Understand Cloud Load Balancer Like a Senior Engineer

Shiqiu Zhang Follow
Nov 1 · 10 min read

Software Load Balancer (SLB)

Most cloud load balancers developers used today are software load balancers. They're not only used to balance network traffic, but almost any computational resources, including CPU resource, memory and disk space resource, networking resource, etc.

Sometimes people just call them proxy, especially in the context of cloud-native or service mesh, since a proxy takes network forwarding as its main function. A proxy / load balancer is primarily responsible for three functions:

- **endpoint discover**
- **health check**
- **load balancing**

L4/L7 Load Balancing

The industry's load balancing solutions are basically two types: Layer 4 load balancer and Layer 7 load balancer. Layer 4 and Layer 7 refer to the layers in the OSI model. Layer 4 is the transport layer and Layer 7 is the application layer.

	Data Unit	Layer	Function
Host Layers	Data	7. Application	Network process to application
		6. Presentation	Data representation and encryption
		5. Session	Interhost communication
	Segments	4. Transport	End-toEnd Connections and Reliability
Media Kayers	Packets	3. Network	Path determination and IP (Logical addressing)
	Frames	2. Data Link	MAC and LLC (Physical addressing)
	Bit	1. Physical	Media, Signal and Binary transmission

OSI model from wikipedia

L4 Load Balancer

L4 load balancing doesn't mean a load balancer will work at Layer 4. In fact, the majority of L4 load balancers still work at Layer 2 and 3. Sometimes you may hear people talk about Layer 2 load balancing or Layer 3 load balancing. They are all called Layer 4 load balancing now.

An L4 load balancer maintains the same TCP connection between upstream and downstream. It will ensure that traffic from the same source is always routed to the same backend. So what happens under the hood? Let's revisit the seven layers in the OSI model.

The first layer is the physical layer, which transmits bitstreams. Devices in this layer, including NICs and various cables fiber, can communicate directly with cross wires.

The second layer is the data link layer, which transmits data frames. WIFI (802.11), Ethernet (802.3), and PPP work here. The switch works at this layer and maintains an address table to associate the MAC address of each device with the LAN port. Devices in the same LAN can communicate through switches.

The third layer is the network layer, which transmits Packets. OSPF, IS-IS, BGP, and other routing protocols work in this layer. Devices can communicate through IPv4/IPv6 addressing with the router's help. Network packets can now transmit over the entire WAN.

The fourth layer is called the transport layer, which transmits segments. TCP / UDP protocols work here and they know the port number of the host. The IP identifies the host's location on the Internet, while the port identifies the program listening to that port on that host.

The first three layers are called Media layers, while the top four layers are called Host layers. In the first three layers, NICs can process traffic and forward them to the next hop. From Layer 4, the traffic will hit the Linux kernel networking stack, which has worse performance than NIC processing (we do have some bypass-kernel options, but that's beyond the scope of today's discussion). In general, lower layers perform better when processing packets. That's why L4 load balancing is mainly working on Layer 2 and 3.

Layer 2 Load Balancing

There are actually two ways to perform a forwarding behavior: packet encapsulation & rewriting.

At Layer 2, it's mostly rewriting. If you were to take a detailed look at the standard 802.3 frame structure, it would look like this

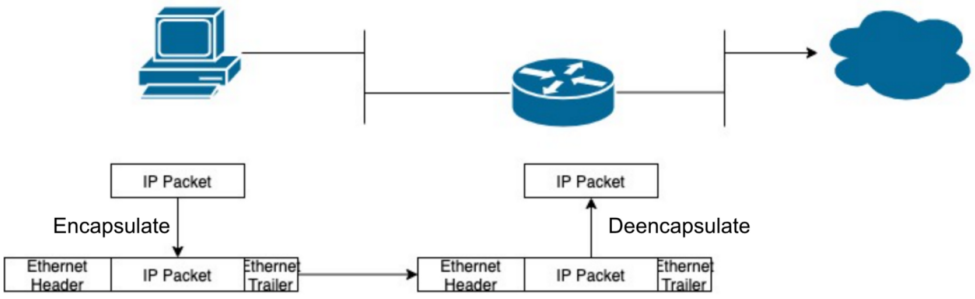
IEEE 802.3 Frame Structure

IEEE 802.3 Frame Structure

Destionation MAC address	Source MAC address	Length	Payload	CRC

802.3 Frame structure

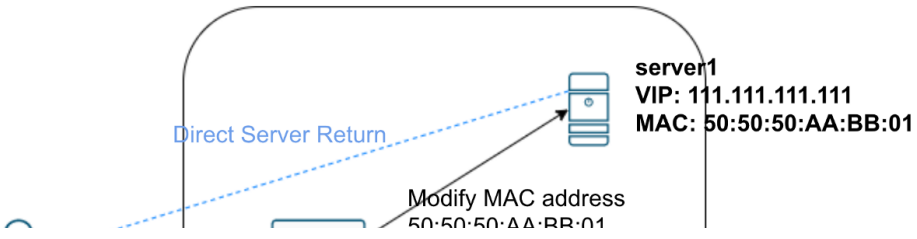
We can ignore the other fields for now and focus on the source MAC address and destination MAC address. The switch will associate the MAC address with the LAN port and forward data frames according to address table. The problem is if we only rely on the switch to forward traffic, the packet will never cross the subnet. So the router will actually encapsulate and deencapsulate the packet.

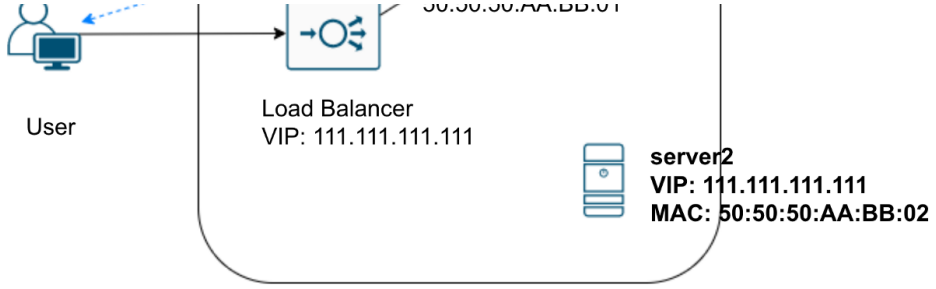


L2 Packet ReWrite

Once the router receives the network packet, it first deencapsulates it, removes the original Layer 2 header and trailer (Ethernet header contains the source and destination MAC addresses), then uses the routing table to determine the next hop address and re-encapsulates the packet. The router will use its own MAC address as the new source MAC address and the next hop address as the new destination MAC address, and encapsulate the packet with a new Layer 2 header and trailer.

Layer 2 load balancer working in the same way. For instance, Google used Netscalers as Layer 2 load balancers, which was later replaced by Maglev. A Virtual IP Address (VIP) will be given to all real servers as well as the load balancer. When the user sends a request to the VIP, it first hits the L2 load balancer, which determines the real server to forward the request to and modifies the destination MAC address in the data frame.





L2 DSR (Direct Server Return)

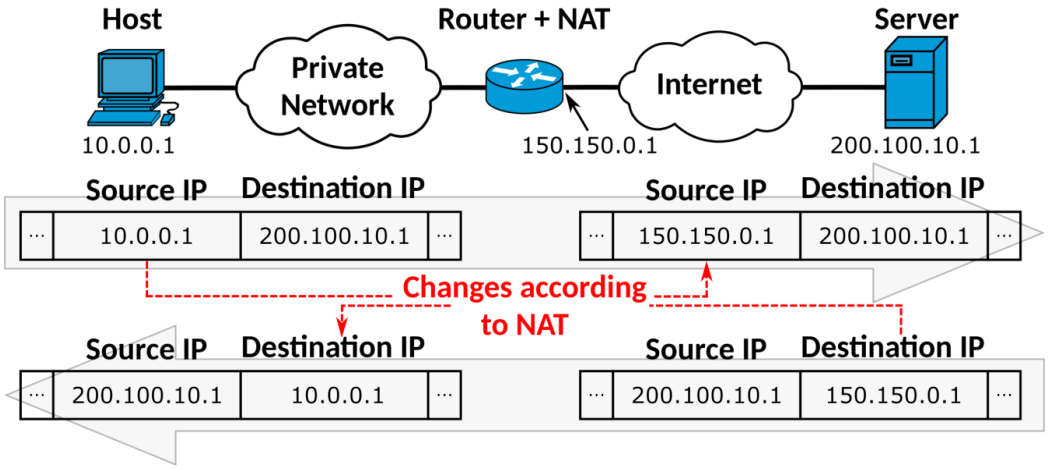
Since the Layer 2 load balancer only alters the destination MAC address of the frame during the forwarding and does not involve the higher layer protocols (no modification of the Payload's contents), it appears to the higher layer (Layer 3) that all the data has remained unchanged.

Since the request packet destination IP matches the server IP, servers can directly transmit the response without passing through the load balancer again. That is called **L2 DSR (Layer 2 Direct Server Return)**.

Layer 3 Load Balancing

Layer 3 can handle more sophisticated load balancing tasks. Network Address Translation (NAT) and IP Encapsulation are the two most common forms.

Your home router uses **network address translation (NAT)** every day to convert the private IP addresses to a public IP address. Similarly, Layer 3 load balancer can also rewrite the IP header to modify the source and destination IP addresses and redirect the traffic to the appropriate backend server. The cost is the load balancer itself has to handle all the traffic, resulting in a performance bottleneck.

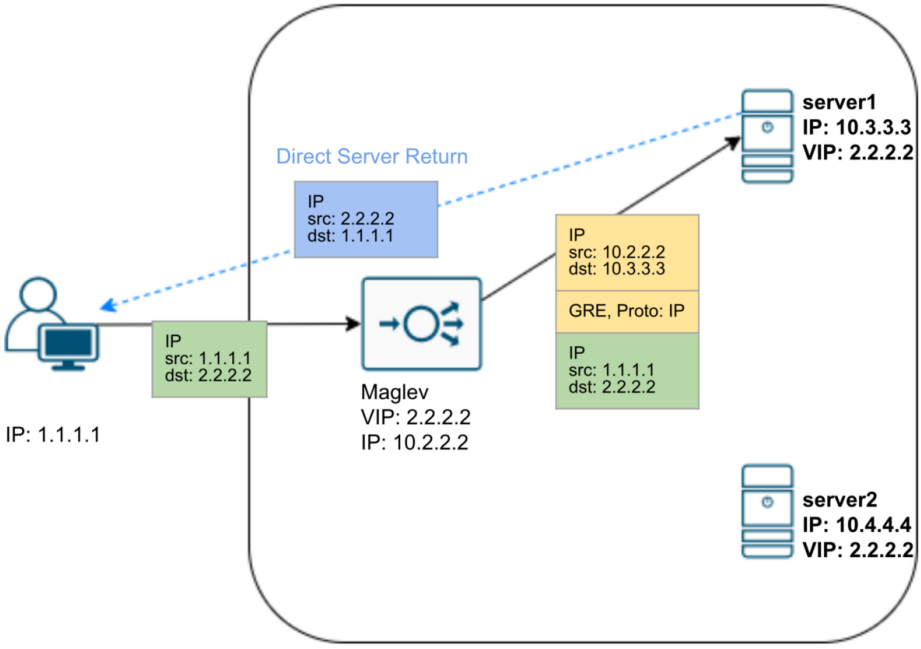


Network address translation between a private network and the Internet (from wikepedia)

Another option is **IP Encapsulation**. It treats the existing IP header and payload as a full payload, then adds a new IP header. For instance, Google and Google Cloud use Maglev

as L4 TCP/UDP External Load Balancer.

According to the Maglev paper (published in 2016), every backend server will have one or more Virtual IP addresses (VIPs). When the Maglev gets a packet to VIP, it chooses a service endpoint associated with that VIP, and encapsulates the packet using **Generic Routing Encapsulation (GRE)** by adding an outer IP header destined to the endpoint. The backend server will decapsulate and consume the client request packet and send back the response. Google uses DSR in Layer 3 as well. So the response packet will have the server VIP as source address and user's IP as destination address.



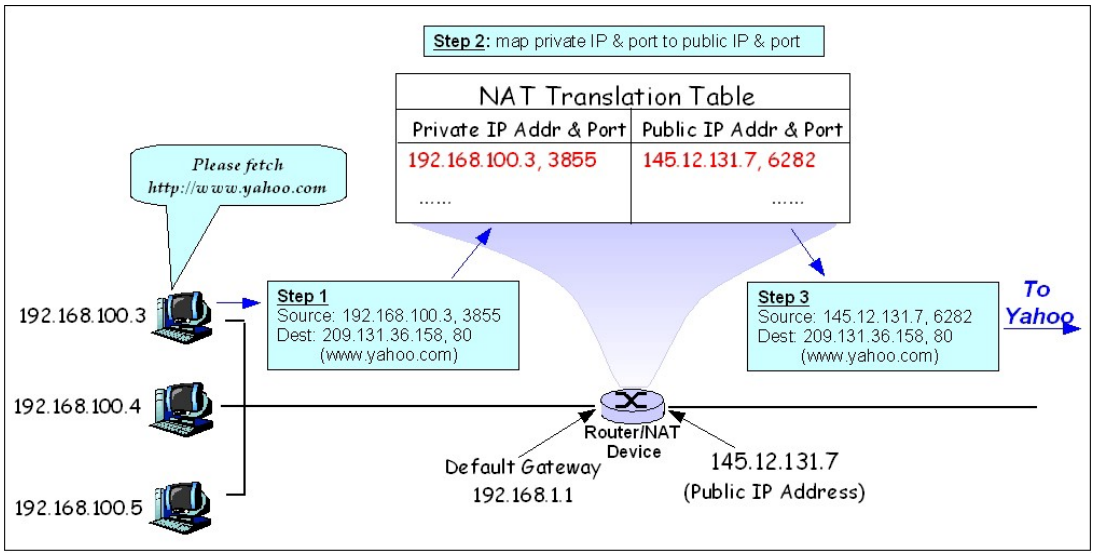
Layer 3 Direct Server Return

DSR load balancer only handles inbound traffic. When you search something on google.com, you request maybe just take a few KB, but the response can take hundreds or even thousands of KB, especially if there are some graphs in the results. With DSR, we can save 99% of the load balancing bandwidth.

Layer 4 Load Balancing

The Layer 4 load balancing will include port numbers into the loop. Instead of using IP address, Layer 4 load balancer will use IP:Port. In fact, this scenario is quite common and can be easily found on your home routers. We discussed how NAT translates private addresses to public addresses at Layer 3. However, how would several private addresses be mapped to one public address, if you have multiple devices at home but only one public address to the outside world?

It turns out NAT mode on the home router was actually NAPT (Network Address and Port Translation), which will map requests from different private IP addresses to different ports of the same public IP. The available port numbers range from 0 to 65535, which is more than enough for most residential scenarios.



Network Address and Port Translation (from wikepedia)

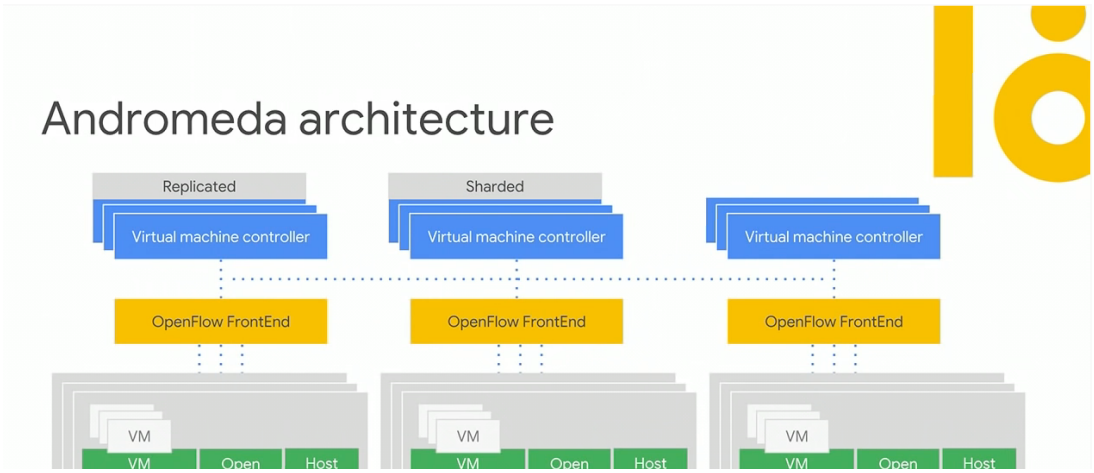
If you recall the paragraph about the seven layers in the OSI model at the beginning of this article, you could easily see that port numbers are not included in the IP protocol (network layer), but in the TCP/UDP protocol (transport layer). Therefore, rewriting ports will take place on Layer 4.

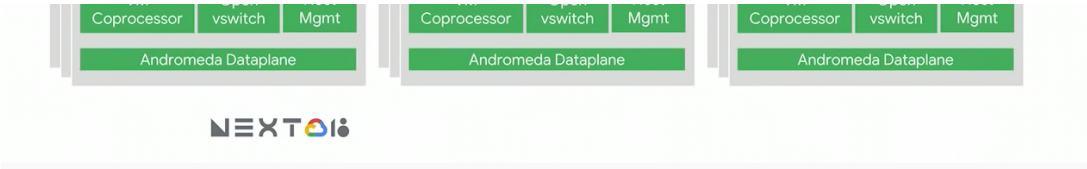
Google Cloud L4 Load Balancer

Usually, cloud companies have a more descriptive product name rather than Layer 4 or Layer 7. If you visit Google Cloud user guide pages, they are called TCP/UDP load balancers and HTTP(s) load balancers.

Load balancers can be categorized as internal load balancers and external load balancers. External load balancers require public IP addresses, whereas internal load balancers only work within Virtual Private Cloud (VPC). Google Cloud uses Maglev as External TCP/UDP Network Load Balancer, which includes all of the Layer 2/3/4 load balancing functions mentioned above.

Google Cloud Internal TCP/UDP load balancer is built on the Andromeda network virtualization stack. All the Layer 3/4 activities are now managed by SDN (software-defined networking).



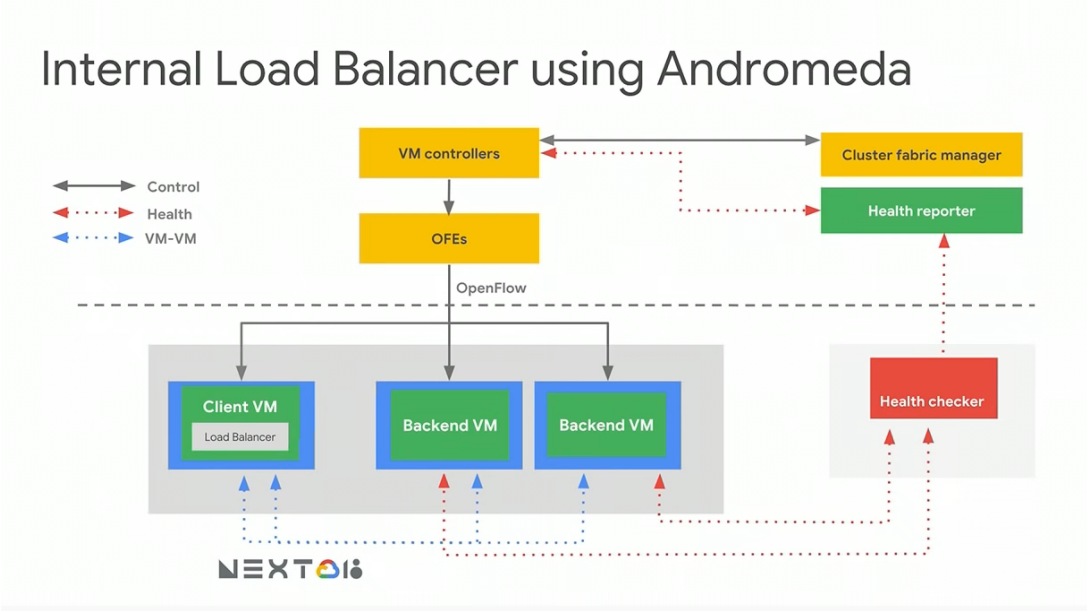


Andromeda architecture (from Cloud Next'18)

To be precise, Layer 4 internal load balancing is one of the virtualized network functions (VNF) provided by Andromeda.

Andromeda uses health checker service to check the endpoints of the Backend VMs and passes the endpoint-related information to the VM controller (VMC) via the health reporter. The VM controller will program that information to OpenFlow Front End (OFE).

OpenFlow Front End will announce routing information to the virtual switch (Open vSwitch, OVS) inside VM Host via the OpenFlow protocol. When the client VM sends a request to the backend VM, the packet will first pass through the virtual switch before leaving the VM Host. Then virtual switch will pick a healthy backend VM and direct the traffic to there, much like a real load balancer would. In other words, there is no middlebox load balancer. The traffic flows VM-to-VM, which can avoid the chokepoint and improve the network performance.



Internal Load Balancer using Andromeda (from Cloud Next'18)

L7 Load Balancer

L4 load balancer works mainly by forwarding TCP traffic, but the same TCP channel is maintained between the client and the server. While L7 load balancer works as a proxy, where one connection is established between the client and the load balancer, and another is maintained between the load balancer and the server. An L7 load balancer can be thought as an additional server between the client and the real backend server,

accepting requests from the client and making an initial determination based on the content of the request before routing the request to the real server.

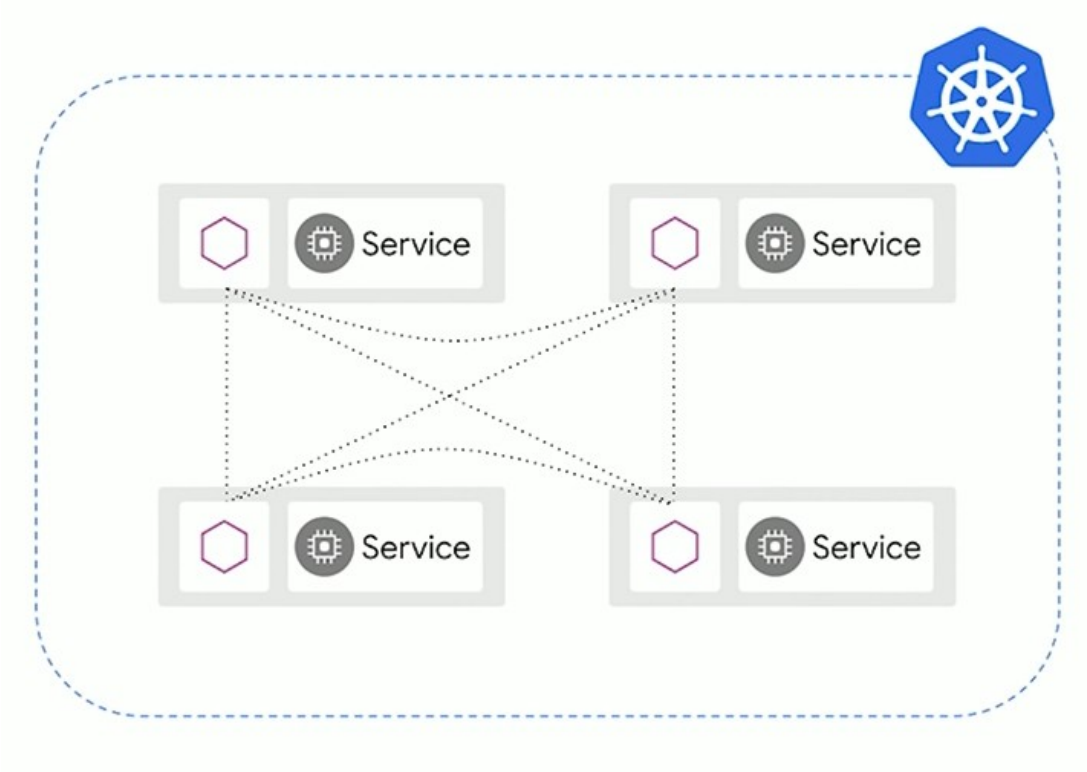
There are some requirements that L4 load balancer doesn't support and L7 load balancer does. L7 load balancer, for example, can redirect traffic to different services based on the URL path, which is a common use case in the context of microservice.

By evaluating the HTTP(s) request content, the L7 load balancer can provide more advanced capabilities. The cost, however, is efficiency, since L7 load balancer will have to go through another round of TCP handshakes. In a microservice architecture, L7 load balancer / proxy would be the cornerstone for traffic management.

Google Cloud L7 Load Balancing

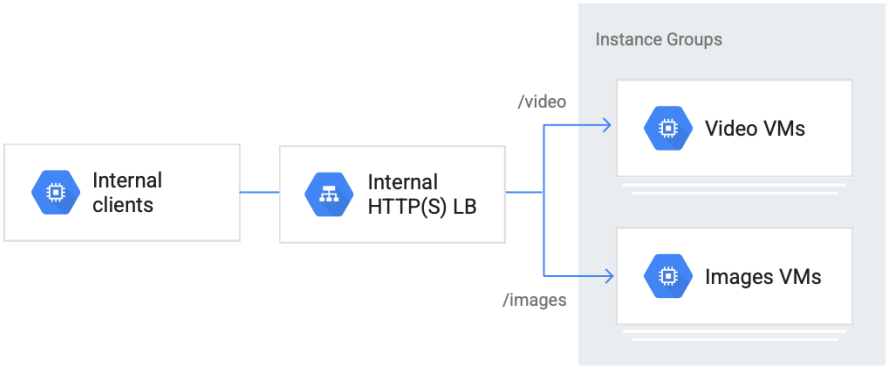
Google Cloud Internal HTTP(S) Load Balancing (L7 Load Balancing) is a managed service based on the Envoy proxy.

Envoy proxy can be used as a sidecar proxy. In a micro-service scenario, services are split into different individual units. Sidecar proxies (Envoy) sit next to workloads (they can run in the same VMs, or Kubernetes Pods). Proxies mediate all inbound/outbound (HTTP, REST, gRPC, Redis, and so on) traffic. As a result, the service instance is only aware of the local proxy and not the entire network.



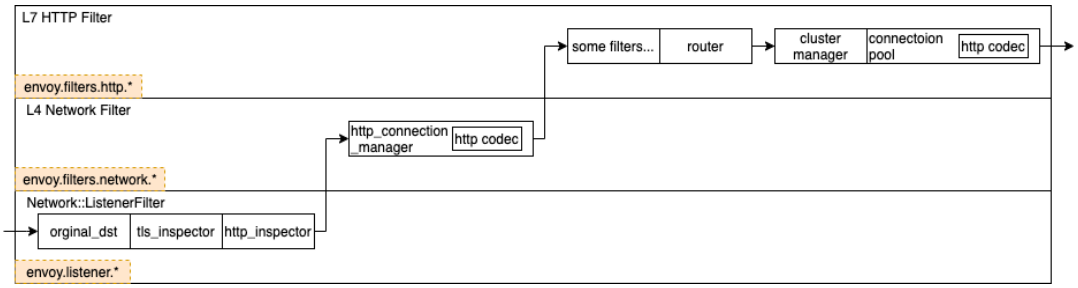
Sidecar proxy (from Cloud Next'19)

In addition to the sidecar proxy, Envoy can also be used as an L7 load balancer. Here is a sample use case provided by Google Cloud Guide. In this diagram, the Internal HTTP(S) Load Balancing is a managed service based on the open source Envoy proxy.



Example from Google Cloud: Internal (micro) services with Layer 7-based load balancing

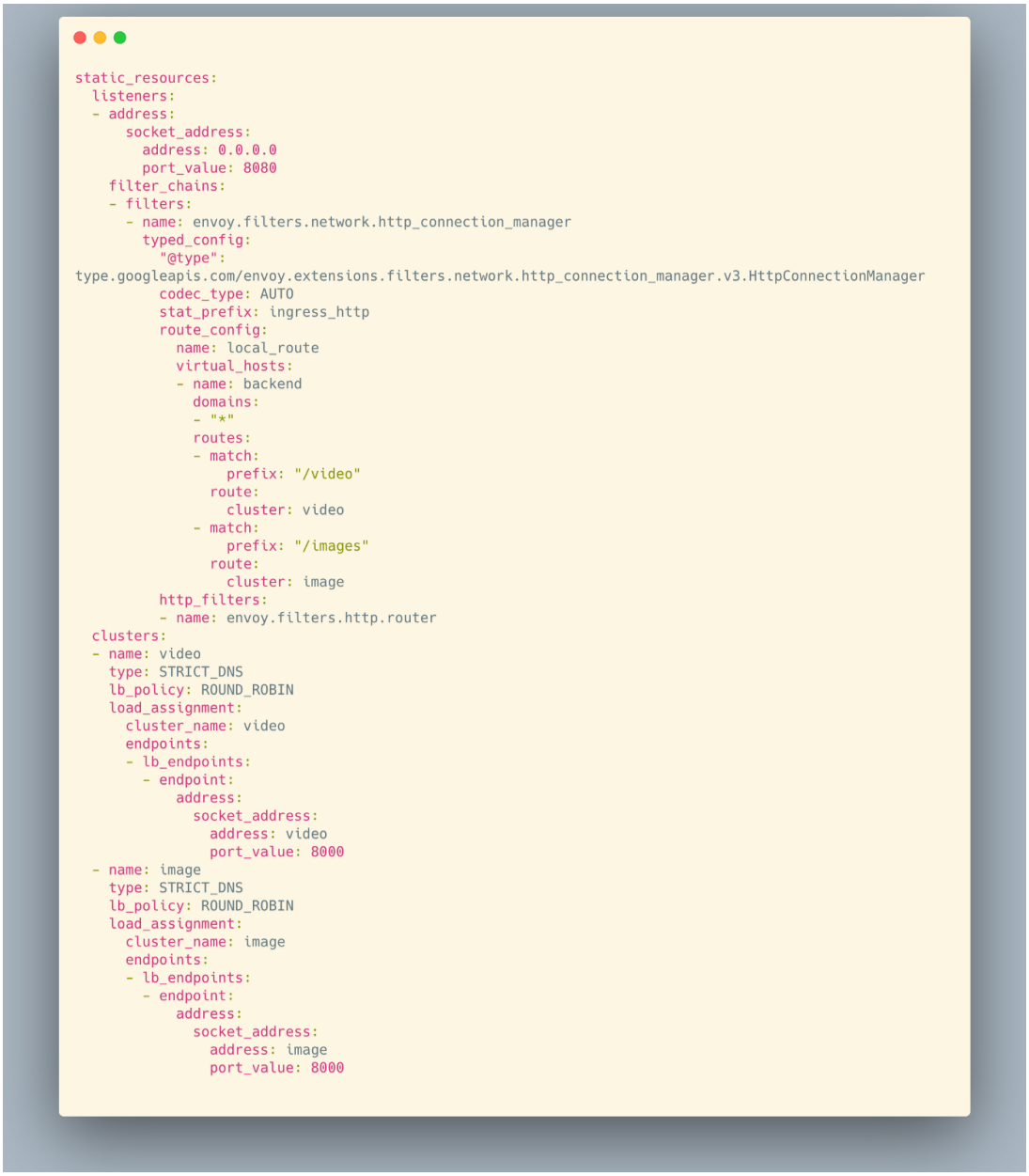
> *One common use case is load balancing traffic among services. In this example, an internal client can request video and image content by using the same base URL with the paths* `/video` *and* `/images`*.*
>
> *The internal HTTP(S) load balancer's URL map specifies that requests to path* `/video` *should be sent to the video backend service, while requests to path* `/images` *should be sent to the images backend service. In the following example, the video and images backend services are served by using Compute Engine VMs, but they can also be served by using GKE pods.*
>
> *When an internal client sends a request to the load balancer's internal IP address, the load balancer evaluates the request according to this logic and sends the request to the correct backend service.*



Envoy handle HTTP requests

Envoy's ListenerFilter intercepts the inbound traffic, determines the protocol, and then passes it to the http_connection_manager at Layer 4 for processing. Once the packets were decoded by `codec`, they will be passed to L7 HTTP filters. The `envoy.filters.http.router` must be the last one in the filter chain. It will locate the target cluster based on the URL path and route configuration, and then selects the target address according to the load balancing strategy.

In the previous sample use case, video and image backend services are deployed behind the load balancer Envoy, each with a sidecar Envoy. Here is my configuration for load balancer Envoy. It will pick available backend in round-robin order.



```
static_resources:
  listeners:
  - address:
      socket_address:
        address: 0.0.0.0
        port_value: 8080
    filter_chains:
    - filters:
      - name: envoy.filters.network.http_connection_manager
        typed_config:
          "@type":
type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpConnectionManager
          codec_type: AUTO
          stat_prefix: ingress_http
          route_config:
            name: local_route
            virtual_hosts:
            - name: backend
              domains:
              - "*"
              routes:
              - match:
                  prefix: "/video"
                route:
                  cluster: video
              - match:
                  prefix: "/images"
                route:
                  cluster: image
          http_filters:
          - name: envoy.filters.http.router
  clusters:
  - name: video
    type: STRICT_DNS
    lb_policy: ROUND_ROBIN
    load_assignment:
      cluster_name: video
      endpoints:
      - lb_endpoints:
        - endpoint:
            address:
              socket_address:
                address: video
                port_value: 8000
  - name: image
    type: STRICT_DNS
    lb_policy: ROUND_ROBIN
    load_assignment:
      cluster_name: image
      endpoints:
      - lb_endpoints:
        - endpoint:
            address:
              socket_address:
                address: image
                port_value: 8000
```

load balancer Envoy configuration

Summary

- Cloud load balancer will mainly do three jobs: endpoint discover, health check & load balancing.
- The load balancing solutions are basically two types: L4 load balancer and L7 load balancer.
- The Layer 4 (transport layer) load balancer keeps the same TCP connection between the client and the server. It includes all Layer 2 / 3 / 4 load balancing techniques.
- Google Cloud Internal TCP/UDP Load Balancer (L4 Load Balancer) is one of the virtualized network functions provided by Andromeda.
- The Layer 7 (application layer) load balancer is more powerful and servers as the foundation for service mesh traffic management, but it comes at a cost: efficiency.

- Google Cloud Internal HTTP(S) Load Balancer (L7 Load Balancer) is based on the open source Envoy proxy.

Proxy Kubernetes Microservices Cloud Google