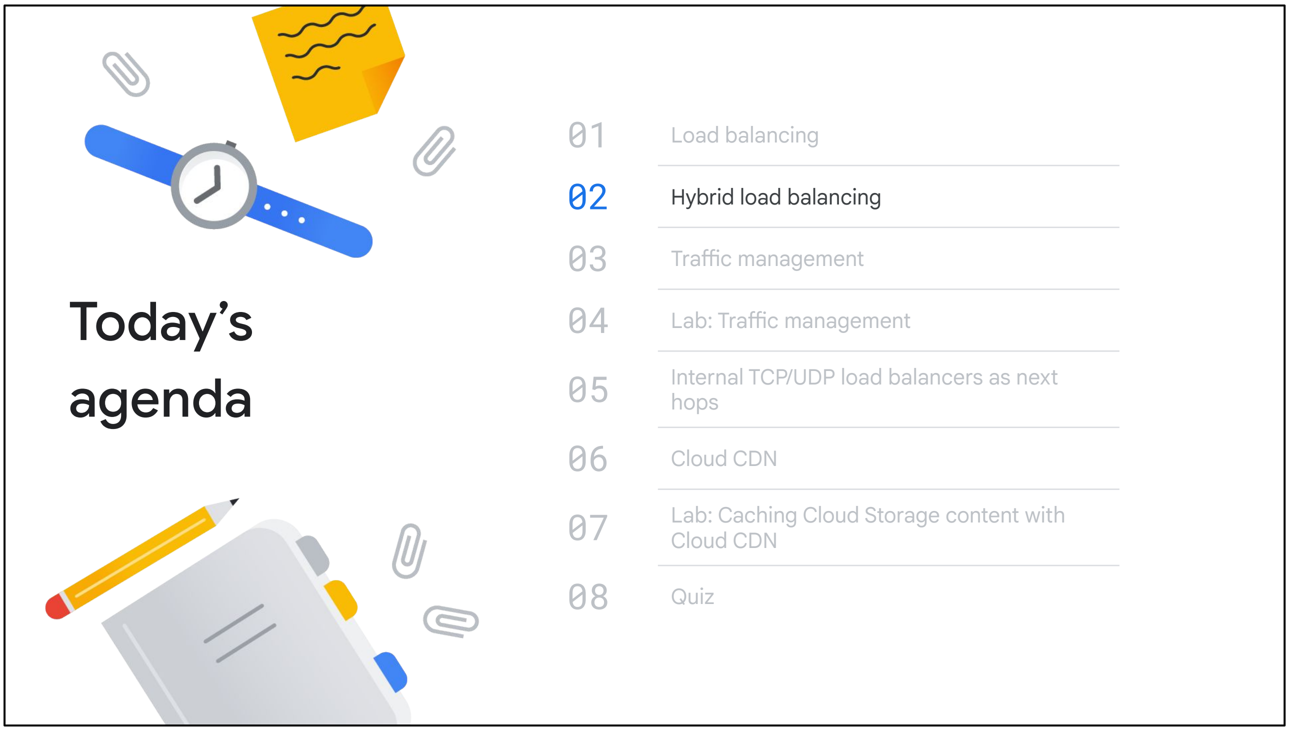
A network endpoint group (NEG) is a configuration object that specifies a group of backend endpoints or services. A common use case for this configuration is deploying services in containers, as in Google Kubernetes Engine. The load balancer must be able to select a pod from the container, as shown in the example. You can also distribute traffic in a granular fashion to workloads and services that run on your backend hosts.

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

- **Zonal and internet NEGs** define how endpoints should be reached, whether they are reachable, and where they are located. **A zonal NEG** contains one or more endpoints that can be Compute Engine virtual machines (VMs) or services that run on the VMs. Each endpoint is specified either by an IP address or an IP:port combination.
- **An internet NEG** contains a single endpoint that is hosted outside of Google Cloud. This endpoint is specified by hostname FQDN:port or IP:port.
- **A serverless NEG** is a backend that points to a Cloud Run, App Engine, Cloud Functions, or API Gateway service that reside in the same region as the NEG.
- **A Private Service Connect NEG** contains a single endpoint. That endpoint that resolves to either a Google-managed regional API endpoint or a managed service published by using Private Service Connect.
- **A hybrid connectivity NEG** points to Traffic Director services that run outside of Google Cloud (on-premises or other public cloud backends). The focus in

- this module is on hybrid connectivity NEGs.

For more information on using NEGs, and a complete list of supported load balancers, please refer to the Network endpoint groups overview in the Google Cloud documentation.



Today's agenda

Next, let's discuss hybrid load balancing, which lets you balance traffic between Google Cloud, on-premises environments, other public cloud providers.

Hybrid connectivity and load balancing

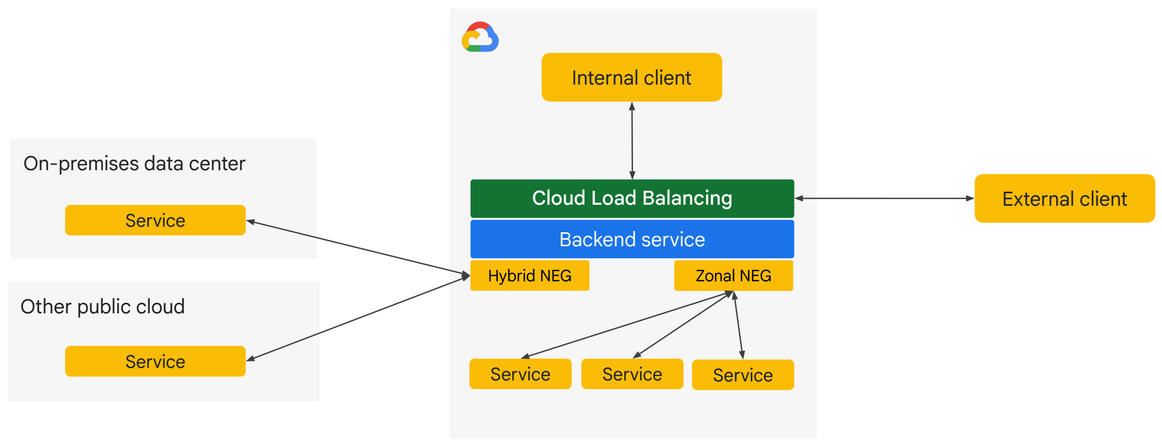
- A hybrid strategy lets you extend Cloud Load Balancing to workloads that run on your existing infrastructure outside of Google Cloud.
- This strategy could be:
  - Permanent to provide multiple platforms for your workloads.
  - Temporary as you prepare to migrate your external workloads to Google Cloud.

A hybrid load balancing lets you extend Cloud Load Balancing to workloads that run on your existing infrastructure outside of Google Cloud.

A hybrid strategy is a pragmatic solution for you to adapt to changing market demands and incrementally modernize the backend services that run your workloads. You can create a hybrid deployment to enable migration to a modern cloud-based solution or a permanent fixture of your organization IT infrastructure.

Next, let’s look at a few general examples of hybrid load balancing.

Hybrid load balancing



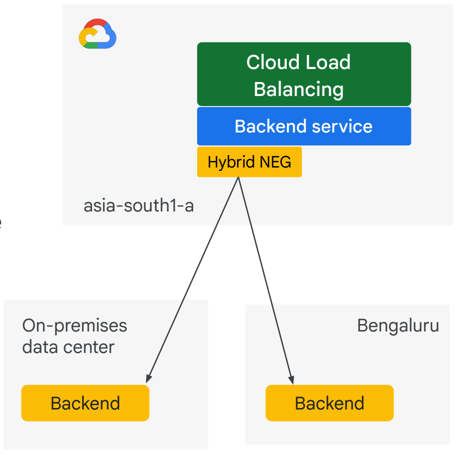
In this example, traffic from clients on the public internet enters your private on-premises environment, and traffic from another public cloud provider enters through a Google Cloud load balancer. The load balancer also gets requests from internal clients.

The load balancer sends requests to the services that run your workloads. These services are the load balancer endpoints, and they can be located inside or outside of Google Cloud. You configure a load balancer backend service to communicate to the external endpoints by using a hybrid NEG. The external environments can use Cloud Interconnect or Cloud VPN to communicate with Google Cloud. The load balancer must be able to reach each service with a valid IP address:Port combination.

The example shows a load balancer backend service with a hybrid and a zonal NEG. The hybrid NEG connects to endpoints that are on-premises and in other public clouds. The zonal NEG points to Cloud Endpoints in the same subnet and zone.

Configuring backend services outside of Google Cloud

- Configure one or more hybrid connectivity network endpoint groups (NEG):
  - Add the IP address:Port for each backend service to a hybrid connectivity NEG.
  - Specify a Google Cloud zone that is as close as possible to your other environment.
  - Add a health check to the NEG.
- Add the hybrid connectivity NEGs to a hybrid load balancer backend service.

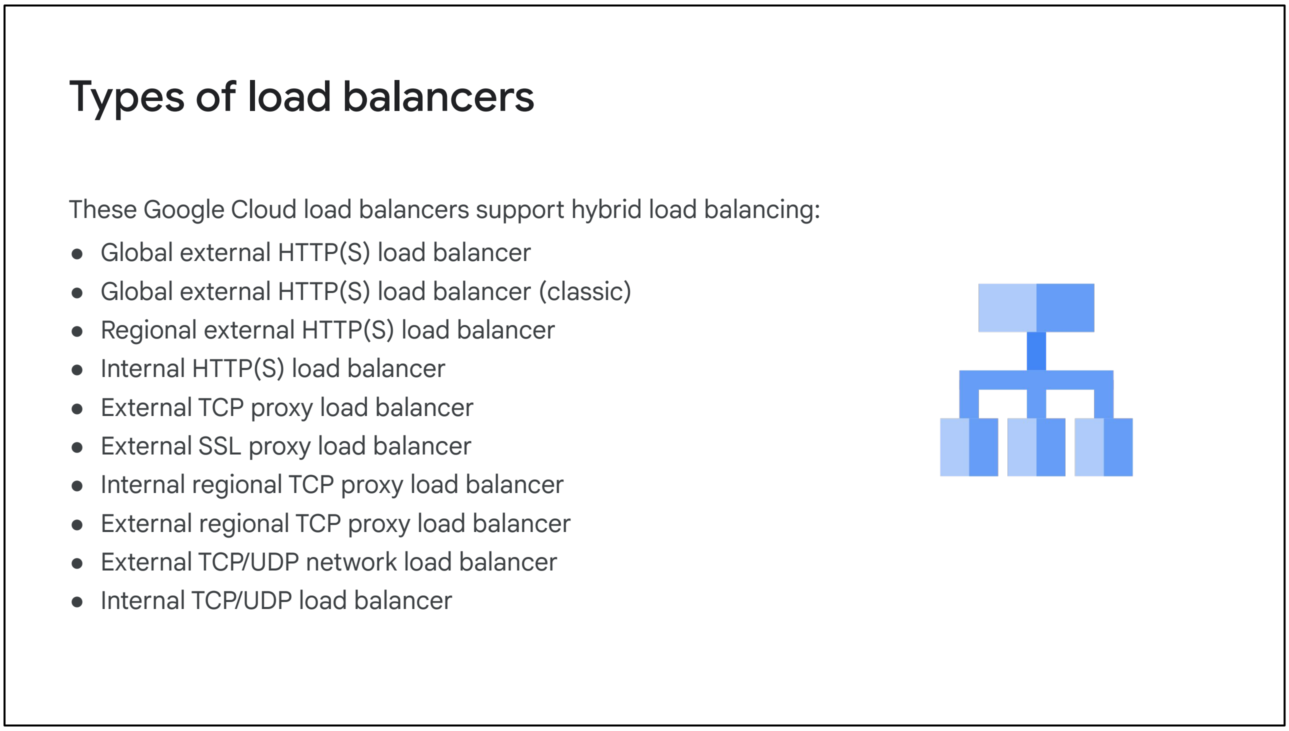


To configure backend services outside of Google Cloud, first configure one or more hybrid connectivity network endpoint groups (NEG).

Add each non-Google Cloud network endpoint IP:Port combination to a hybrid connectivity network endpoint group (NEG). Ensure that the IP address and port are reachable from Google Cloud. For hybrid connectivity NEGs, you set the network endpoint type to NON_GCP_PRIVATE_IP_PORT.

Create the NEG in a Google Cloud zone that is as close as possible to your other environment. For example, if you're hosting a service in an on-premises environment in Bengaluru, India, you can place the NEG in the asia-south1-a Google Cloud zone, as shown in the example.

Add the hybrid connectivity NEGs to a hybrid load balancer backend. A hybrid connectivity NEG must only include endpoints outside Google Cloud. Traffic might be dropped if a hybrid NEG includes endpoints for resources within a Google Cloud VPC network.



Types of load balancers

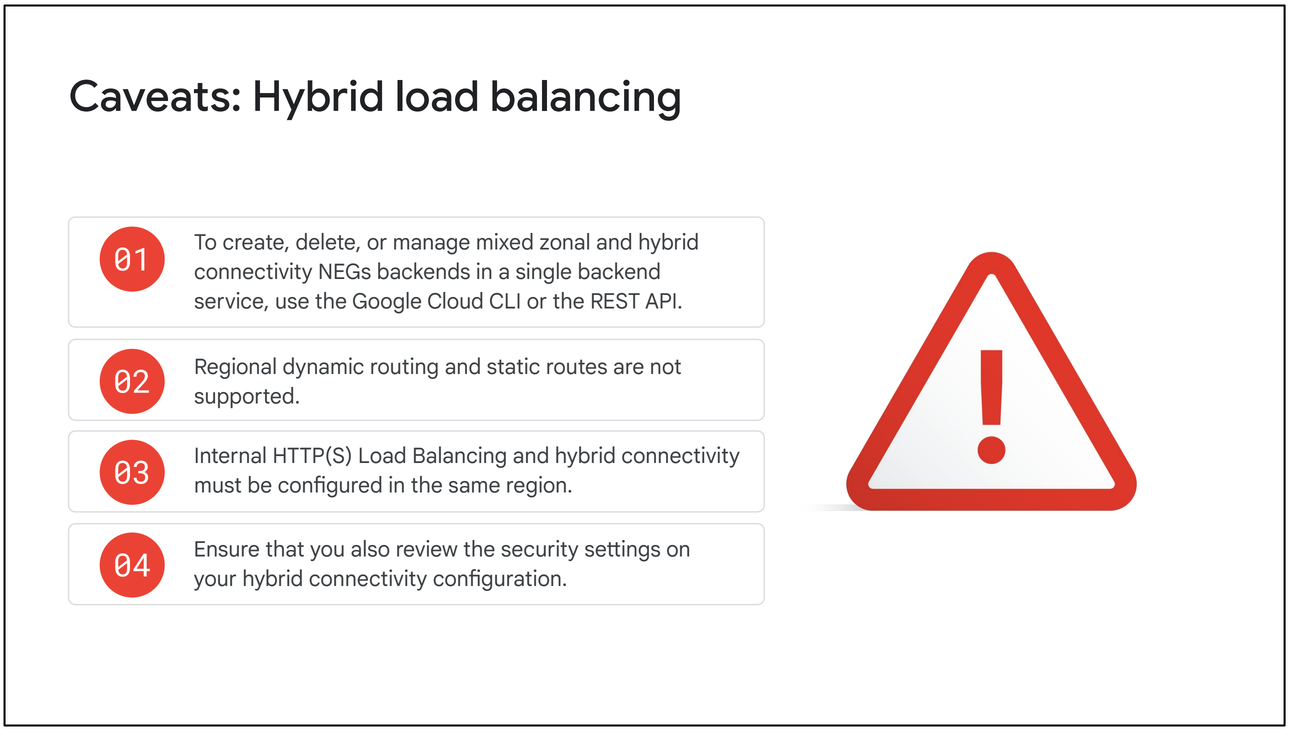
These Google Cloud load balancers support hybrid load balancing:

- Global external HTTP(S) load balancer
- Global external HTTP(S) load balancer (classic)
- Regional external HTTP(S) load balancer
- Internal HTTP(S) load balancer
- External TCP proxy load balancer
- External SSL proxy load balancer
- Internal regional TCP proxy load balancer
- External regional TCP proxy load balancer
- External TCP/UDP network load balancer
- Internal TCP/UDP load balancer

You can use hybrid load balancing with the following:

- Global external HTTP(S) load balancer
- Global external HTTP(S) load balancer (classic)
- Regional external HTTP(S) load balancer
- Internal HTTP(S) load balancer
- External TCP proxy load balancer
- External SSL proxy load balancer
- Internal regional TCP proxy load balancer
- External regional TCP proxy load balancer
- External TCP/UDP network load balancer
- Internal TCP/UDP load balancer

You can choose a load balancer depending on your needs, such as where the clients and workloads are located.



Caveats: Hybrid load balancing

01 To create, delete, or manage mixed zonal and hybrid connectivity NEGs backends in a single backend service, use the Google Cloud CLI or the REST API.

02 Regional dynamic routing and static routes are not supported.

03 Internal HTTP(S) Load Balancing and hybrid connectivity must be configured in the same region.

04 Ensure that you also review the security settings on your hybrid connectivity configuration.

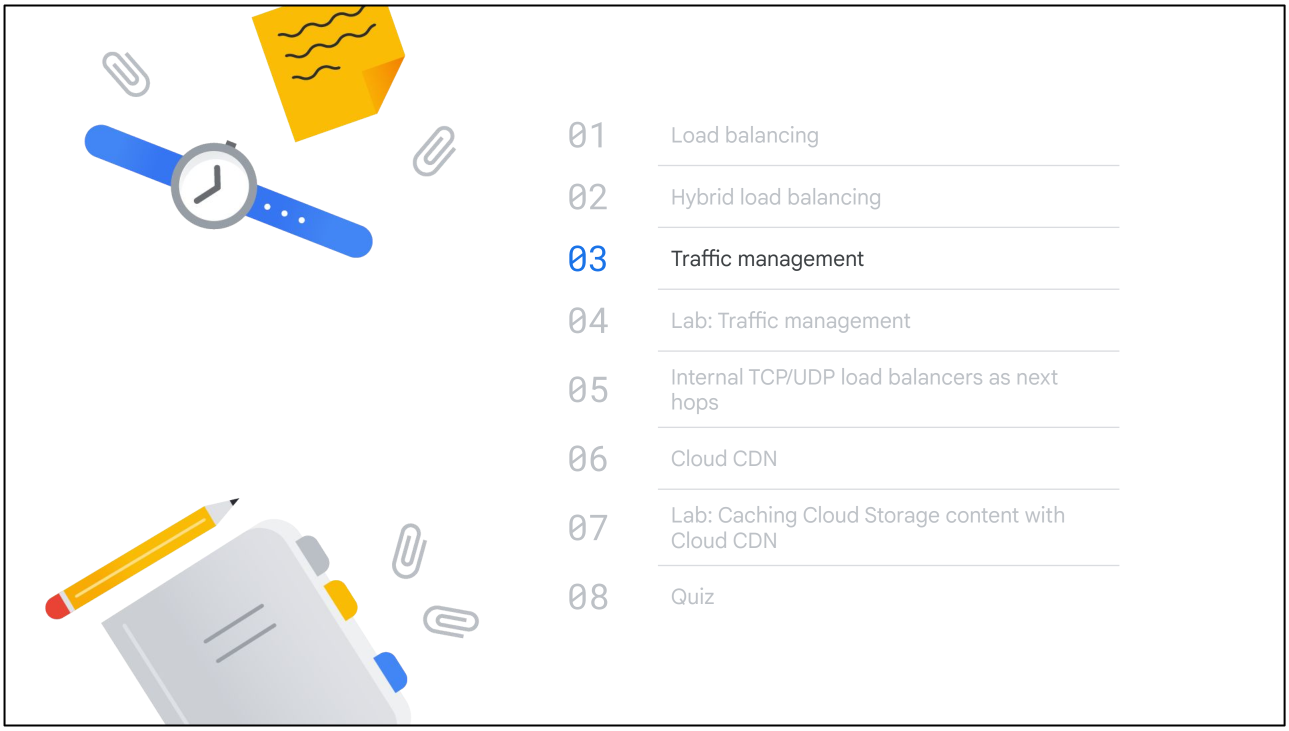
To create, delete, or manage mixed zonal and hybrid connectivity NEGs backends in a single backend service, you must use the Google Cloud CLI or the REST API.

Regional dynamic routing and static routes are not supported. The Cloud Router used for hybrid connectivity must be enabled with global dynamic routing.

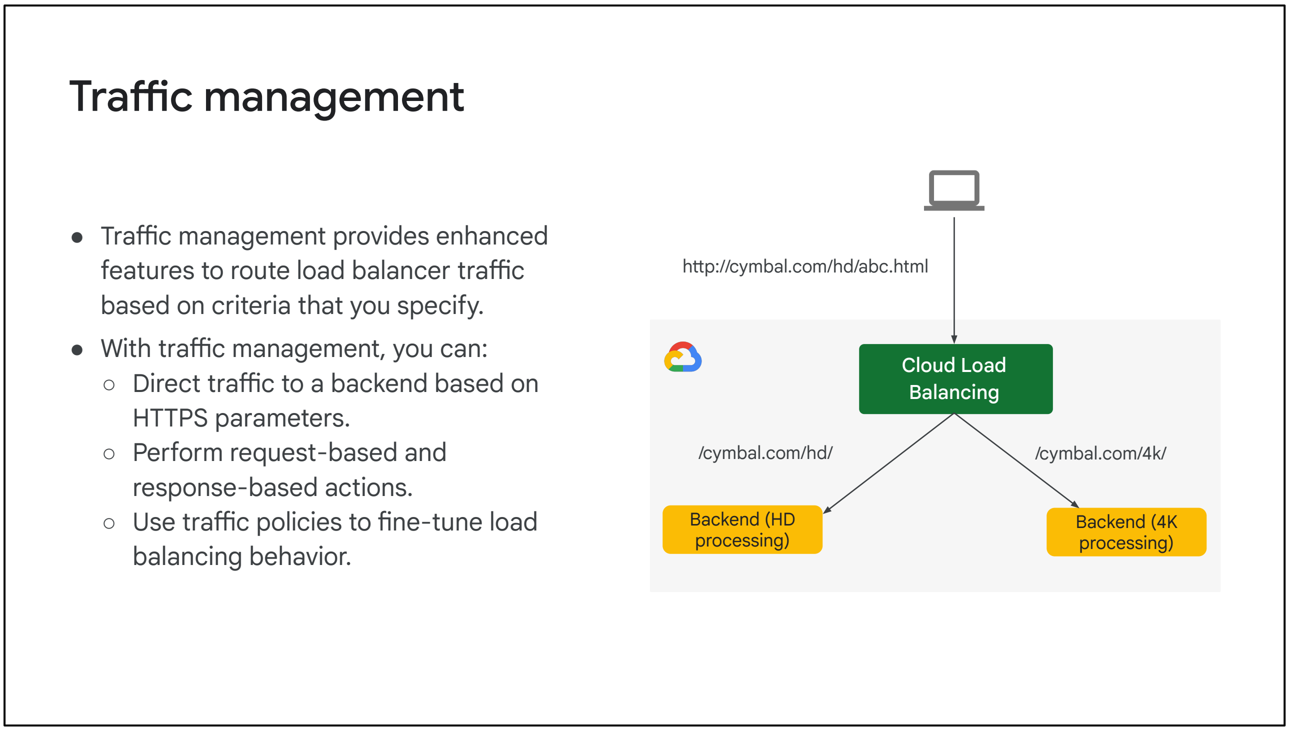
Internal HTTP(S) Load Balancing and hybrid connectivity must be configured in the same region. If they are configured in different regions, you might see backends as healthy, but client requests will not be forwarded to the backend.

Ensure that you also review the security settings on your hybrid connectivity configuration. Currently, HA Cloud VPN connections are encrypted by default, using IPsec encryption. Cloud Interconnect connections are not encrypted by default.

For more details, go to Encryption in Transit in Google Cloud on the Google Cloud website.

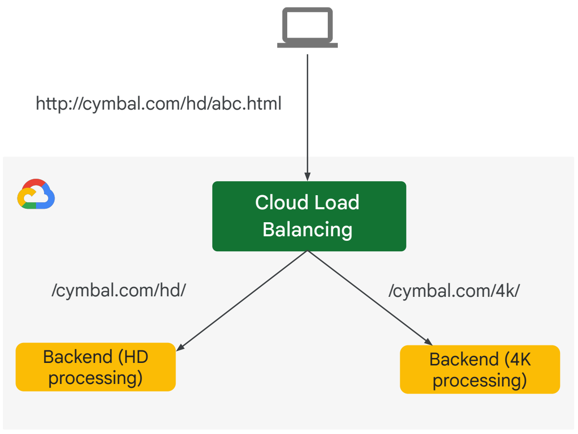


Next, let’s discuss traffic management, which provides enhanced features to route traffic based on criteria that you specify. You will be introduced to the URL map and how it is used to configure traffic management.



Traffic management

- Traffic management provides enhanced features to route load balancer traffic based on criteria that you specify.
- With traffic management, you can:
  - Direct traffic to a backend based on HTTPS parameters.
  - Perform request-based and response-based actions.
  - Use traffic policies to fine-tune load balancing behavior.

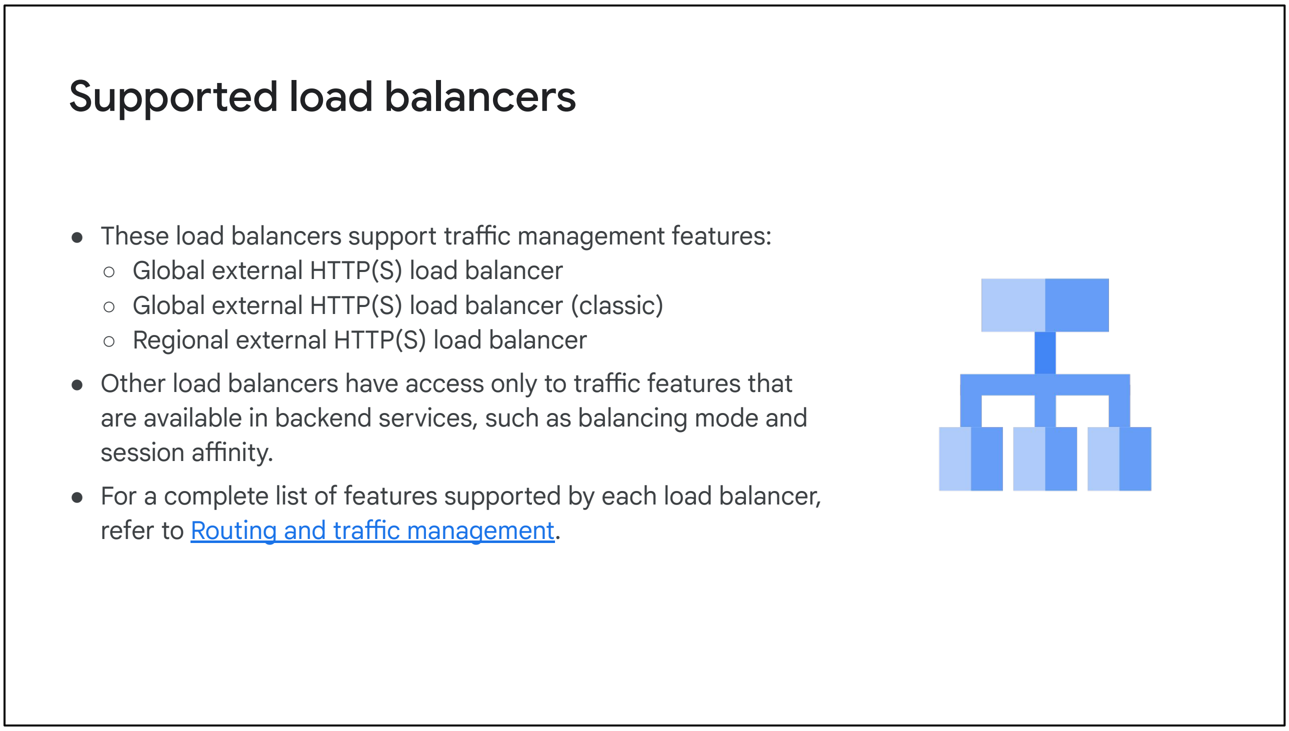


Traffic management provides enhanced features to route load balancer traffic based on criteria that you specify.

With traffic management, you can:
- Implement traffic steering based on HTTPS parameters, such as the host, path, headers, and other request parameters.
- Perform request-based and response-based actions, such as redirects and header transformations.
- Use traffic policies to fine-tune load balancing behavior, such as retry policies, request mirroring, and cross-origin resource sharing (CORS).

The traffic features that are available can vary per load balancer. Check the Google Cloud documentation for details, for example, Traffic management overview for a global external HTTP(S) load balancer (classic), Traffic management overview for global external HTTP(S) load balancers, and Traffic management overview for regional external HTTP(S) load balancers.

Recall that, in addition to traffic management, Cloud Load Balancing offers backend services like health checks, session affinity, balancing mode, and capacity scaling.



Supported load balancers

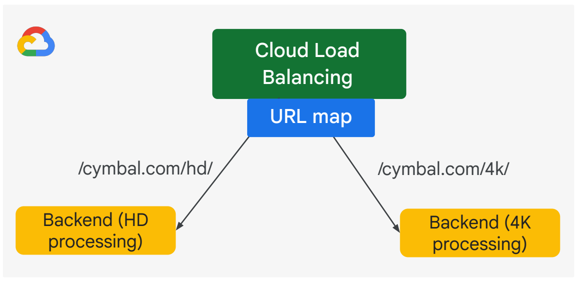
- These load balancers support traffic management features:
  - Global external HTTP(S) load balancer
  - Global external HTTP(S) load balancer (classic)
  - Regional external HTTP(S) load balancer
- Other load balancers have access only to traffic features that are available in backend services, such as balancing mode and session affinity.
- For a complete list of features supported by each load balancer, refer to Routing and traffic management.

These load balancers support traffic management: global external HTTP(S) load balancer, global external HTTP(S) load balancer (classic), and the regional external HTTP(S) load balancer. Other load balancers have access to only traffic features available in backend services, such as balancing mode and session affinity.

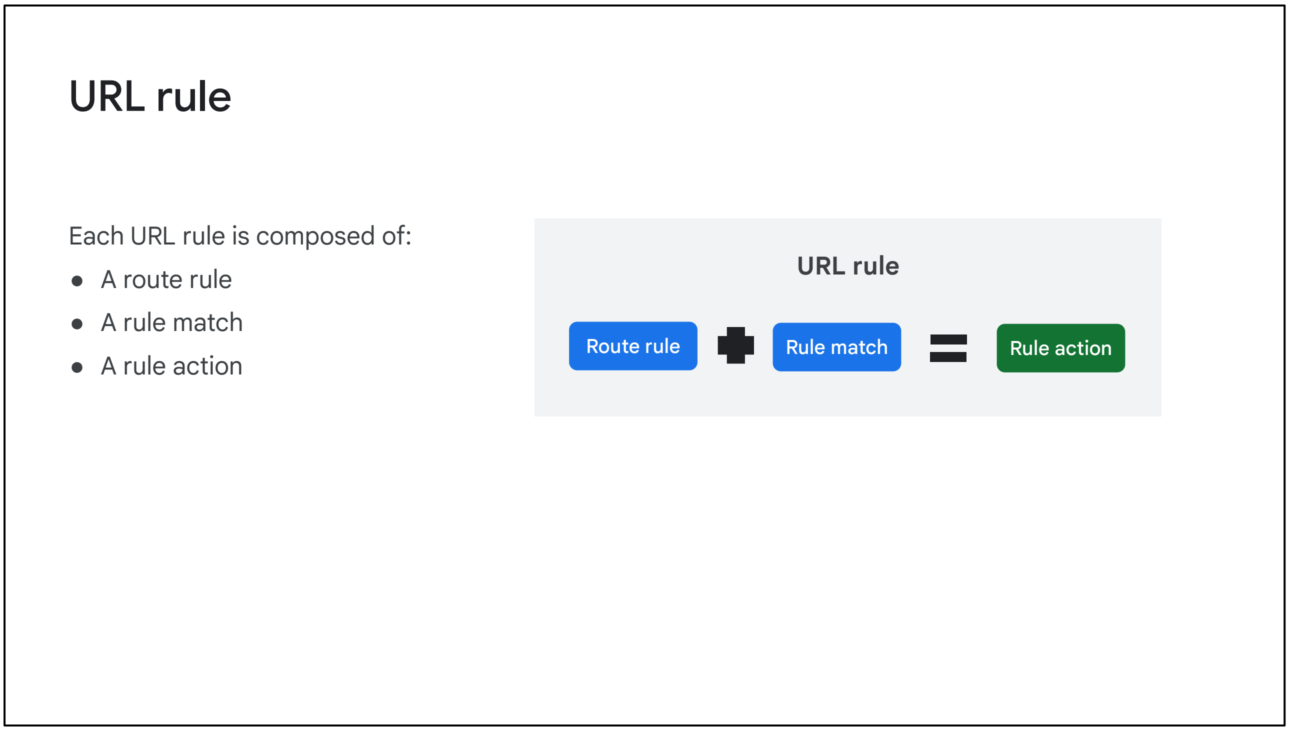
Not all load balancers support all traffic management features. For a complete list of traffic management features supported for each load balancer, refer to Routing and traffic management in the Google Cloud documentation.

URL map

- The URL map contains rules that define the criteria to use to route incoming traffic to a backend service.
- Traffic management features are configured in a URL map.
- The load balancer uses the URL map to determine where to route incoming traffic.



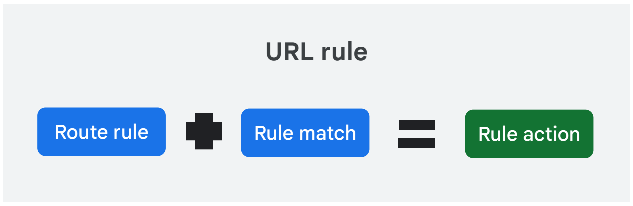
The URL map contains rules that define the criteria to use to route incoming traffic to a backend service. Traffic management features are configured in a URL map. In other words, the load balancer uses the URL map to determine where to route incoming traffic.



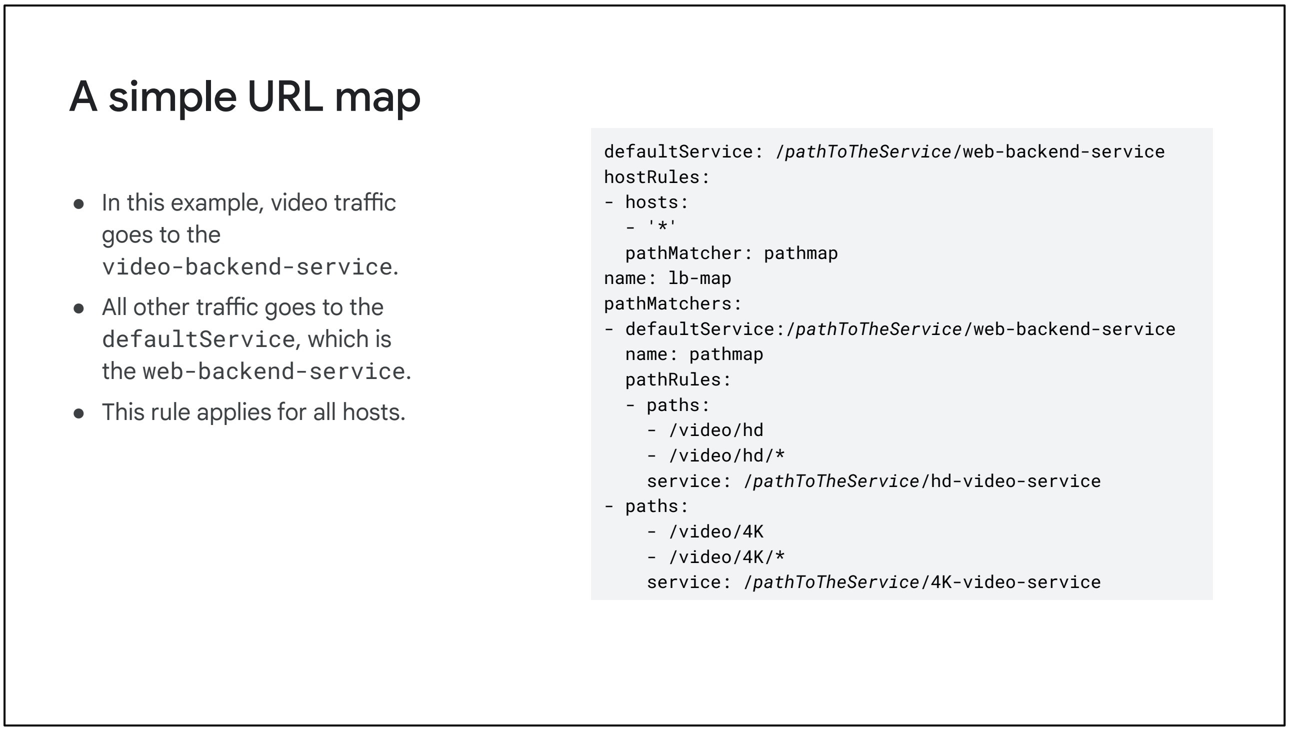
URL rule

Each URL rule is composed of:

- A route rule
- A rule match
- A rule action



Each URL rule in a URL map is composed of a route rule, a rule match, and a rule action. Let's look at an example, using simple routing.



A simple URL map

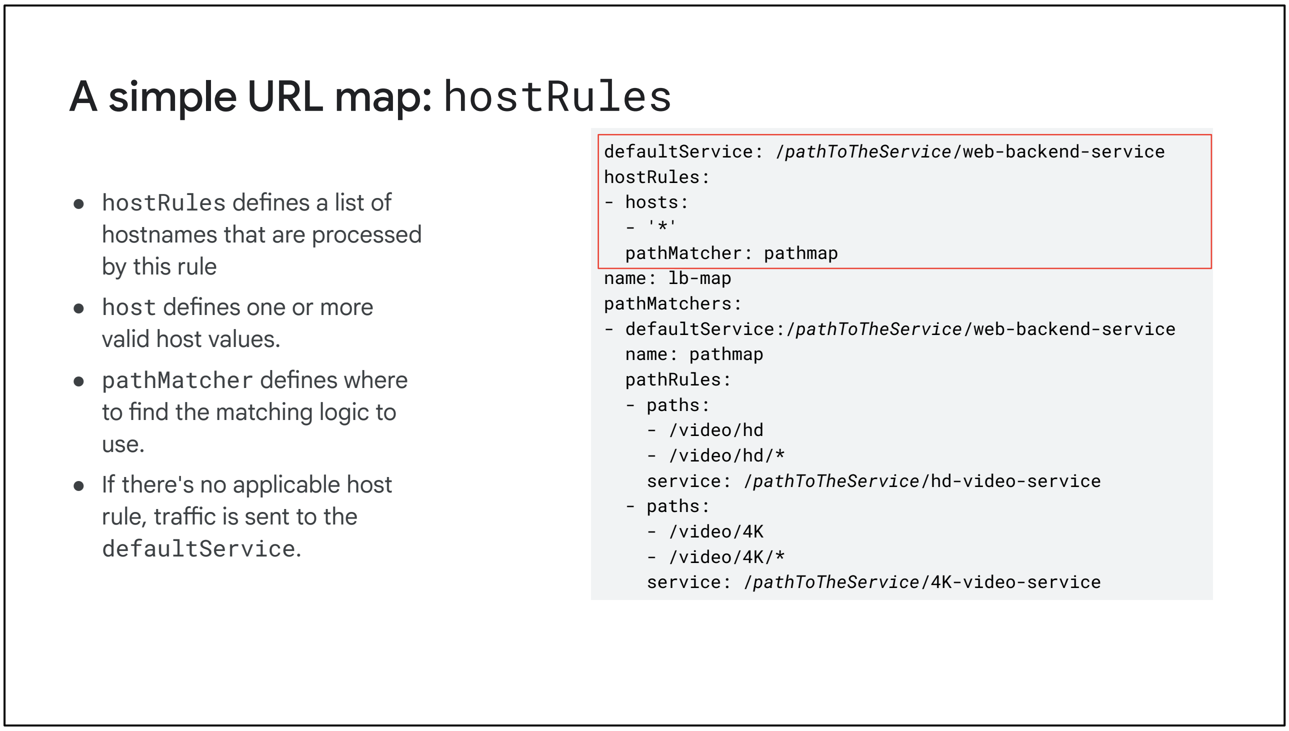
- In this example, video traffic goes to the `video-backend-service`.
- All other traffic goes to the `defaultService`, which is the `web-backend-service`.
- This rule applies for all hosts.

```
defaultService: /pathToTheService/web-backend-service
hostRules:
- hosts:
  - '*'
  pathMatcher: pathmap
name: lb-map
pathMatchers:
- defaultService:/pathToTheService/web-backend-service
  name: pathmap
  pathRules:
  - paths:
    - /video/hd
    - /video/hd/*
    service: /pathToTheService/hd-video-service
- paths:
    - /video/4K
    - /video/4K/*
    service: /pathToTheService/4K-video-service
```

This slide shows a simple routing example. In the YAML file, you see a URL map that routes video traffic to the video-backend-service. All other traffic is routed to the default service, which is the web-backend-service.

Instead of using a YAML file, you can also use the Google Cloud console to configure URL maps.

Let’s look at how this example works.



A simple URL map: `hostRules`

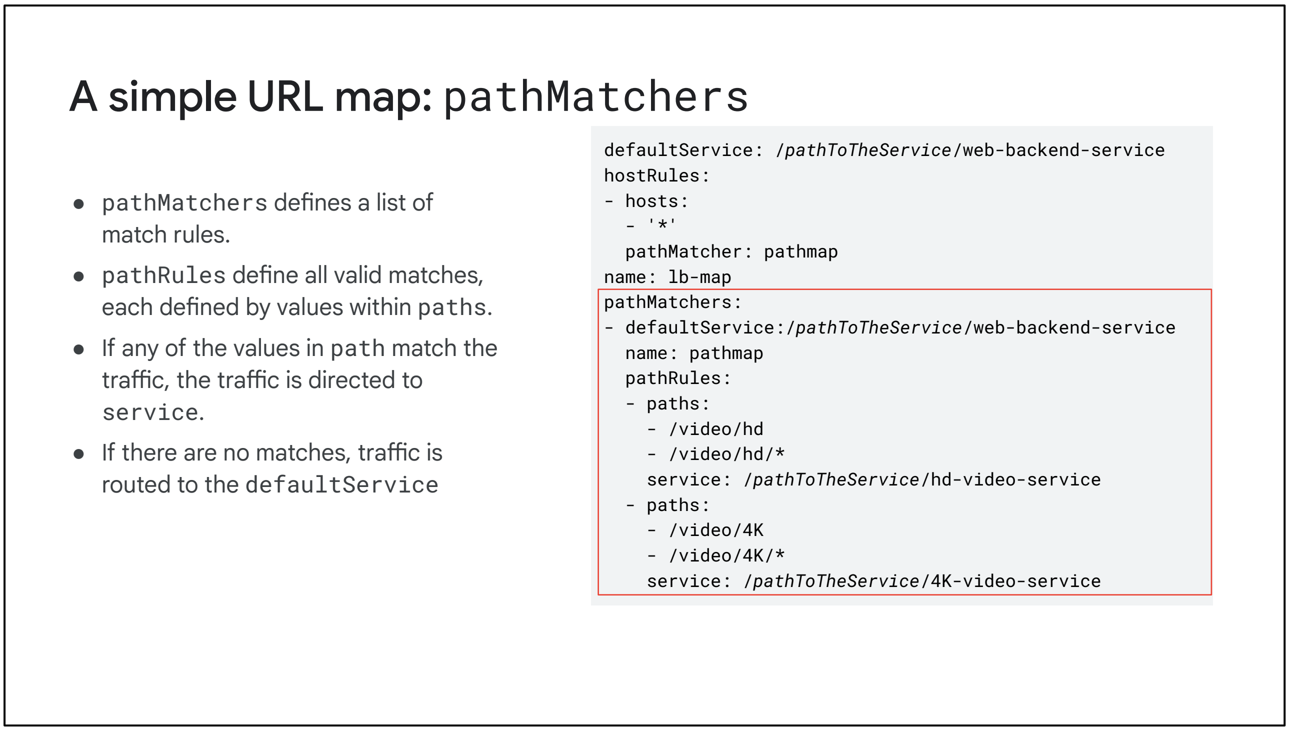
- `hostRules` defines a list of hostnames that are processed by this rule
- `host` defines one or more valid host values.
- `pathMatcher` defines where to find the matching logic to use.
- If there's no applicable host rule, traffic is sent to the `defaultService`.

```
defaultService: /pathToTheService/web-backend-service
hostRules:
- hosts:
  - '*'
  pathMatcher: pathmap
name: lb-map
pathMatchers:
- defaultService:/pathToTheService/web-backend-service
  name: pathmap
  pathRules:
  - paths:
    - /video/hd
    - /video/hd/*
    service: /pathToTheService/hd-video-service
  - paths:
    - /video/4K
    - /video/4K/*
    service: /pathToTheService/4K-video-service
```

The `defaultService` defines a service where traffic should be routed when no matching URL rule is found. You must specify a `defaultService` or a `backendBucket`.

The `hostRules` defines a list of hostnames that are processed by this rule. In this example, there’s only one item in the list, which means that only one host rule is defined. This host rule contains an asterisk (*). The asterisk is a wildcard, which matches all hosts.

To see where to find the matching logic to use, look at the value of `pathMatcher`. For this host rule, `pathMatcher` is set to `pathmap`.



A simple URL map: `pathMatchers`

- `pathMatchers` defines a list of match rules.
- `pathRules` define all valid matches, each defined by values within `paths`.
- If any of the values in `path` match the traffic, the traffic is directed to `service`.
- If there are no matches, traffic is routed to the `defaultService`

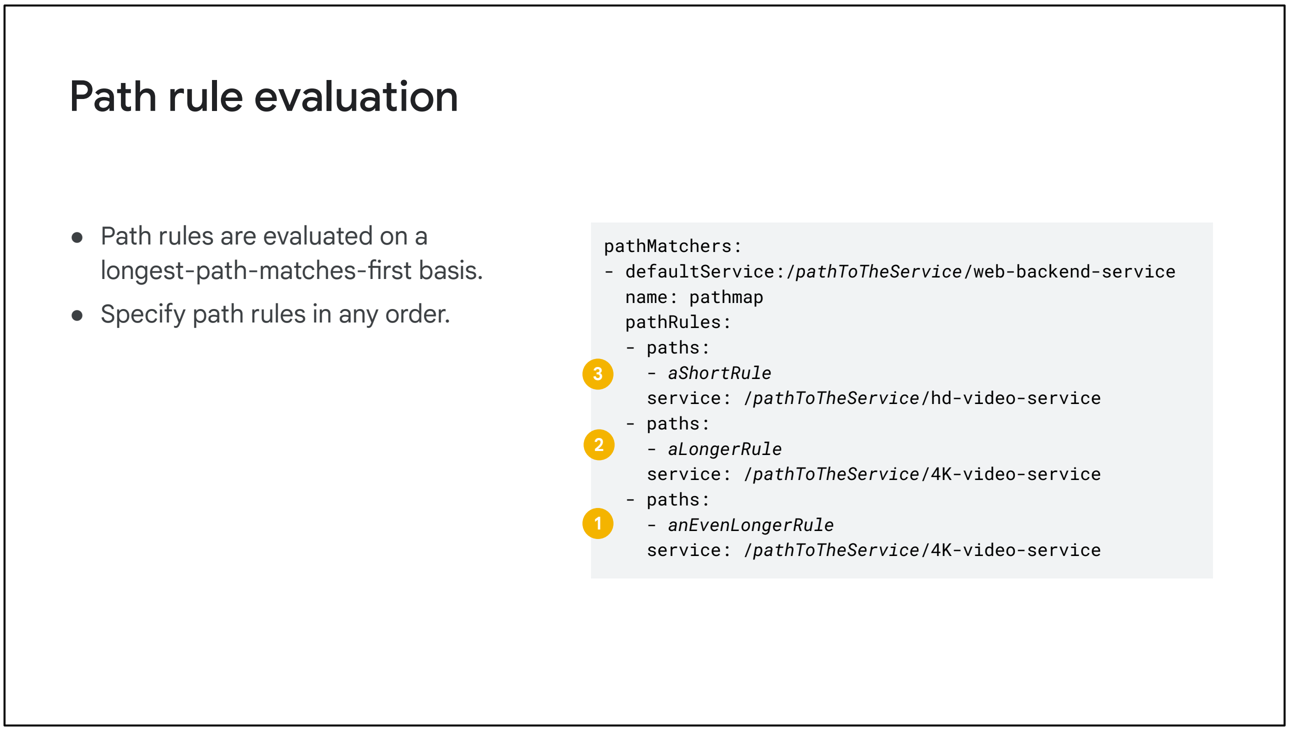
```
defaultService: /pathToTheService/web-backend-service
hostRules:
- hosts:
  - '*'
  pathMatcher: pathmap
name: lb-map
pathMatchers:
- defaultService:/pathToTheService/web-backend-service
  name: pathmap
  pathRules:
  - paths:
    - /video/hd
    - /video/hd/*
    service: /pathToTheService/hd-video-service
  - paths:
    - /video/4K
    - /video/4K/*
    service: /pathToTheService/4K-video-service
```

Now, let's explore this example in detail. There's a `pathMatchers` list, which contains a list of path matching rules. The only element in this list is `pathmap`, which in this example is also specified in the `hostRules`.

Each match rule defines logic to process the traffic that is sent to the load balancer.

In this example, there are two sets of `paths`. One `paths` list defines valid URL paths for the `hd-video-service`. The other `paths` list defines valid URL paths for the `4K-video-service`. If the URL contains a match for one of these paths lists, the load balancer routes the traffic to the corresponding service.

If the traffic contains a path that matches none of the `paths` lists, then it's sent to the default backend service, `web-backend-service`. In other words, the traffic is sent to the service denoted by `pathMatchers/defaultService`.



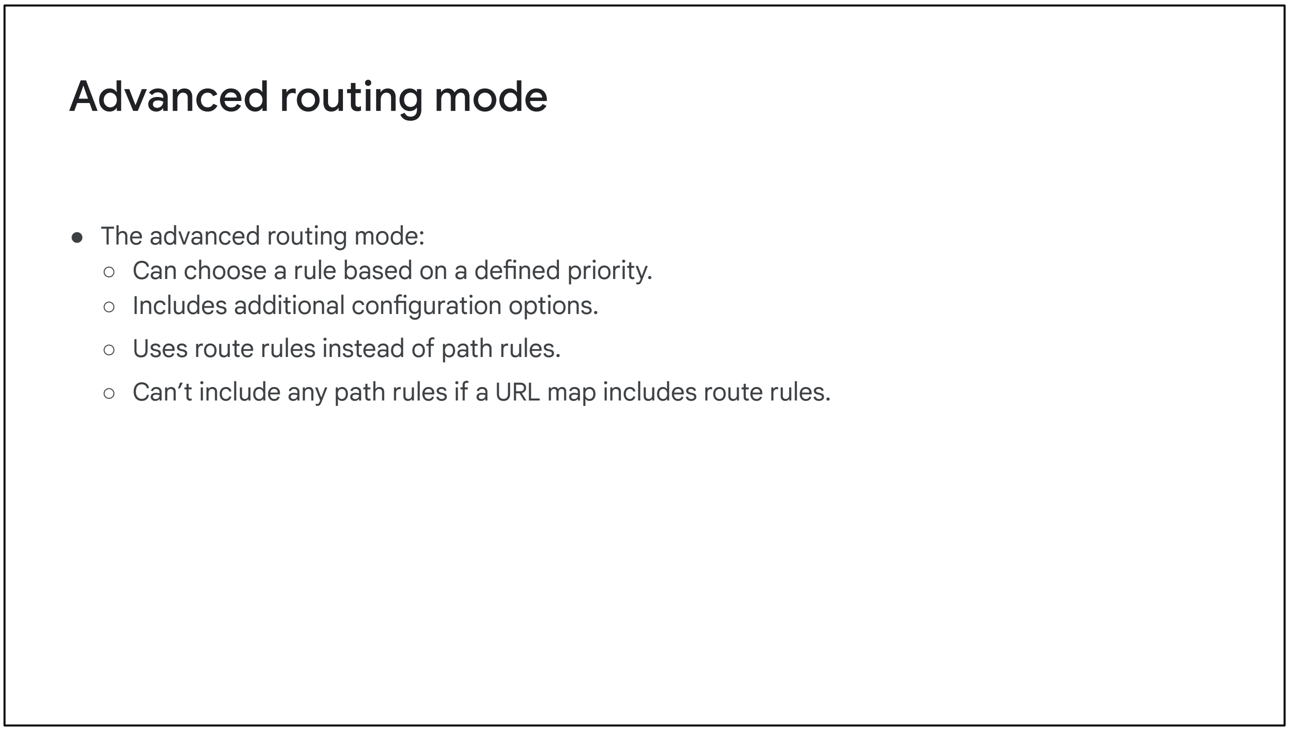
Path rule evaluation

- Path rules are evaluated on a longest-path-matches-first basis.
- Specify path rules in any order.

```
  pathMatchers:
  - defaultService:/pathToTheService/web-backend-service
    name: pathmap
    pathRules:
    - paths:
3     - aShortRule
      service: /pathToTheService/hd-video-service
    - paths:
2     - aLongerRule
      service: /pathToTheService/4K-video-service
    - paths:
1     - anEvenLongerRule
      service: /pathToTheService/4K-video-service
```

Path rules are evaluated on a longest-path-matches-first basis. You can specify the path rules in any order.

In the example, each path rule is labeled to show the order of evaluation. The rule labeled with 1 is evaluated first. The rule labeled with 3 is evaluated last.

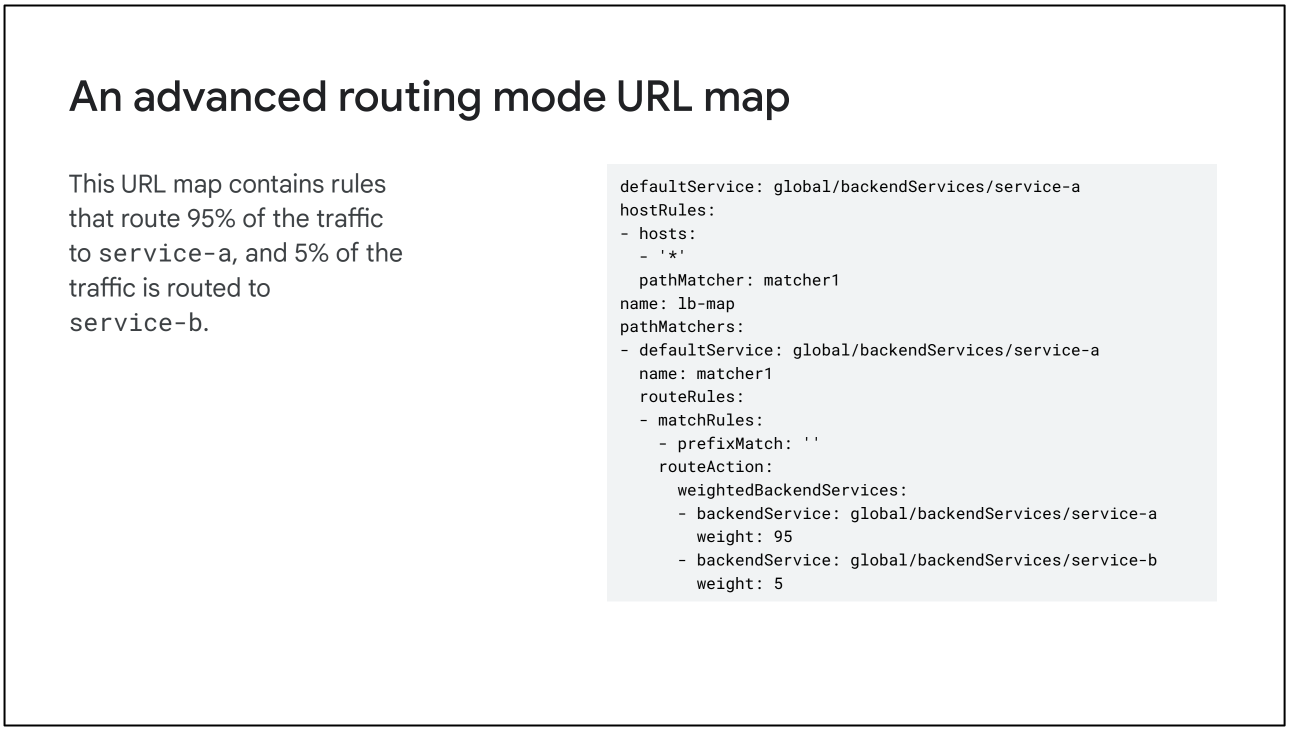


Advanced routing mode

- The advanced routing mode:
  - Can choose a rule based on a defined priority.
  - Includes additional configuration options.
  - Uses route rules instead of path rules.
  - Can’t include any path rules if a URL map includes route rules.

The advanced routing mode can choose a rule based on a defined priority and includes additional configuration options. Instead of path rules, advanced routing uses route rules.

Each URL map can include either simple or advanced rules, but not both.



An advanced routing mode URL map

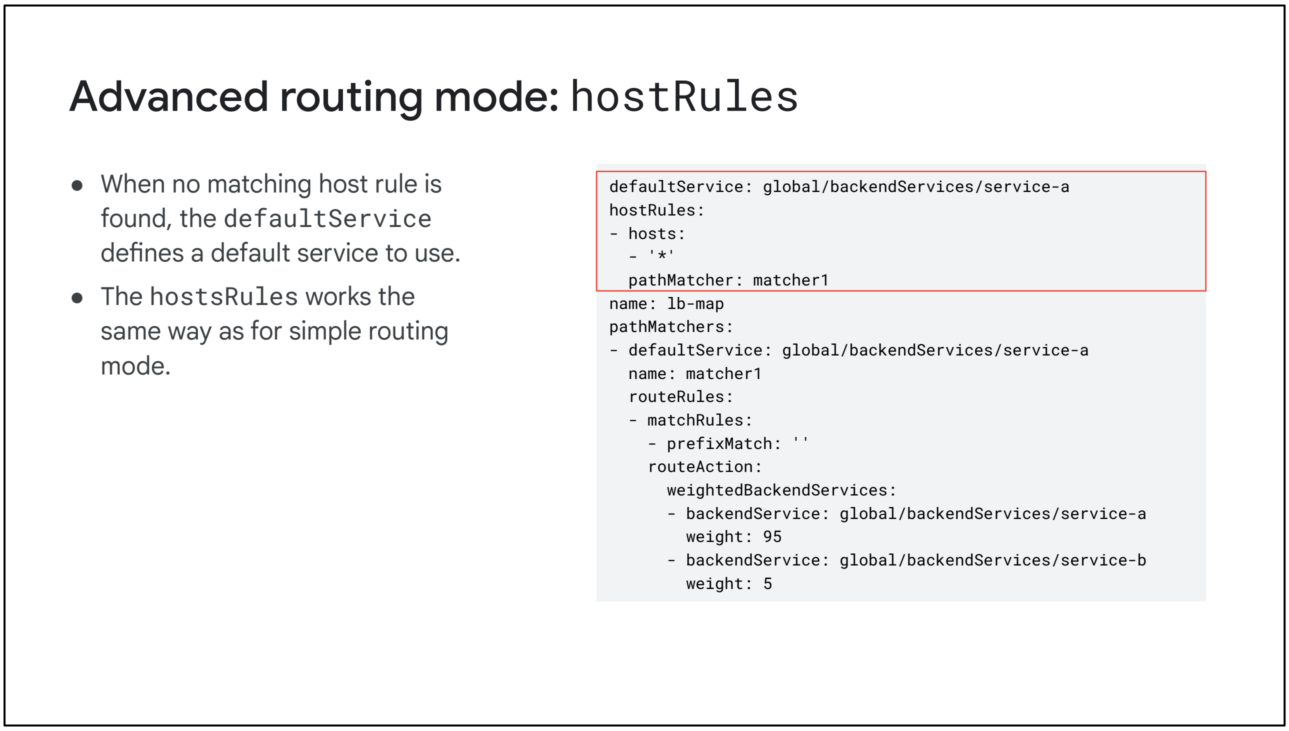
This URL map contains rules that route 95% of the traffic to `service-a`, and 5% of the traffic is routed to `service-b`.

```
defaultService: global/backendServices/service-a
hostRules:
- hosts:
  - '*'
  pathMatcher: matcher1
name: lb-map
pathMatchers:
- defaultService: global/backendServices/service-a
  name: matcher1
  routeRules:
  - matchRules:
    - prefixMatch: ''
    routeAction:
      weightedBackendServices:
      - backendService: global/backendServices/service-a
        weight: 95
      - backendService: global/backendServices/service-b
        weight: 5
```

This URL map contains rules that route 95% of the traffic to `service-a`, and 5% of the traffic is routed to `service-b`.

The example shows a YAML implementation of an advanced routing mode. You can also use the Google Cloud console to configure URL maps.

Let’s look at how this example works.



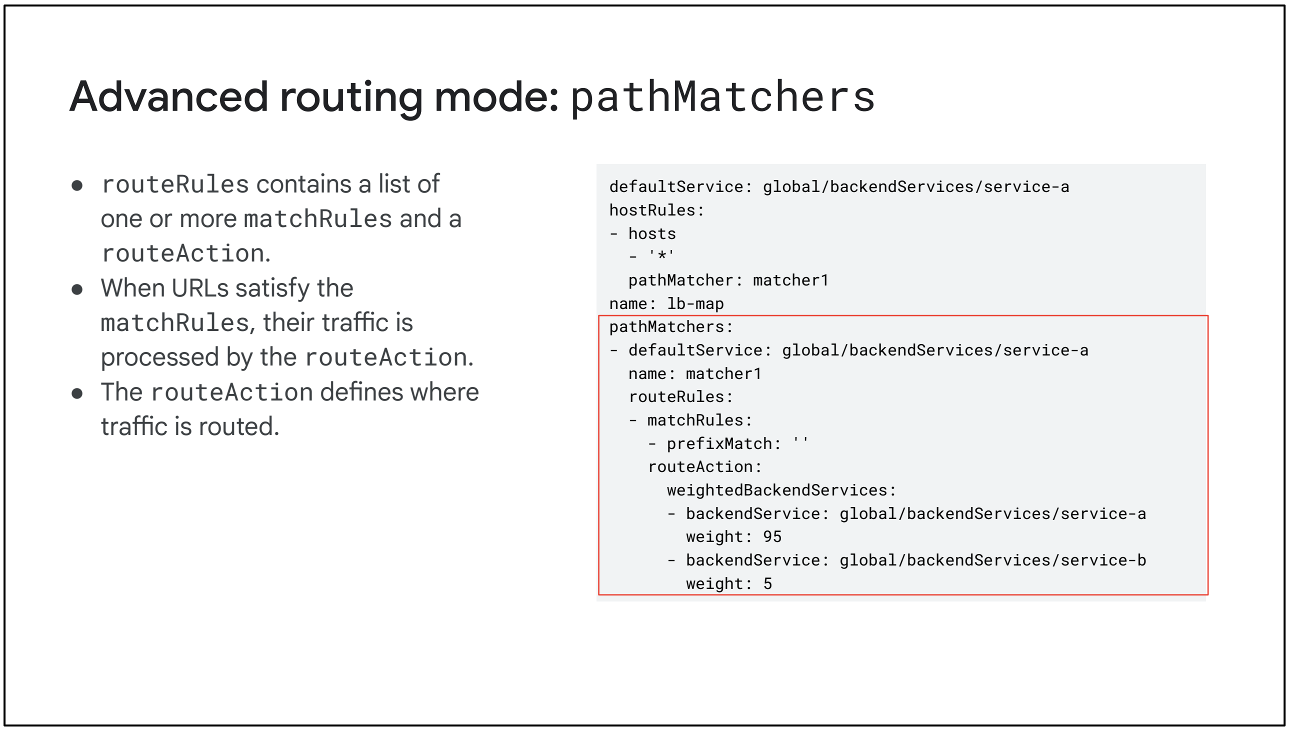
Advanced routing mode: `hostRules`

- When no matching host rule is found, the `defaultService` defines a default service to use.
- The `hostsRules` works the same way as for simple routing mode.

```
defaultService: global/backendServices/service-a
hostRules:
- hosts:
  - '*'
  pathMatcher: matcher1
name: lb-map
pathMatchers:
- defaultService: global/backendServices/service-a
  name: matcher1
  routeRules:
  - matchRules:
    - prefixMatch: ''
    routeAction:
      weightedBackendServices:
      - backendService: global/backendServices/service-a
        weight: 95
      - backendService: global/backendServices/service-b
        weight: 5
```

When no matching host rule is found, the `defaultService` defines a service to use. `defaultService` is a required field.

The `hostsRules` works the same way as for simple routing mode. As in the previous example, this host rule uses the asterisk to match all hosts. Because `pathMatcher` is set to `matcher1`, `/pathMatchers/matcher1` defines the matching logic.



Advanced routing mode: `pathMatchers`

- `routeRules` contains a list of one or more `matchRules` and a `routeAction`.
- When URLs satisfy the `matchRules`, their traffic is processed by the `routeAction`.
- The `routeAction` defines where traffic is routed.

```
defaultService: global/backendServices/service-a
hostRules:
- hosts
  - '*'
  pathMatcher: matcher1
name: lb-map
pathMatchers:
- defaultService: global/backendServices/service-a
  name: matcher1
  routeRules:
  - matchRules:
    - prefixMatch: ''
    routeAction:
      weightedBackendServices:
      - backendService: global/backendServices/service-a
        weight: 95
      - backendService: global/backendServices/service-b
        weight: 5
```

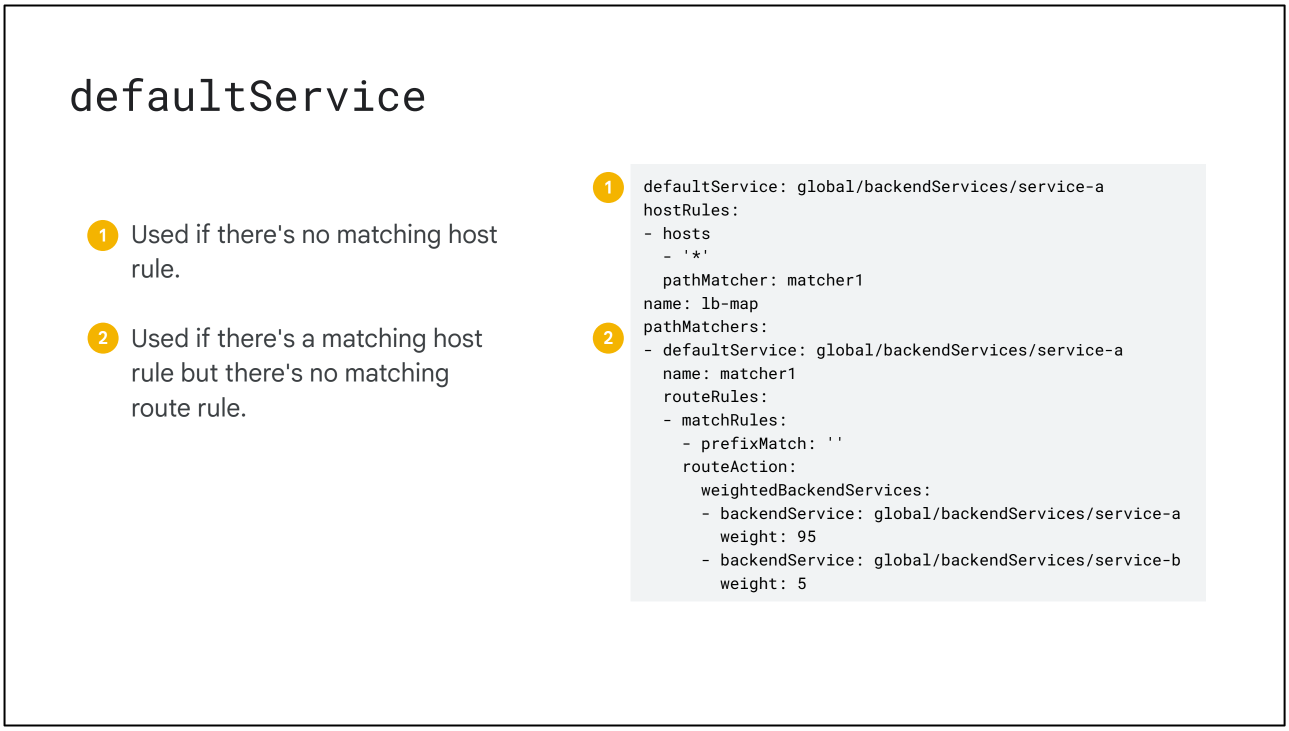
`/pathMatchers/matcher1` contains a list of `routeRules`. The `routeRules` contain a list of one or more `matchRules` and a `routeAction`. When URLs satisfy the `matchRules`, their traffic is processed by the `routeAction`.

In this example, there's only one item in `matchRules`, where `prefixMatch` equals an empty string. The `prefixMatch` condition matches the URL path prefix; URLs that start with the same string match. In th example, the `prefixMatch` is the empty string, which matches all URLs. In other words, all URLs trigger this match rule, and the `routeAction` is applied.

The `routeAction` defines how the traffic is routed. In the example, the `routeAction` is set to `weightedBackendServices`. `weightedBackedServices` is a list of backend services. A weight value is specified for each backend service; representing a percentage of the total traffic. 95% of the traffic is sent to `service-a`, and 5% of the traffic is sent to `service-b`.

The `routeAction` can also define traffic policies, such as retry policies and CORS.

For a complete list of `routeAction` values, refer to the Google Cloud documentation for the load balancer that you're using.



defaultService

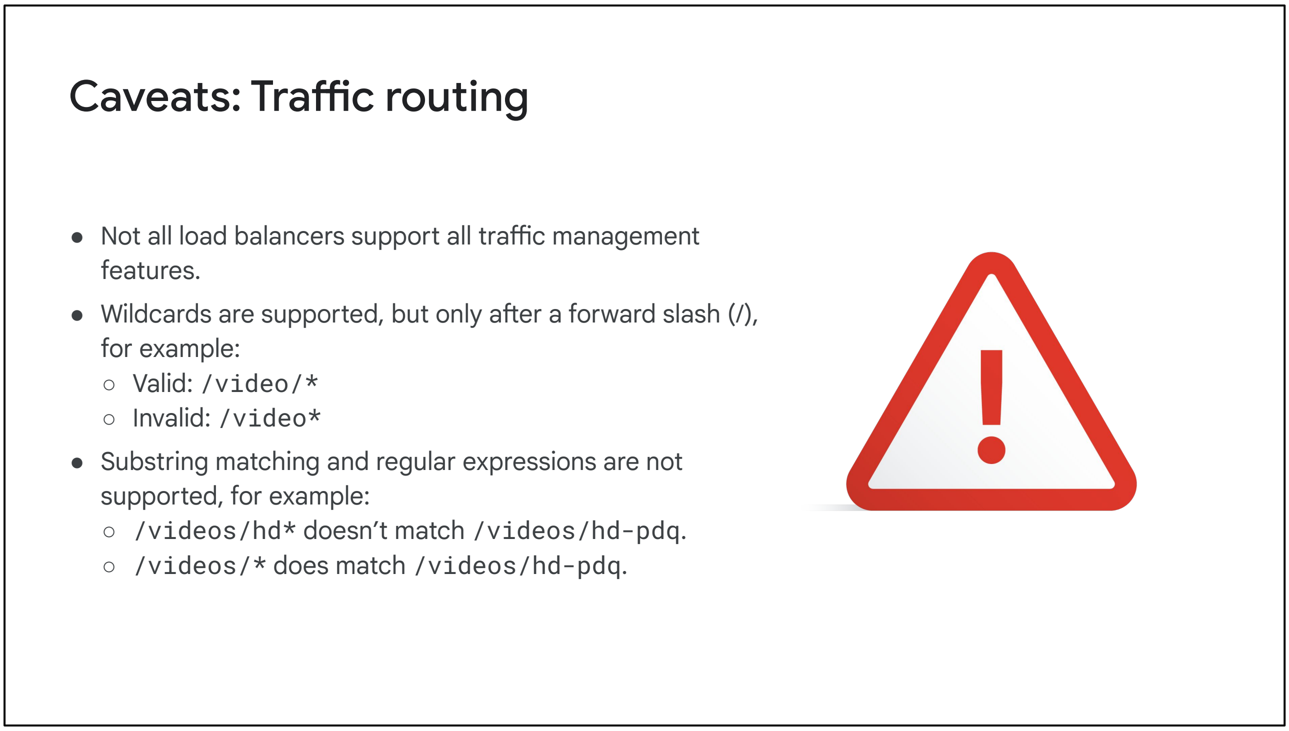
1. Used if there's no matching host rule.
2. Used if there's a matching host rule but there's no matching route rule.

```
1 defaultService: global/backendServices/service-a
  hostRules:
  - hosts
    - '*'
    pathMatcher: matcher1
  name: lb-map
  pathMatchers:
2 - defaultService: global/backendServices/service-a
    name: matcher1
    routeRules:
    - matchRules:
      - prefixMatch: ''
      routeAction:
        weightedBackendServices:
        - backendService: global/backendServices/service-a
          weight: 95
        - backendService: global/backendServices/service-b
          weight: 5
```

In this example, you might notice that there are two `defaultService` key-value pairs. One `defaultService` is associated with the `hostRules`, and the other is associated with the `routeRules`.

If there's no matching host rule, the first `defaultService` is used.

If there’s no matching route rule, the second `defaultService` is used.



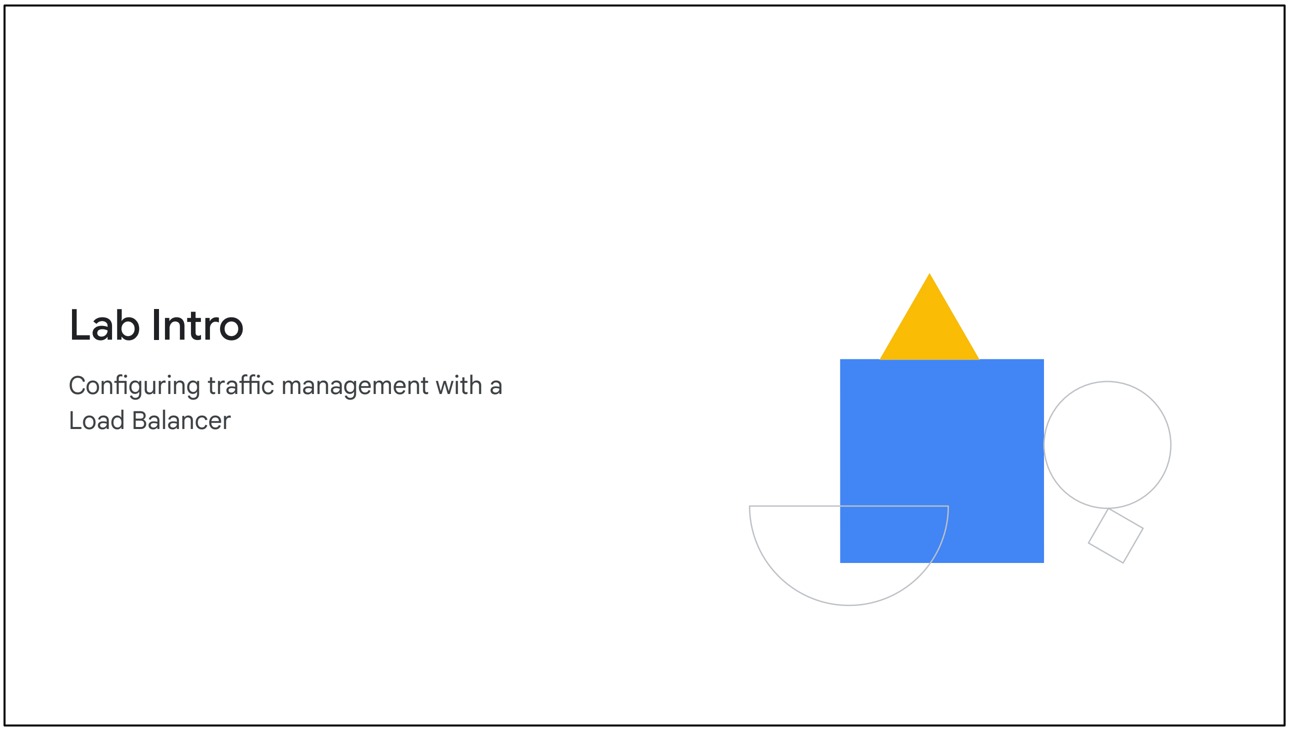
Caveats: Traffic routing

- Not all load balancers support all traffic management features.
- Wildcards are supported, but only after a forward slash (/), for example:
    - Valid: `/video/*`
    - Invalid: `/video*`
- Substring matching and regular expressions are not supported, for example:
    - `/videos/hd*` doesn’t match `/videos/hd-pdq`.
    - `/videos/*` does match `/videos/hd-pdq`.

Not all load balancers support all traffic management features. For a complete list of traffic management features supported for each load balancer, refer to Routing and traffic management in the Google Cloud documentation.

Wildcards are supported, but only after a forward slash (/). For example, /video/* is valid, and /video* is invalid.

Rule matching does not use regular expressions or substring matching. For example, `/videos/hd/*` does not match `/videos/hd-pdq`, because -pdq is a substring and also because it comes after the forward slash. `/videos/*` matches `/videos/hd-pqd`.

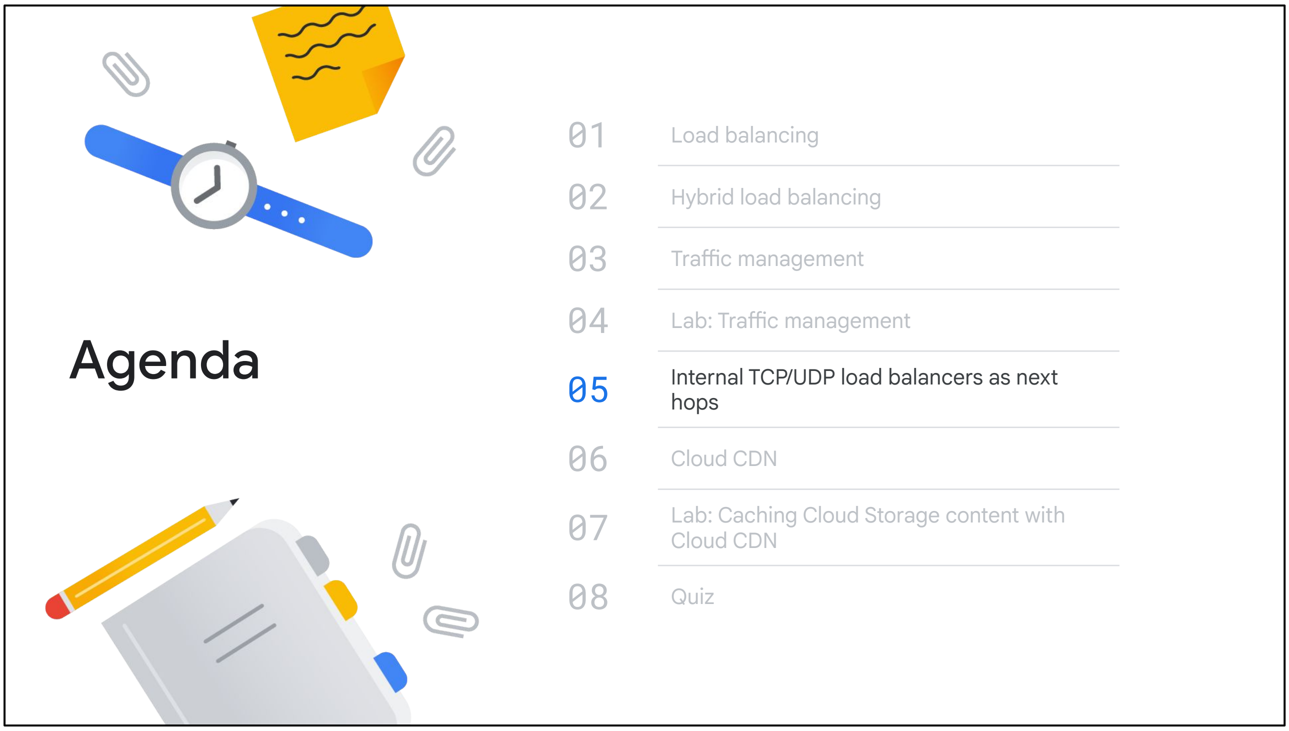


Lab Intro

Configuring traffic management with a Load Balancer

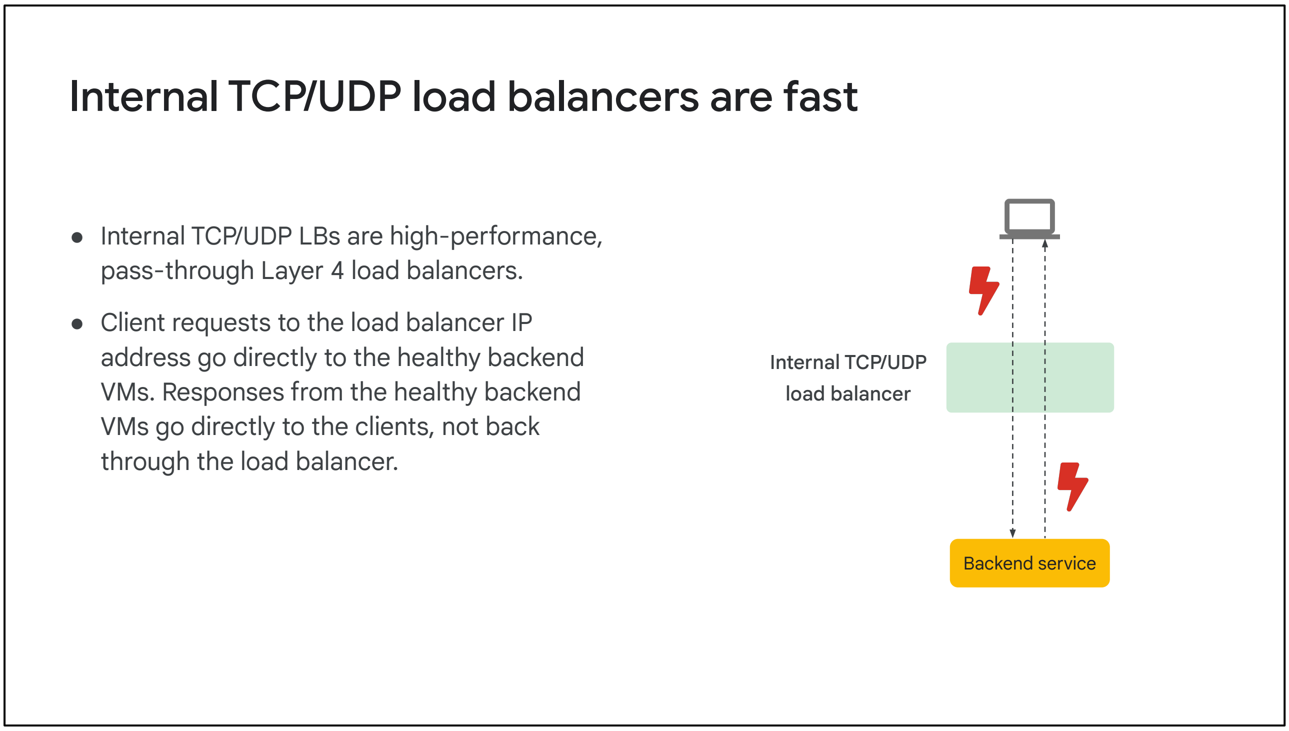
In this lab, you create a regional internal load balancer, with two backends. Each backend will be an instance group. You will use configure the load balancer create a blue/green deployment.

The blue deployment refers to the current version of your application. The green deployment refers to a new application version. In this lab exercise, you will configure the load balancer to send 70% of the traffic to the blue deployment and 30% to the green deployment.



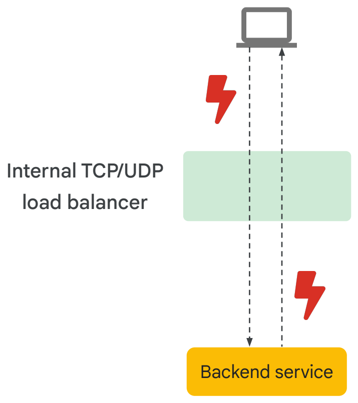
Agenda

Next, we’ll discuss using Internal TCP/UDP load balancers as next hops, including benefits, caveats, and some use cases.



Internal TCP/UDP load balancers are fast

- Internal TCP/UDP LBs are high-performance, pass-through Layer 4 load balancers.
- Client requests to the load balancer IP address go directly to the healthy backend VMs. Responses from the healthy backend VMs go directly to the clients, not back through the load balancer.

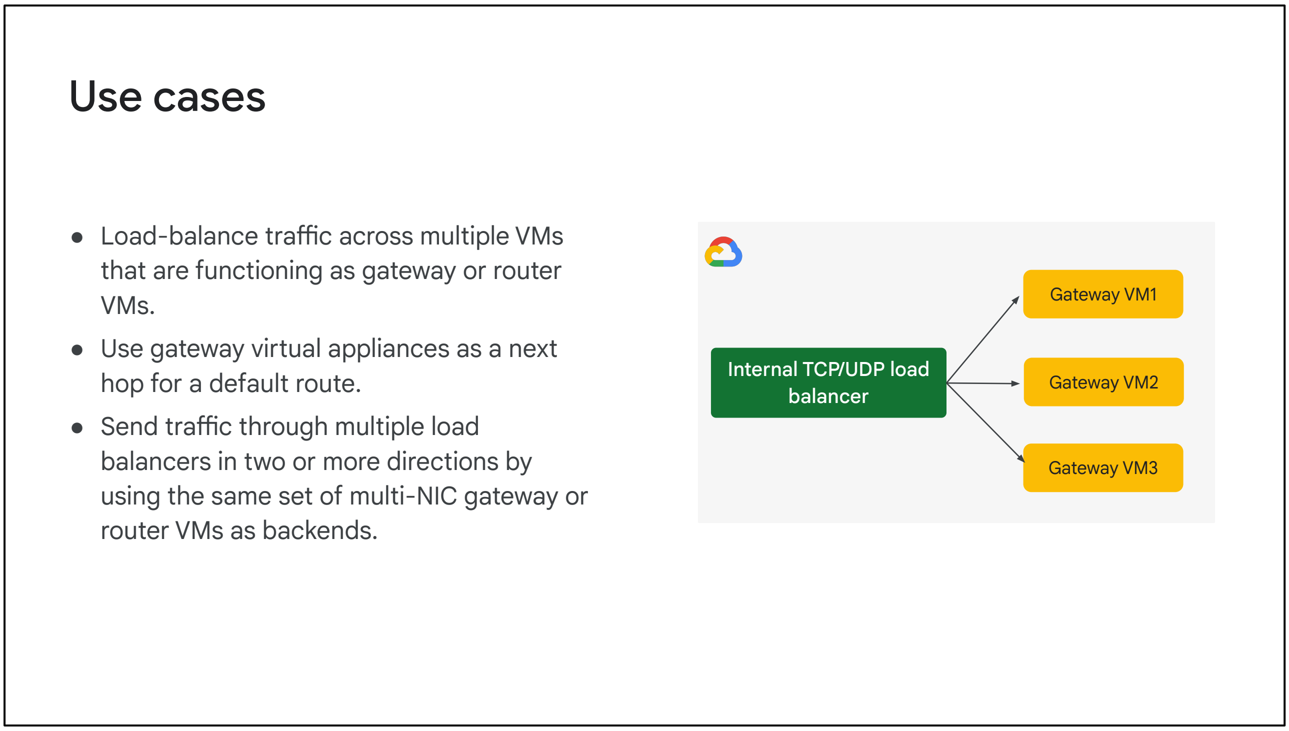


Before we cover how to use an Internal TCP/UDP load balancer as a routing next hop, let's discuss why these load balancers are useful: they're fast.

Internal TCP/UDP load balancers don't have the overhead associated with other types of Cloud load balancers; the reduced overhead makes them fast.

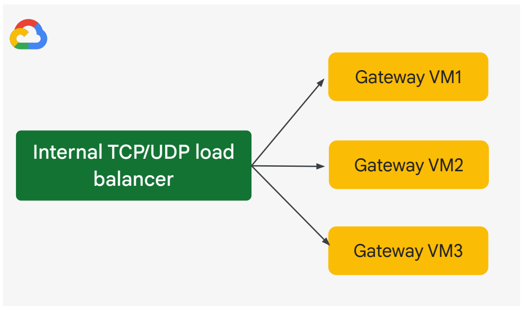
An internal TCP/UDP load balancer routes connections directly from clients to the healthy backend without any interruption. There's no intermediate device or single point of failure. Client requests to the load balancer IP address go directly to the healthy backend VMs. Unlike other types of load balancers, there's minimal processing of the incoming traffic.

Responses from the healthy backend VMs go directly to the clients, not back through the load balancer. TCP responses use direct server return. For more information, see [IP addresses for request and return packets] in the Google Cloud documentation.



Use cases

- Load-balance traffic across multiple VMs that are functioning as gateway or router VMs.
- Use gateway virtual appliances as a next hop for a default route.
- Send traffic through multiple load balancers in two or more directions by using the same set of multi-NIC gateway or router VMs as backends.



Let’s consider some use cases for internal TCP/UDP load balancers.

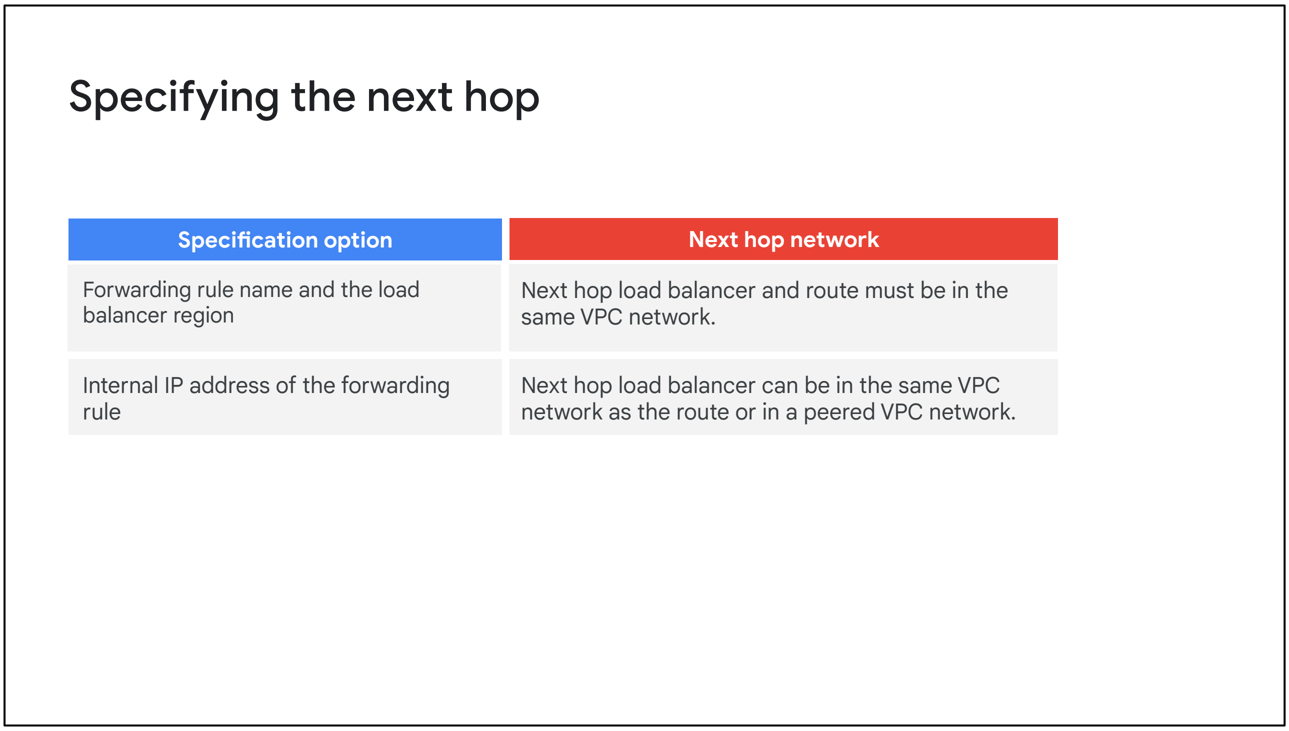
You can load-balance traffic across multiple VMs that are functioning as gateway or router VMs.

You can use gateway virtual appliances as the next hop for a default route. With this configuration, VM instances in your virtual private cloud (VPC) network send traffic to the internet through a set of load balanced virtual gateway VMs.

You can send traffic through multiple load balancers in two or more directions by using the same set of multi-NIC gateway or router VMs as backends. To accomplish this result, you create a load balancer and use it as the next hop for a custom static route in each VPC network. Each internal TCP/UDP load balancer operates within a single VPC network; distributing traffic to the network interfaces of backend VMs in that network.

In these use cases, the backend services are the gateway VMs, gateway virtual appliances, multi-NIC gateways, and router VMs. Because these resources are all internal, it makes sense to access them through an internal TCP/UDP load balancer. As we discussed a moment ago, these load balancers have lower overhead than other load balancers that Google Cloud offers.

Next, let’s consider how to make access to these backends even faster.



Specifying the next hop

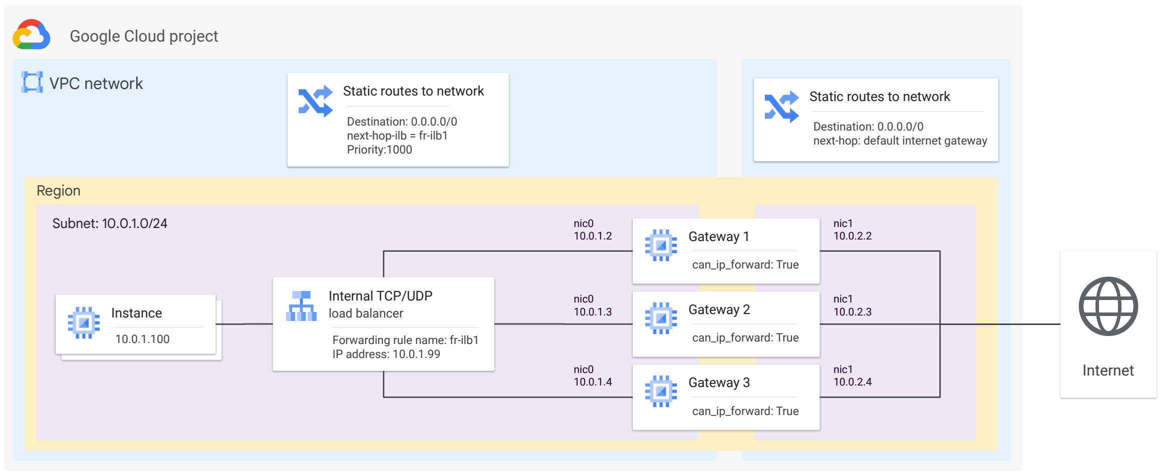
Specification option	Next hop network
Forwarding rule name and the load balancer region	Next hop load balancer and route must be in the same VPC network.
Internal IP address of the forwarding rule	Next hop load balancer can be in the same VPC network as the route or in a peered VPC network.

To specify the next hop, you have two choices, as shown in the table. The main difference concerns the location of the next hop load balancer.

If the next hop load balancer is in the same VPC network, you can specify the forwarding rule name and the load balancer region. To use a next hop load balancer in a peered VPC network, specify the internal IP address of the forwarding rule.

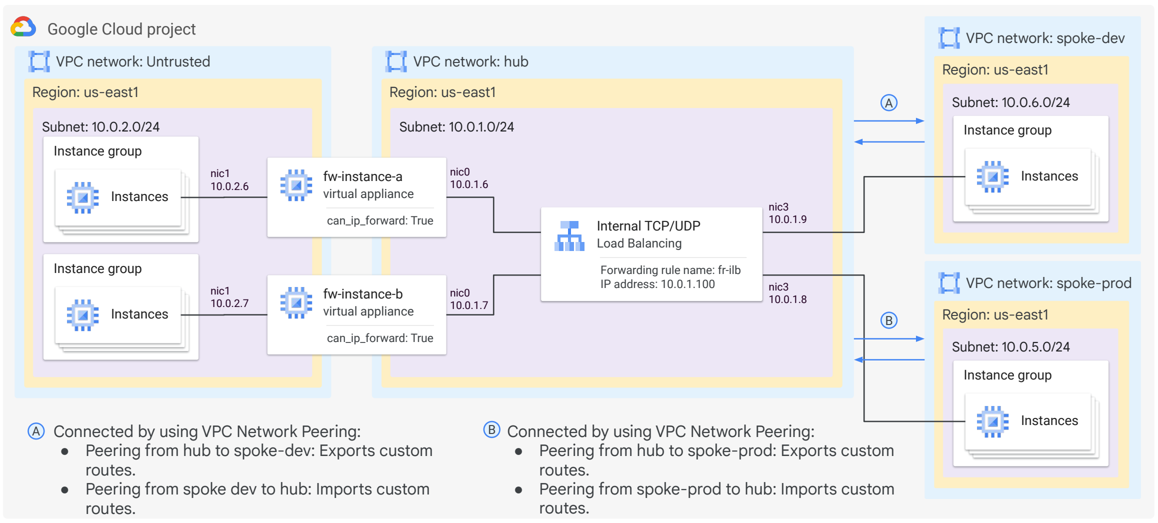
Next, let’s look at some sample topologies.

Next hop to a NAT gateway



This use case load balances traffic from internal VMs to multiple network address translation (NAT) gateway instances that route traffic to the internet. In this example, an internal TCP/UDP load balancer has next hops configured to three Compute Engine VMs. Each Compute Engine VM has a NAT gateway that runs on it, and has can_ip_forward set to true. These VMs then forward traffic to the internet. Optionally, you can set up the gateways to apply custom logic to fine-tune access to the internet.

Using a hub and spoke topology



Ⓐ Connected by using VPC Network Peering:
- Peering from hub to spoke-dev: Exports custom routes.
- Peering from spoke dev to hub: Imports custom routes.

Ⓑ Connected by using VPC Network Peering:
- Peering from hub to spoke-prod: Exports custom routes.
- Peering from spoke-prod to hub: Imports custom routes.

In addition to exchanging subnet routes, you can configure VPC Network Peering to export and import custom static and dynamic routes. Custom static routes that have a next hop of the default internet gateway are excluded. Custom static routes that use next-hop internal TCP/UDP load balancers are included.

You can configure a hub-and-spoke topology with your next-hop firewall virtual appliances located in the hub VPC network by doing the following:

1. In the hub VPC network, create an internal TCP/UDP load balancer with firewall virtual appliances as the backends.

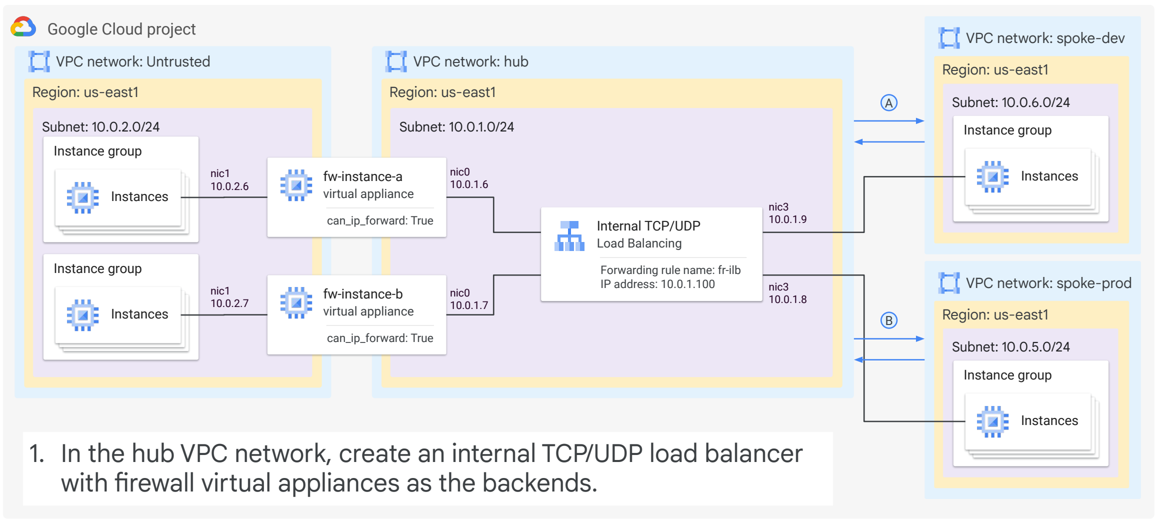
2. In the hub VPC network, create a custom static route, and set the next hop to be the internal TCP/UDP load balancer.

3. Use VPC Network Peering to connect the hub VPC network to each of the spoke VPC networks.

For each peering, configure the hub network to export its custom routes, and configure the corresponding spoke network to import custom routes. The route with the load balancer next hop is one of the routes that the hub network exports.

Subject to the routing order, the next hop firewall appliance load balancer in the hub VPC network is available in the spoke networks. If global access is enabled, the firewall appliance is available according to the routing order. If global access is disabled, then resources are only available to requestors in the same region.

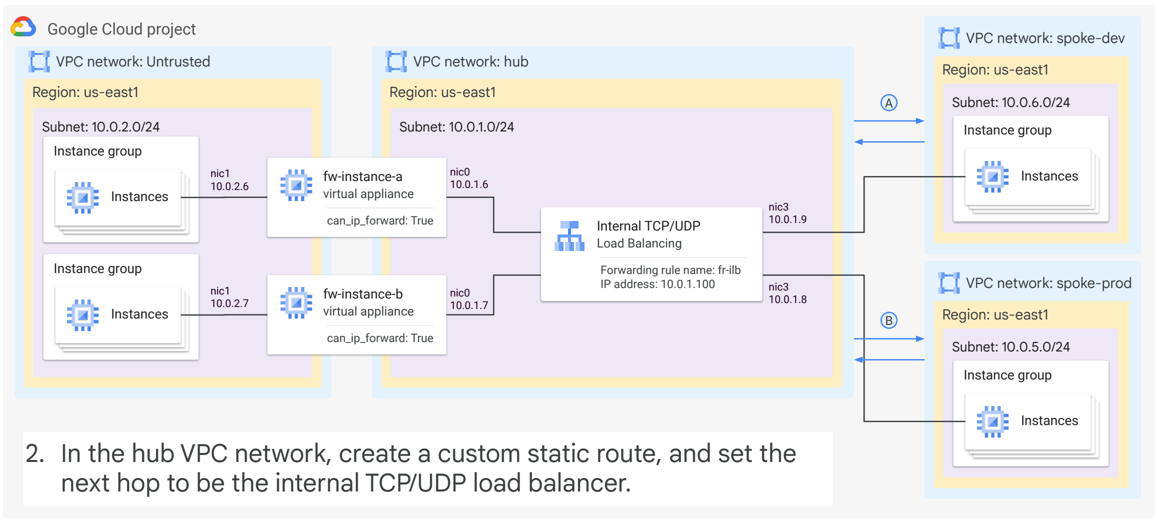
Using a hub and spoke topology



1. In the hub VPC network, create an internal TCP/UDP load balancer with firewall virtual appliances as the backends.

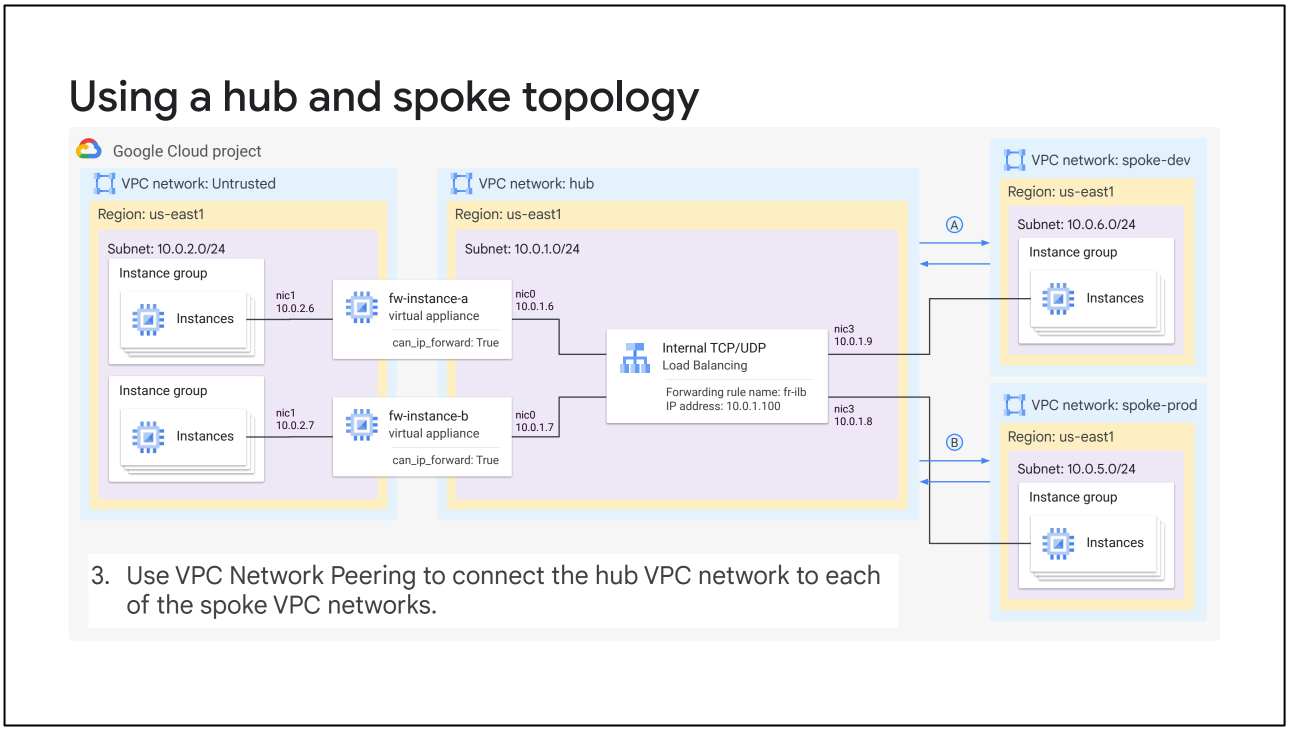
1. In the hub VPC network, create an internal TCP/UDP load balancer with firewall virtual appliances as the backends.

Using a hub and spoke topology

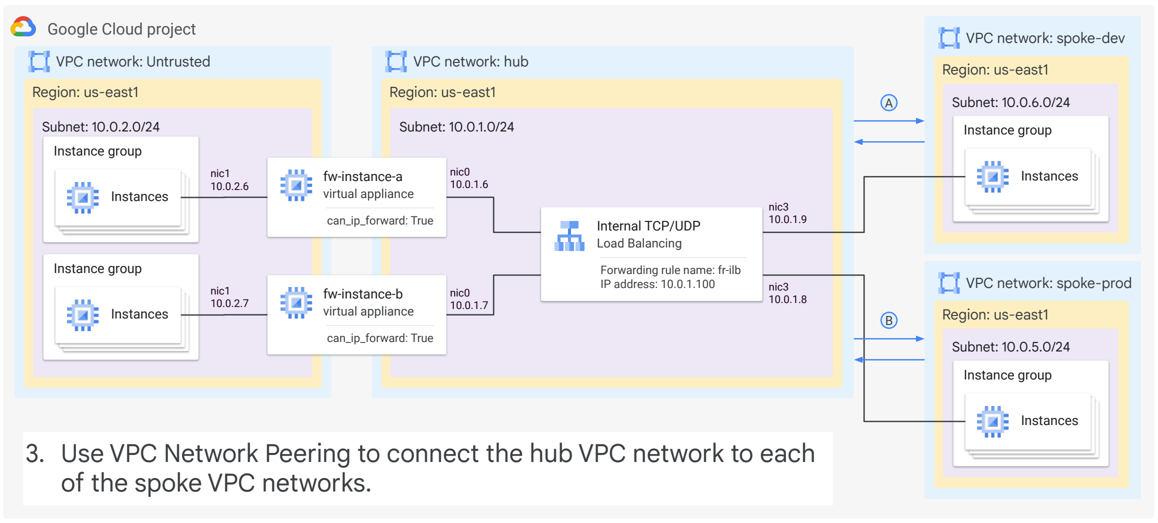


2. In the hub VPC network, create a custom static route, and set the next hop to be the internal TCP/UDP load balancer.

2. In the hub VPC network, create a custom static route, and set the next hop to be the internal TCP/UDP load balancer.



Using a hub and spoke topology

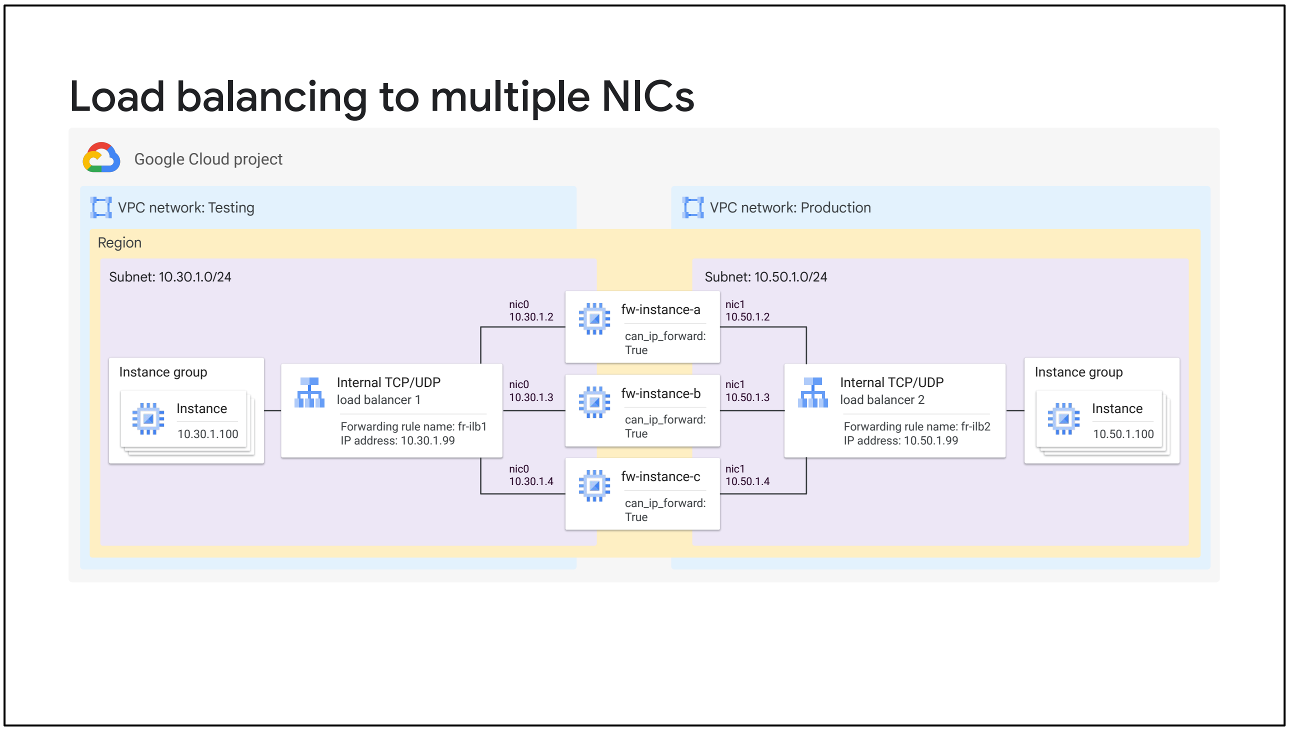


3. Use VPC Network Peering to connect the hub VPC network to each of the spoke VPC networks.

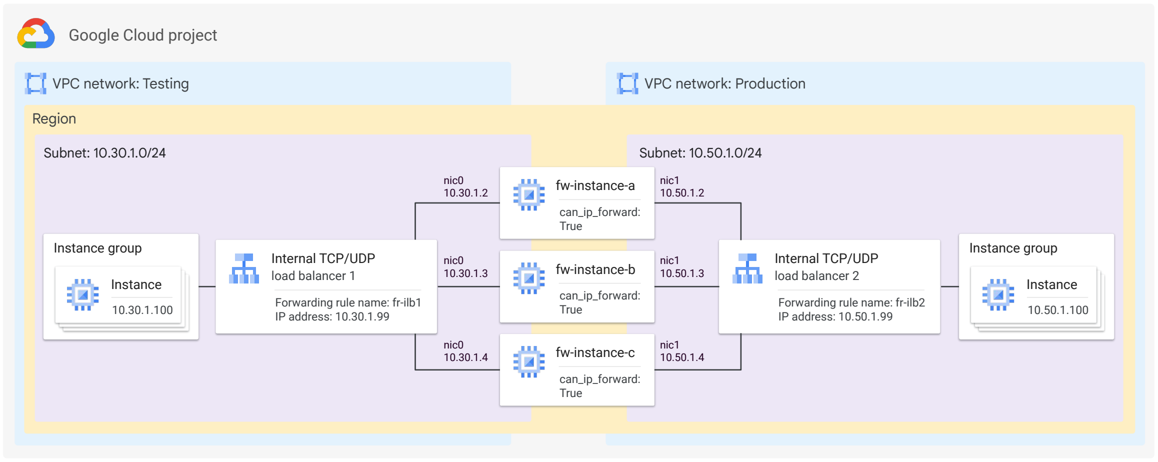
3. Use VPC Network Peering to connect the hub VPC network to each of the spoke VPC networks.

For each peering, configure the hub network to export its custom routes, and configure the corresponding spoke network to import custom routes. The route with the load balancer next hop is one of the routes that the hub network exports.

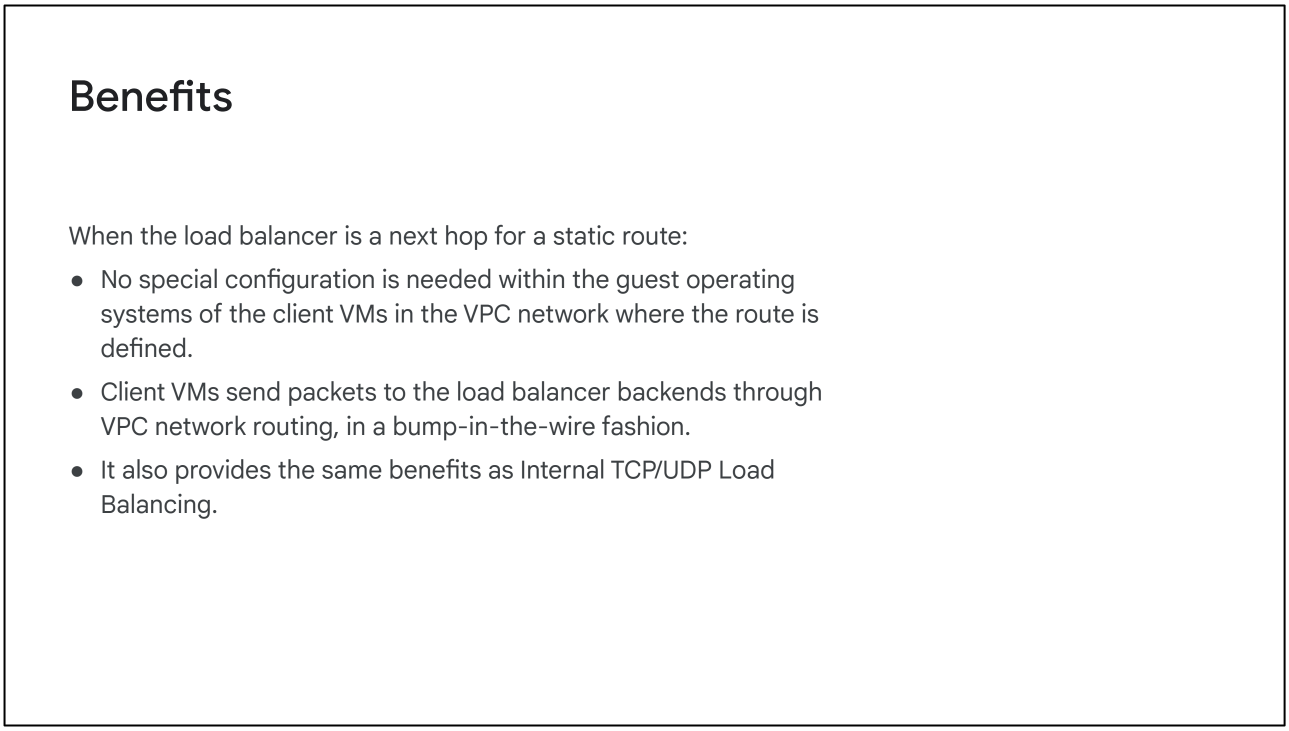
Subject to the routing order, the next hop firewall appliance load balancer in the hub VPC network is available in the spoke networks. If global access is enabled, the firewall appliance is available according to the routing order. If global access is disabled, then resources are only available to requestors in the same region.



Load balancing to multiple NICs



Internal TCP/UDP load balancer 1, shown on the left, distributes traffic from the clients to nic0, the primary interface on the backend services. The internal TCP/UDP load balancer 2, shown on the right, distributes traffic from the clients to nic1, the secondary interface on the backend services. The result is that clients can connect to the backend services through nic0 or nic1.



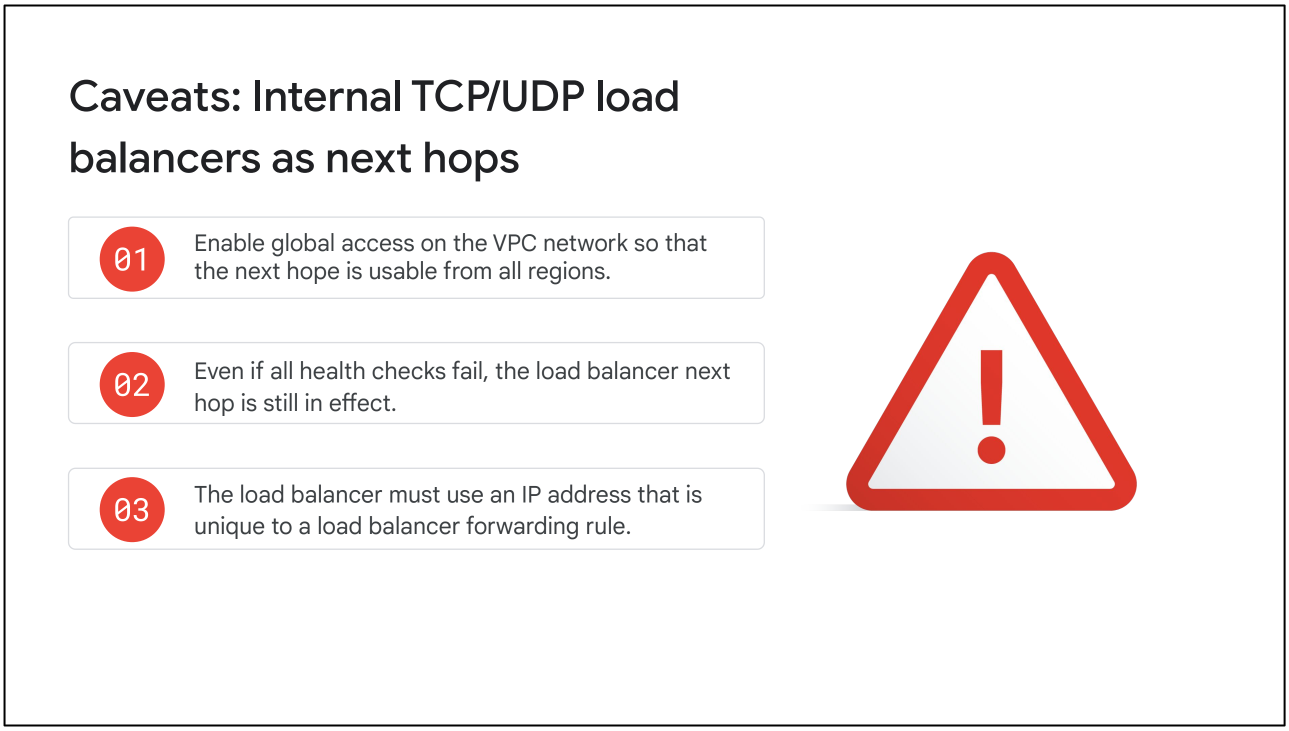
Benefits

When the load balancer is a next hop for a static route:

- No special configuration is needed within the guest operating systems of the client VMs in the VPC network where the route is defined.
- Client VMs send packets to the load balancer backends through VPC network routing, in a bump-in-the-wire fashion.
- It also provides the same benefits as Internal TCP/UDP Load Balancing.

When the load balancer is a next hop for a static route, no special configuration is needed within the client VMs. Client VMs send packets to the load balancer backends through VPC network routing, in a bump-in-the-wire fashion.

Using an internal TCP/UDP load balancer as a next hop for a static route provides the same benefits as Internal TCP/UDP Load Balancing. The health check ensures that new connections are routed to healthy backend VMs. By using a managed instance group as a backend, you can configure autoscaling to grow or shrink the set of VMs based on service demand.



Caveats: Internal TCP/UDP load balancers as next hops

01 Enable global access on the VPC network so that the next hope is usable from all regions.

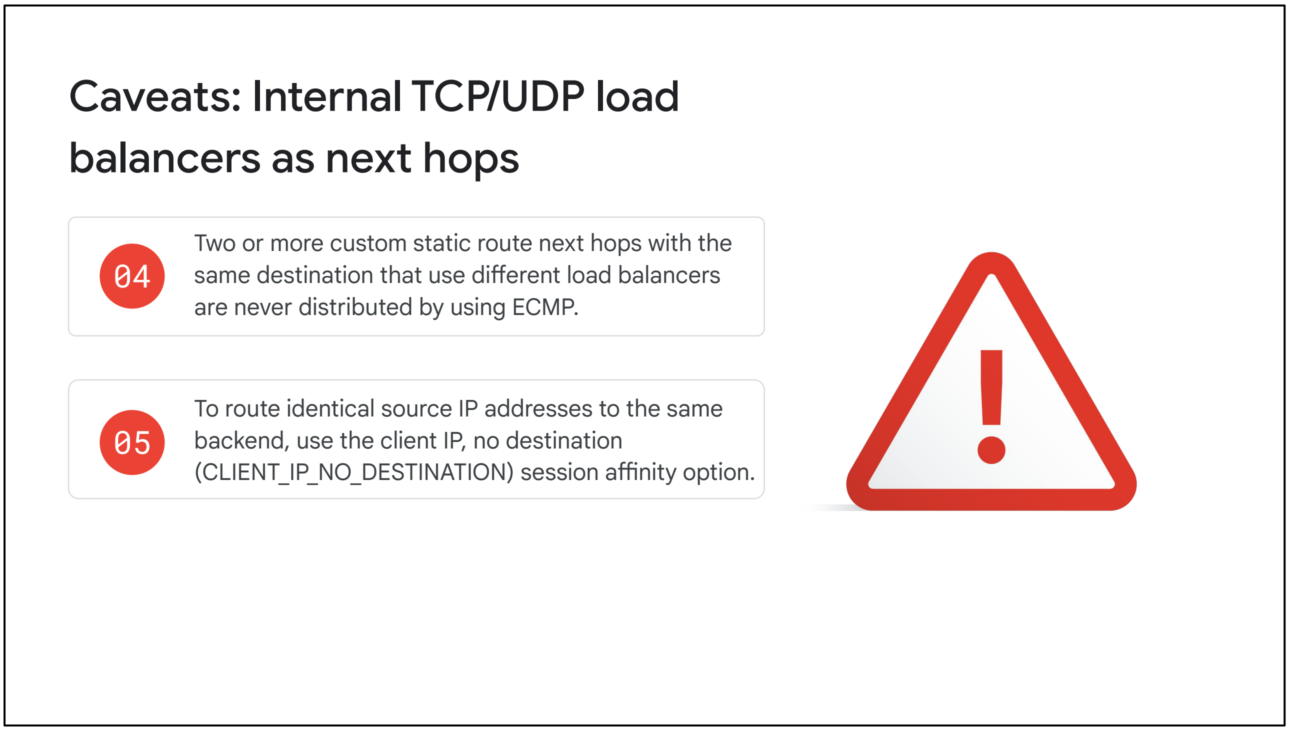
02 Even if all health checks fail, the load balancer next hop is still in effect.

03 The load balancer must use an IP address that is unique to a load balancer forwarding rule.

Using an internal TCP/UDP load balancer as a next hop comes with a few caveats. You must enable global access on the VPC network so that the next hope is usable from all regions. Whether the next hop is usable depends on the global access setting of the load balancer. With global access enabled, the load balancer next hop is accessible in all regions of the VPC network. With global access disabled, the load balancer next hop is only accessible in the same region as the load balancer. With global access disabled, packets sent from another region to a route that uses an internal TCP/UDP load balancer next hop are dropped.

Even if all health checks fail, the load balancer next hop is still in effect. Packets processed by the route are sent to one of the next hop load balancer backends. If needed, configure a failover policy.

A next hop internal TCP/UDP load balancer must use an IP address that is unique to a load balancer forwarding rule. Only one backend service is unambiguously referenced.



Caveats: Internal TCP/UDP load balancers as next hops

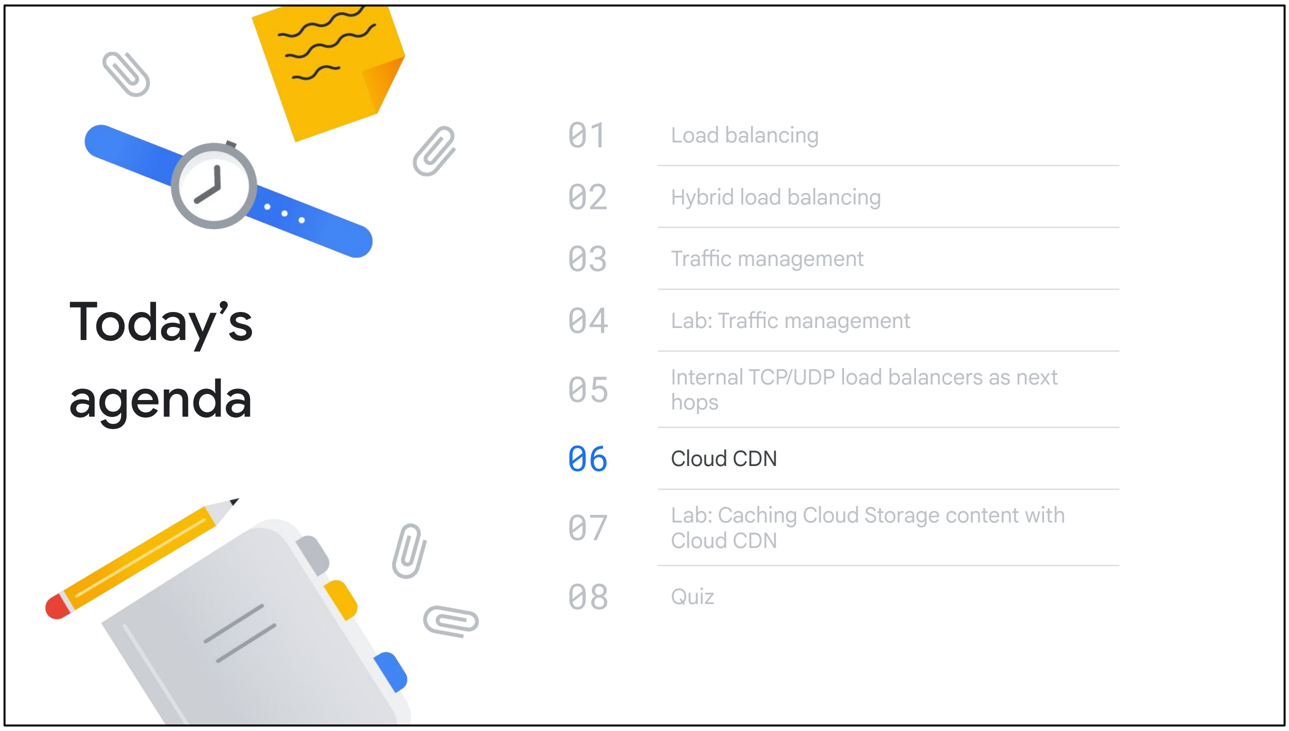
04 Two or more custom static route next hops with the same destination that use different load balancers are never distributed by using ECMP.

05 To route identical source IP addresses to the same backend, use the client IP, no destination (CLIENT_IP_NO_DESTINATION) session affinity option.

Two or more custom static route next hops with the same destination that use different load balancers are never distributed by using ECMP. If the routes have unique priorities, Google Cloud uses the next hop internal TCP/UDP load balancer from the route with the highest priority. If the routes have equal priorities, Google Cloud still selects just one next hop internal TCP/UDP load balancer.

For packets with identical source IP addresses routed to the same backend, use the client IP, no destination (CLIENT_IP_NO_DESTINATION) session affinity option.

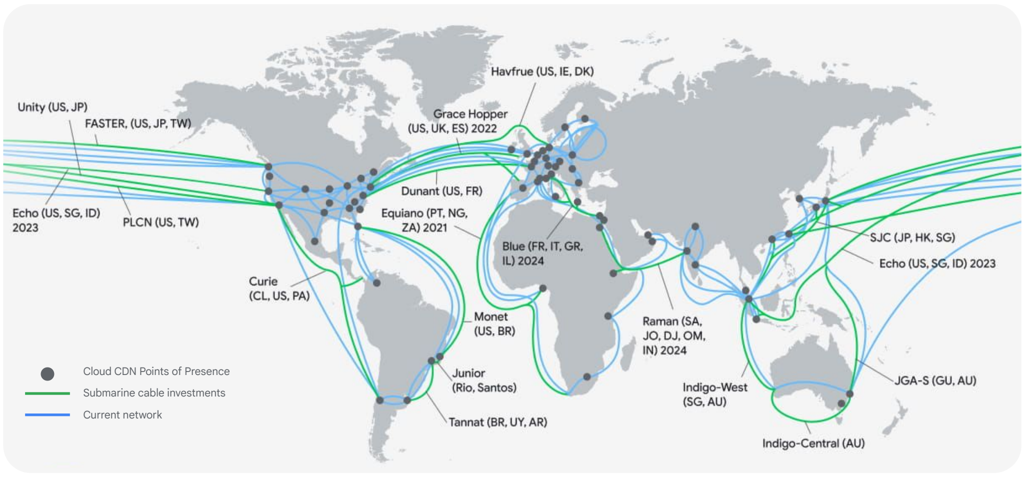
There are some additional caveats for using an internal TCP/UDP load balancer as a next hop, for example, pertaining to the use of network tags. For additional information on this and other caveats, refer to Additional considerations on the Internal TCP/UDP load balancers as next hops page in the Google Cloud documentation.



Today's agenda

Next, let's discuss Content Delivery Network (CDN), which caches content nearer to users. We'll also talk about CDN Interconnect, which lets select third-party Content Delivery Network (CDN) providers establish Direct Interconnect links at edge locations in the Google network.

Cloud CDN (Content Delivery Network)

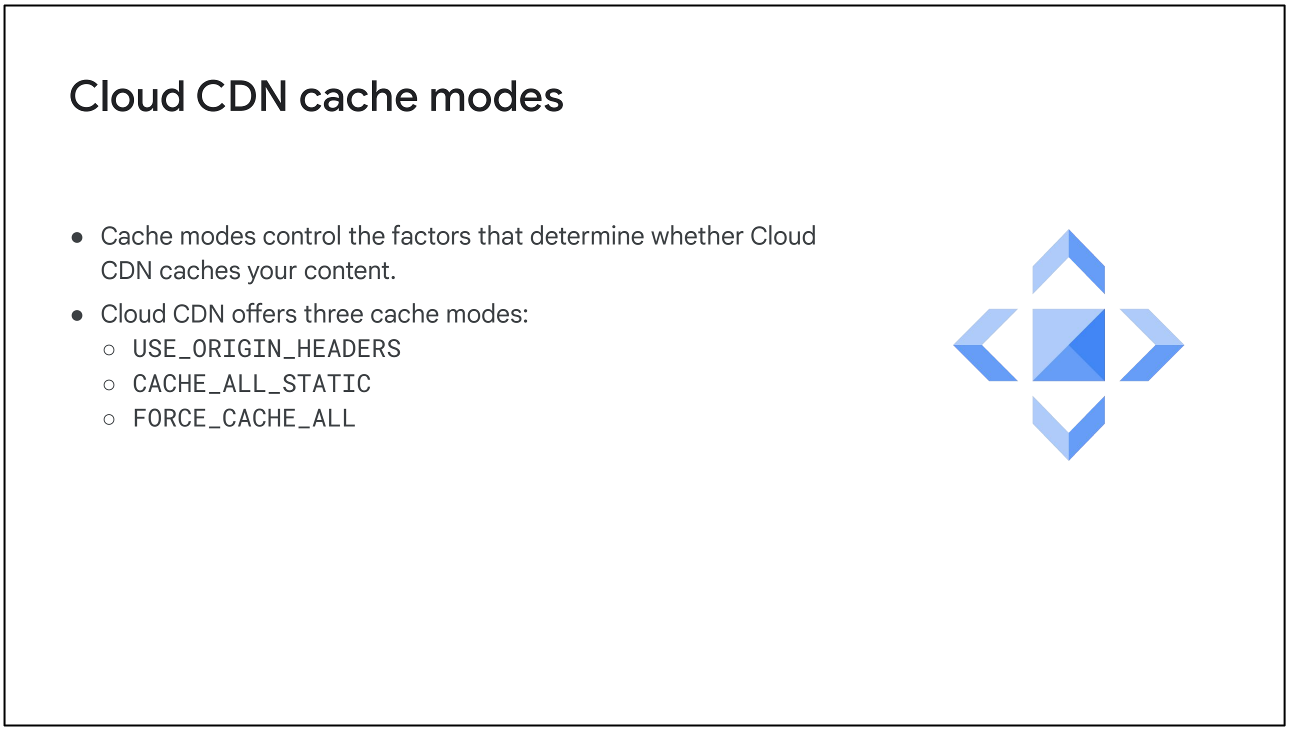


Cloud CDN (Content Delivery Network) caches content at the edges of the Google network. This caching provides faster content delivery to users while reducing transmission costs.

Content can be cached at CDN nodes as shown on this map. There are over 90 of these cache sites spread across metropolitan areas in Asia Pacific, Americas, and EMEA. For an updated list, please refer to Cache locations in the Google Cloud documentation.

For Cloud CDN performance measured by Cedexis, please refer to these reports on the Citrix website.

When setting up the backend service of a HTTP(S) load balancer, you can enable Cloud CDN with a checkbox.



Cloud CDN cache modes

- Cache modes control the factors that determine whether Cloud CDN caches your content.
- Cloud CDN offers three cache modes:
  - `USE_ORIGIN_HEADERS`
  - `CACHE_ALL_STATIC`
  - `FORCE_CACHE_ALL`

Using cache modes, you can control the factors that determine whether Cloud CDN caches your content.

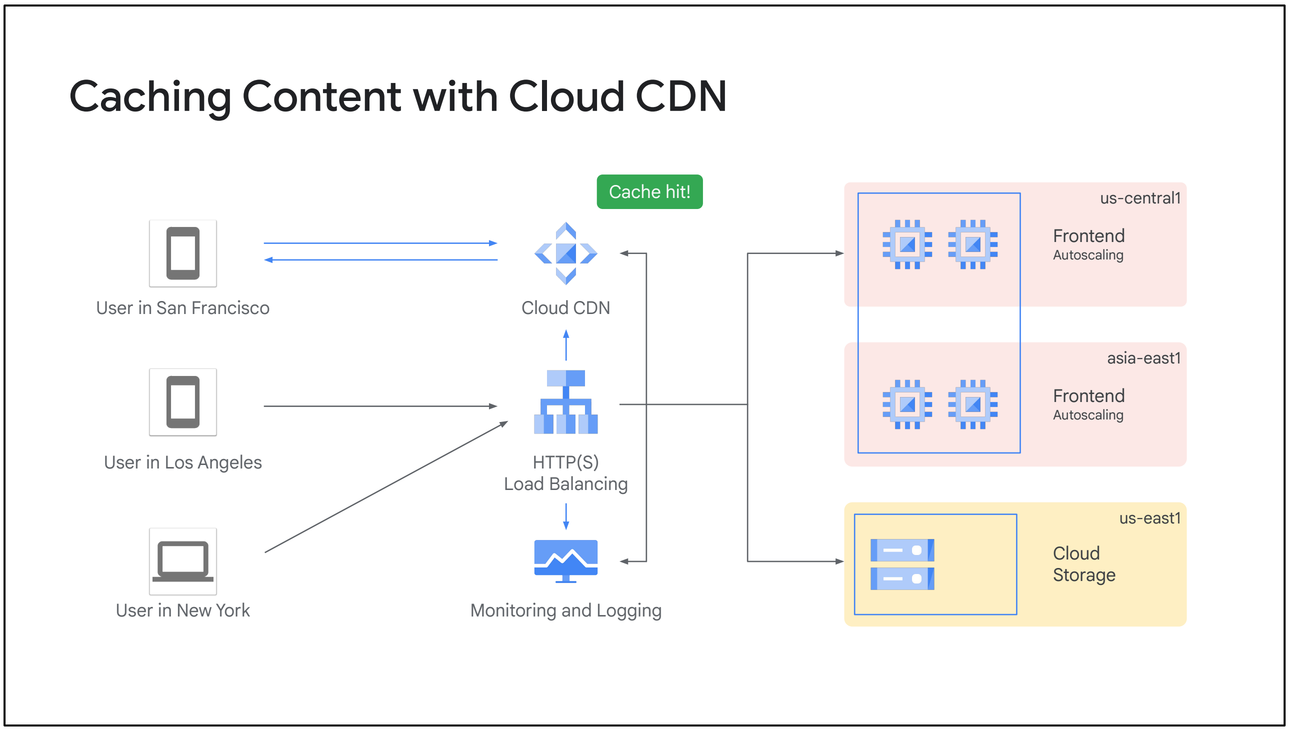
Cloud CDN offers three cache modes. The cache modes define how responses are cached, whether Cloud CDN respects cache directives sent by the origin, and how cache TTLs are applied.

The available cache modes are USE_ORIGIN_HEADERS, CACHE_ALL_STATIC and FORCE_CACHE_ALL.

USE_ORIGIN_HEADERS mode requires origin responses to set valid cache directives and valid caching headers.

CACHE_ALL_STATIC mode automatically caches static content that doesn't have the no-store, private, or no-cache directive. Origin responses that set valid caching directives are also cached.

FORCE_CACHE_ALL mode unconditionally caches responses; overriding any cache directives set by the origin. If you use a shared backend with this mode configured, ensure that you don’t cache private, per-user content (such as dynamic HTML or API responses).



Caching Content with Cloud CDN

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

In this example, the HTTP(S) load balancer has two types of backends. There are managed VM instance groups in the us-central1 and asia-east1 regions, and there's a Cloud Storage bucket in us-east1. A URL map decides which backend to send the content to: the Cloud Storage bucket could be used to serve static content and the instance groups could handle PHP traffic.

When a user in San Francisco is the first to access content, the cache site in San Francisco sees that it can't fulfill the request. This situation is called a cache miss. If content is in a nearby cache, Cloud CDN might attempt to get the content from it, for example if a user in Los Angeles has already accessed the content. Otherwise, the request is forwarded to the HTTP(S) load balancer, which in turn forwards the request to one of your backends.

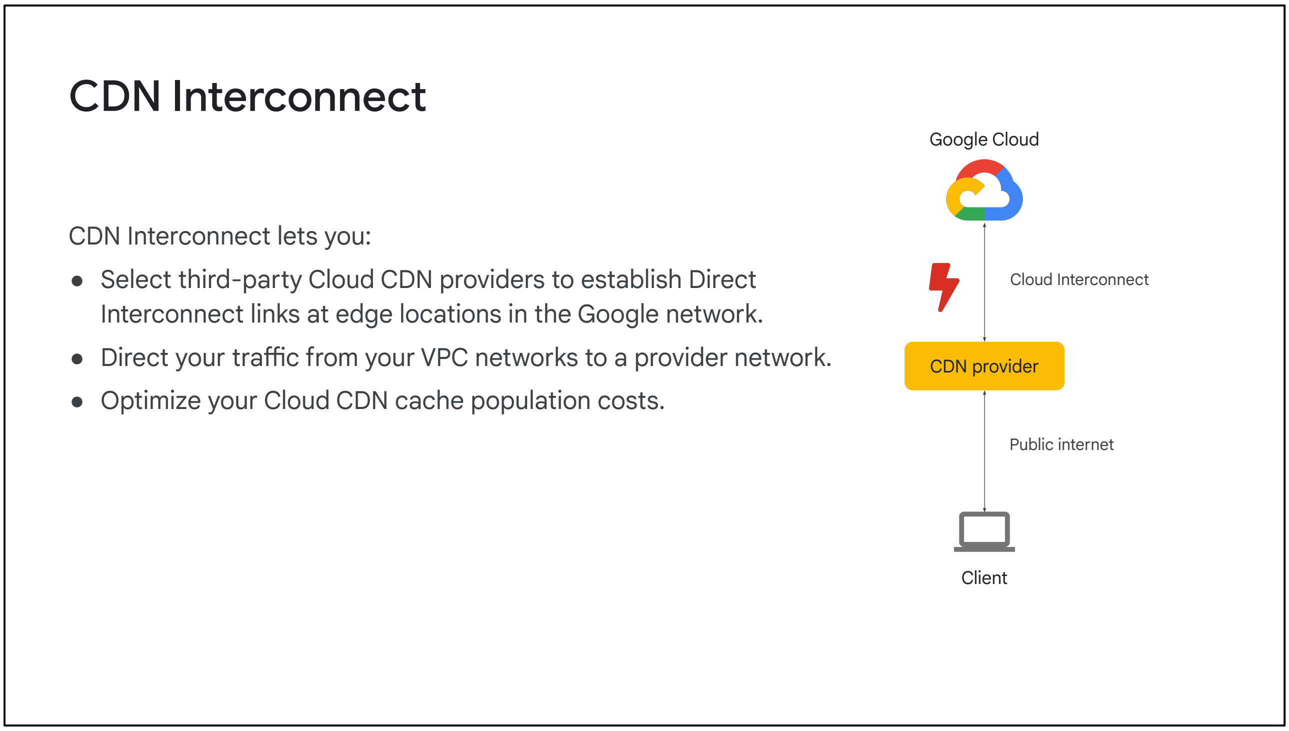
Depending on the content that is being served, the request will be forwarded to the us-central1 instance group or the us-east1 storage bucket.

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

For more information on what content can be cached, please refer to Caching overview in the Google Cloud documentation.

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

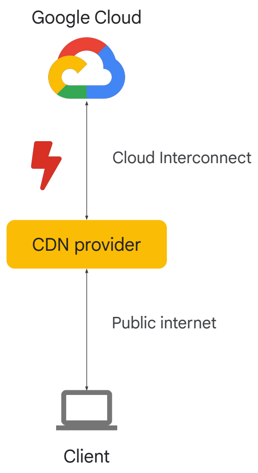
Cache modes let you control how content is cached.



CDN Interconnect

CDN Interconnect lets you:

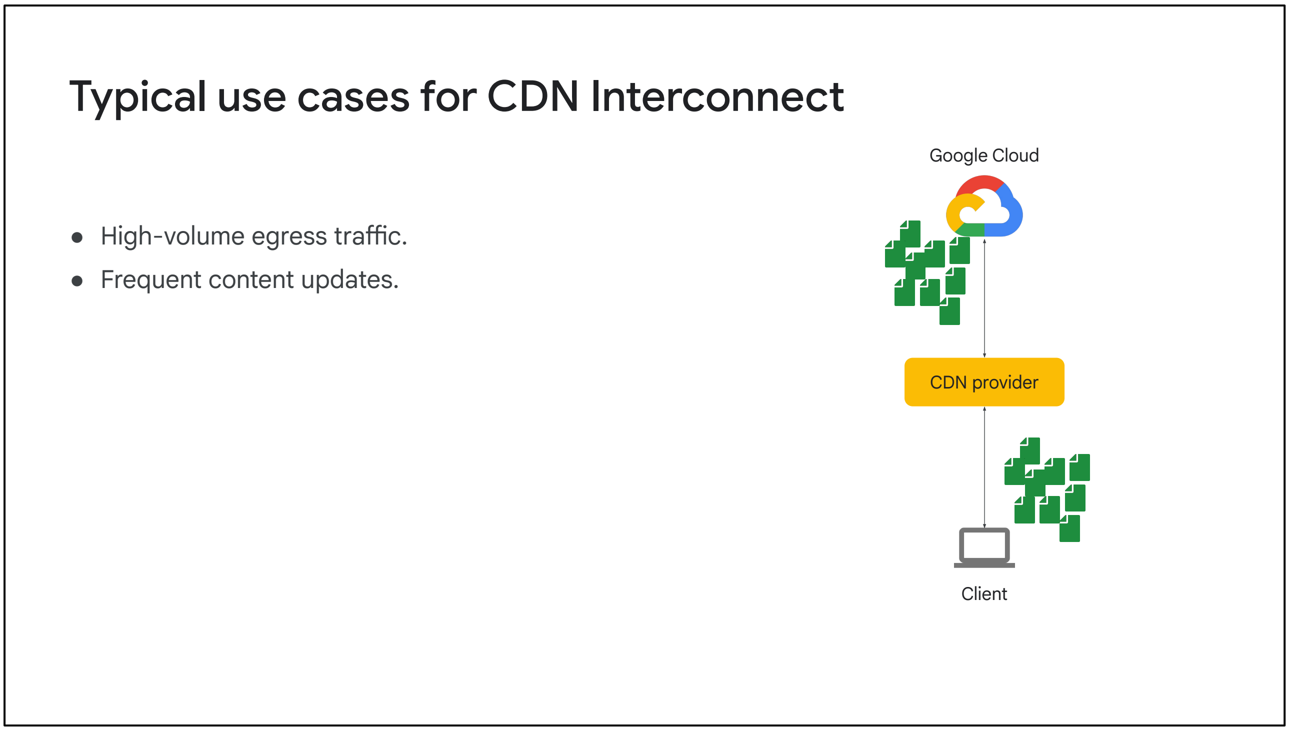
- Select third-party Cloud CDN providers to establish Direct Interconnect links at edge locations in the Google network.
- Direct your traffic from your VPC networks to a provider network.
- Optimize your Cloud CDN cache population costs.



CDN Interconnect lets select third-party Content Delivery Network (CDN) providers establish Direct Interconnect links at edge locations in the Google network. These connections let you direct your traffic from your VPC networks to a CDN provider network. For a complete list of CDN providers, refer to [CDN Interconnect overview] in the Google Cloud documentation.

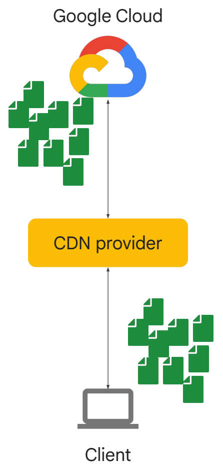
CDN Interconnect lets you connect directly to select CDN providers from Google Cloud. Your network traffic that egresses from Google Cloud through one of these links benefits from the direct connectivity to supported CDN providers.

The billing charges that CDN Interconnect reduces your Cloud CDN cache population costs.



Typical use cases for CDN Interconnect

- High-volume egress traffic.
- Frequent content updates.



If you have a high-volume of egress traffic, consider using CDN Interconnect. You can use the CDN Interconnect links between Google Cloud and selected providers to automatically optimize the egress traffic and save money. If you're populating the Cloud CDN cache locations with large data files from Google Cloud, this optimization can be especially helpful.

Frequent content updates are another typical CDN Interconnect use case. Cloud workloads that frequently update data stored in Cloud CDN cache locations benefit from using CDN Interconnect. The direct link to the Cloud CDN provider reduces latency.

CDN Interconnect traffic billing

- Ingress traffic is free for all regions.
- Egress traffic rates apply only to data that leaves Compute Engine or Cloud Storage.
- The reduced price applies only to IPv4 traffic.
- Egress charges for CDN Interconnect appear on the invoice as *Compute Engine Network Egress via Carrier Peering Network*.

Ingress traffic is free for all regions.

Egress traffic rates apply only to data that leaves Compute Engine or Cloud Storage. Egress charges for CDN Interconnect appear on the invoice as *Compute Engine Network Egress via Carrier Peering Network*.

The special pricing for your traffic that egresses from Google Cloud to a CDN provider is automatic. Google works with approved CDN partners in supported locations to accept provider IP addresses. Any data that you send to your allowlisted CDN provider from Google Cloud is charged at the reduced price.

This reduced price applies only to IPv4 traffic. It does not apply to IPv6 traffic.

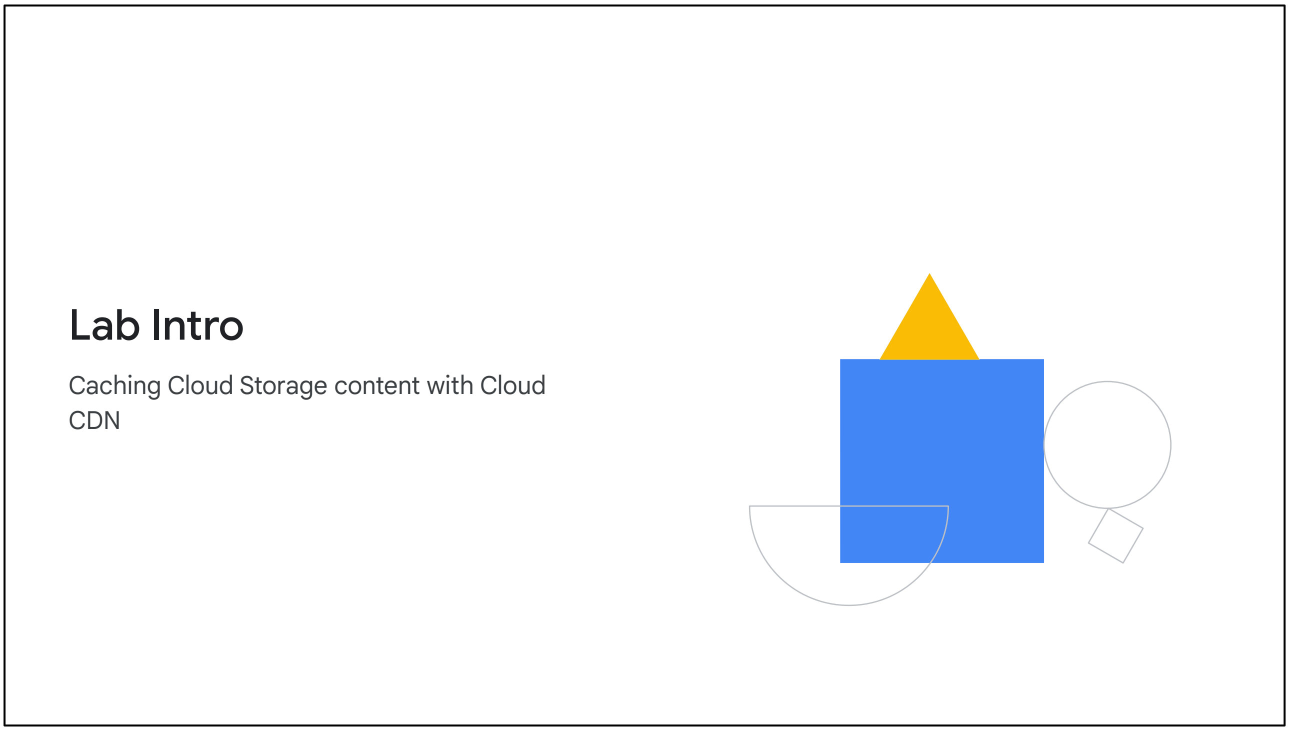
Intra-region pricing for CDN Interconnect applies only to intra-region egress traffic that is sent to Google-approved CDN providers at specific locations.

Setting up CDN Interconnect

- CDN Interconnect does not require any configuration or integration with Cloud Load Balancing.
- Work with your supported CDN provider to:
  - Learn which locations are supported.
  - Correctly configure your deployment to use intra-region egress routes.

CDN Interconnect does not require any configuration or integration with Cloud Load Balancing. If your CDN provider is already part of the program, you don't need to do anything. Traffic from supported Google Cloud locations to your CDN provider automatically benefits from the direct connection and reduced pricing.

Work with your supported CDN provider to learn what locations are supported. Your supported CDN service provider can also help you correctly configure your deployment to use intra-region egress routes, which cost less than inter-region egress traffic.



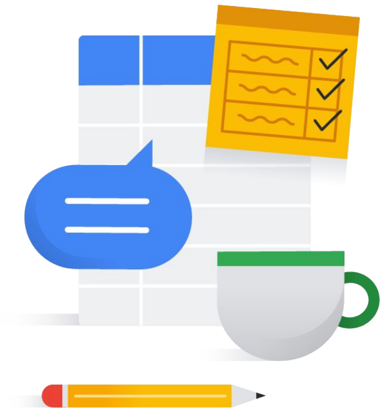
Lab Intro

Caching Cloud Storage content with Cloud CDN

In this lab, you learn how to perform the following tasks:

- Create and populate a Cloud Storage bucket.
- Create an HTTP load balancer with Cloud CDN.
- Verify the caching of your bucket's content.
- Invalidate the cached content.

Debrief



In this module, we began with an overview of load balancing in Google Cloud. We continued with a discussion of hybrid load balancing. You learned that hybrid load balancing can be used to migrate your workloads into Google Cloud or to provide multiple platforms for your workloads. We covered the load balancers that support hybrid load balancing and an overview of the components that you must configure.

We then talked about using traffic management with your load balancers. You learned which load balancers support traffic management features. You were introduced to the URL map, where you configure traffic management features. We walked through a simple example of traffic management. In the example, you saw how to configure a URL map to match against incoming traffic and specify where the traffic should be sent.

You then applied what you learned in a lab exercise.

Next, we covered using internal TCP/UDP load balancers as next hops. You learned about some of the major use cases, and some simple topologies were shown.

We continued by discussing using Cloud CDN to get content to your clients faster. You also learned about CDN Interconnect to direct traffic from your VPC networks to a supported CDN provider network. You also learned how CDN Interconnect optimizes your Cloud CDN cache population costs.

Finally, you took a brief quiz to test your knowledge.