

Optimized Load Balancing Architecture: Enhancing Network Performance and Reliability

(Practical Research Project)



California State University, Fullerton
CPSC 558: Advanced Computer Networks
Professor Hernan Manabat

Team Members:

Pratishtha Soni (CWID: 885587907)
Sai Siva Pavan Paruchuri (CWID: 885176172)
Kunal Chetan Doshi (CWID:886686773)

Context

1. Abstract	3
2. Introduction	4
2.1 Approaches	4
3. Design	5
3.1 Scenarios	7
4. Advanced Network Considerations	10
5. Key Load Balancing Algorithms	11
6. Implementation	12
7. Conclusion	16
8. References	16

1. Abstract

As modern enterprises digitally transform business operations, computer networks constitute the critical backbone synchronizing data and services. However, exploding internet adoption plus the rise of cloud platforms now impose extreme demands on corporate IT infrastructure.

Unprecedented quality, performance, and security expectations challenge conventional network architectures. Crippling outages from minor failures or malicious attacks cause unacceptable productivity and revenue losses. Yet continually expanding network capacity via costly hardware upgrades only offers temporary relief as utilization growth soon eclipses each capacity leap.

This research paper examines optimized network infrastructure strategies for escalating availability, scalability, and resilience imperatives. Intelligent load balancing mechanisms distribute intensive client workloads across diverse shared resources, increasing throughput while reducing response times. Streamlined failover systems also instantly reroute traffic from disrupted endpoints minimizing downtime.

Evaluating trade-offs around visibility, control, and debugging for different load balancer topologies allows the selecting of tailored solutions. Combined tactics further boost robustness through redundancy and dynamic capacity scaling. Such comprehensive approaches underpin guaranteed, performant network access critical for modern business productivity. They also strengthen confidentiality and data integrity by isolating vulnerabilities and malicious activity.

Adapting at the architecture level with innovations like software-defined networking coupled with workload-aware load balancing paves the way for responsive, secure networks that continue excelling as usage patterns evolve.

2. Introduction

Computer networks now interlink nearly all modern technological infrastructure, underpinning vital societal services from banking systems to media consumption. As billions of users and devices interconnect globally, these indispensable networks endure ever-rising demands for higher availability, greater reliability, and faster response times. However, existing network architectures are under massive capacity and performance burdens.

Radical innovations like the public internet, mobile platforms, and cloud computing subject networks to unprecedented complexity too. Supporting such a breadth of new use cases taxes legacy network hardware. As emerging 5G, artificial intelligence, and internet-of-things ecosystems generate increasingly diverse traffic patterns, existing network designs falter. Ever-looming cybersecurity threats also impose strict safeguard requirements in parallel.

Forward-thinking network transformations prove critical for managing these intensifying user expectations and technology shifts simultaneously. Software-defined architectures and virtualization paradigms promise improved flexibility compared to entrenched hardware-constrained models. Other network infrastructure advancements like load balancing offer immediate performance, scalability, and resilience gains.

Load balancing – distributing client requests across multiple servers – prevents individual server overloading, which risks detrimental latency spikes or total failures. This crucial capability not only enhances throughput and efficiency but also bolsters availability through continuous server health checks and prompt failovers. As networks brace for the next waves of unpredictability, innovating higher-level network logic like intelligent traffic routing will complement fundamental re-architecting efforts.

In this paper, we explored the vital network strategy of load balancing specifically from architectural perspectives of optimal performance, visibility, adaptability and access continuity that define industry-grade application delivery now and into emerging networked horizons.

Load Balancer Network Functions

Core networking functions provided by load balancers:

- Virtual IP DNS Mapping – Create a consistent virtual server entry point
- Request Distribution – Intelligently spread inbound packets across real servers
- Health Checks – Continuously validate server health status
- Dynamic Failover – Automatically shift users away from downed servers

These capabilities rely on and influence underlying network configuration.

2.1 Approaches

Several measures, including a redundant array of inexpensive disks (RAID), clustering, load balancing, redundant data and power lines, disk shadowing, software and data backups, co-location and off-site facilities, roll-back features, fail-over configurations, and service level agreements, can be put in place to ensure the availability of an information system and its resources.

Designing and deploying a **fault-tolerant system** rigorously is one way to increase availability. These systems have fail-over or load-balancing software, redundant peripherals, connections, and CPUs. In the event of a significant hardware or software malfunction, these systems may function at a respectable level thanks to their fail-safe features. The system can contain numerous components that can each do the same task thanks to redundancy and replication, allowing the other components to take over the services if a component fails. The drawback is that some parts might not be utilized while others are, resulting in unnecessary expenses. An instance of this would be a company that simultaneously contracts with two ISPs to lessen the impact of disruptions. If they choose the "redundant" or "backup link" strategy, they will use one link at a time until it fails, connecting the second line. This strategy had the benefit of being simple to execute, but because the resources that were purchased were not fully utilized, it was quite expensive.

The **load-balanced** mode is another technique that divides the workload among several computers or other resources via network connections. When used in network traffic management, this method enables traffic to be assigned to numerous routes that lead to the exact location, resulting in traffic being dispersed effectively over those routes. The system will be set up so that both ISPs supply their bandwidth to a standard intelligent device, such as a router or firewall with load-balancing capabilities, assuming the organization has subscribed to 20 Mbps of bandwidth from two different ISPs, each of which provides half of the total bandwidth. While the consumers are unaware of which of the two lines is carrying their data, the bandwidth will be allocated to all users from the shared pool. However, if one fails, the router or firewall will instantly and smoothly reroute all users to the working link. Apart from a transmission speed decrease resulting from the available bandwidth being reduced by half, all this ought to be transparent to the users.

3. Design

Positioned in front of your server, a load balancer acts as a "traffic police," distributing client requests among all servers. It only ensures that no single server handles excessive requests, which could lower the application's overall performance by efficiently dividing up the set of requested activities (cache queries, database write requests) among several servers. A load balancer may be a software process, a virtualized instance operating on specialized hardware, or a physical device. Imagine a scenario in which a client connects directly to a single server hosting an application without using load balancing. It will look like the image below.



Fig: System without load balancer (Load balancer - system design interview question 2023)

There are two issues with this design:

- **Single Point of Failure:** The entire program will be disrupted and unavailable to users for a certain amount of time if the server goes down or has some other problem. It will result in a negative user experience that service providers cannot tolerate.
- **Overloaded Servers:** The quantity of requests that a web server can process will be limited. Should the enterprise expand and the volume of requests rise, the server will experience overloading. We must spread the requests throughout the cluster of computers and add a few more servers to handle the growing volume of requests.

We can place a load balancer in front of the web servers to address the problem above and disperse the volume of requests. This will enable our services to process infinite requests by expanding the number of web servers within the network. We can distribute the request among several servers. The service will continue even if one of the servers goes down for whatever reason.

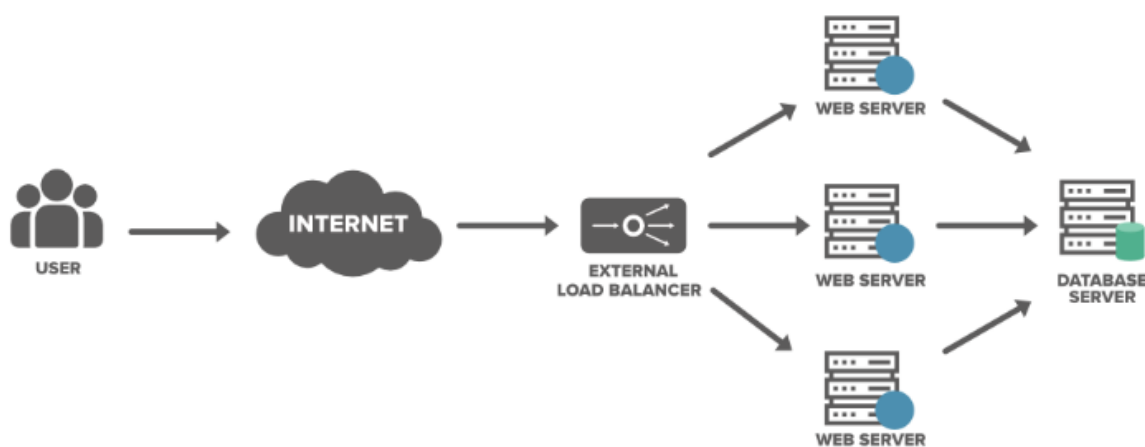


Fig: System with Load balancer (Load balancer - system design interview question 2023)

- Throughput increases, and server response time is reduced via load balancers.
- By directing requests solely to online servers, load balancers provide high availability and dependability.
- To keep track of the server's ability to process requests, load balancers continuously perform health checks.
- The number of servers that load balancers add or remove depends on the volume of requests or demand.

3.1 Scenarios

The concept of load balancing is quite impressive. Regardless of vendor, any load balancer can be configured easily if you understand the fundamentals and the three situations typically used in real-world deployments.

The basic load-balancer scenarios are:

- Two-arm
- One-Arm
- Direct Server Response (DSR)

3.1.1 Two-Arm Load-Balancer (Routed mode)

The most straightforward design has the load balancer operating "in-line" between two networks - usually with the client subnet on one side and the back-end server farm on the other. This separates external traffic from internal infrastructure for improved security, too, since all sessions must flow through the load balancer gatekeeper. Two-Arm can also be utilized in "bridge mode" or "transparent mode." The load-balancer would function similarly to a switch load-balancer by altering L2 frames, but otherwise, the scenario diagram would remain the same.

Routed mode scenarios are used far more frequently since they are clearer, more accessible to debug, and more straightforward. Since there are rarely any compelling reasons to use a two-arm situation in switched mode over a two-arm scenario in routed mode, must acknowledge that have not yet seen one implemented in a true production network.

A basic example of a routed two-arm solution is shown in the following image, which shows that a load balancer physically divides the users from the server; as a result, all traffic must pass through it.

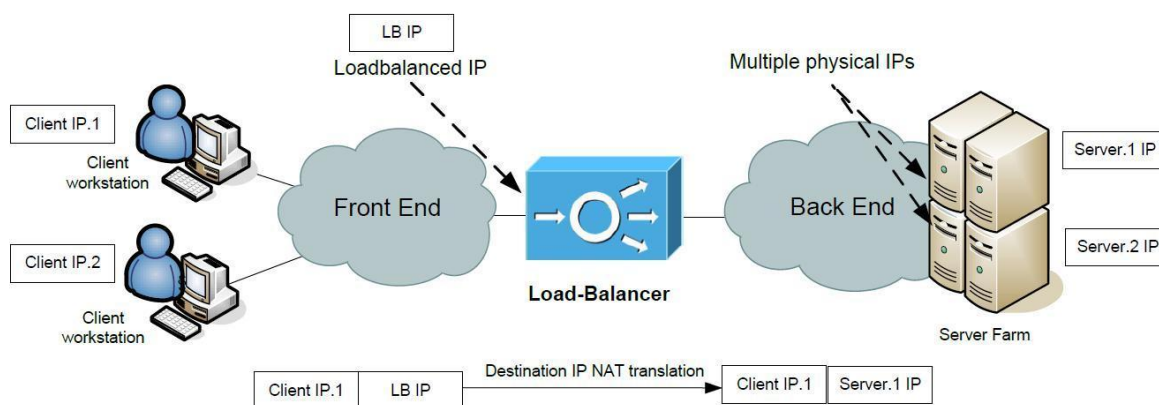


Fig: *Two-Arm Load-Balancer (Havrila, 2011)*

The load balancer also functions as a router connecting the "Front End" and "Back End" networks. To pass the packet to a server farm server, he can thus easily perform destination IP NAT on client requests that arrive at the load-balanced virtual IP. Throughout this procedure, the load-balancing algorithm selects the final physical server.

One key benefit of Two-Arm Load Balancer is the ability to easily monitor and analyze all traffic entering or leaving the server infrastructure since it is forced to flow via the centralized load balancer. This facilitates troubleshooting, performance tuning, and security governance use cases. Logical isolation also tends to simplify management and access control configurations.

However, the in-line model introduces a couple of Single Points of Failure (SPOF) - the load balancer itself and the interconnecting network link, which both represent capacity bottlenecks that could take down the entire application if disrupted. High availability load balancer pairs or clusters are common to address failover requirements in two-arm topologies.

3.1.2 One-Arm Load-Balancer

One-Arm refers to the situation where the load balancer is not physically "in line" with the traffic; rather, it must obstruct traffic flow in some way to regulate all client-server connections that are flowing both ways. The client workstations can be on the same LAN or behind the internet; it makes no difference to the load-balancing system how far apart they are. All servers are connected to the same L2 network as the Load-Balancer, which only uses one interface.

Traffic initialized by the client will reach the load balancer with the virtual load-balanced IP. The load-sharing algorithm will select a physical server, and the traffic having a destination IP that is NATed to the server's physical IP will be sent by the load balancer to that server via the same interface.

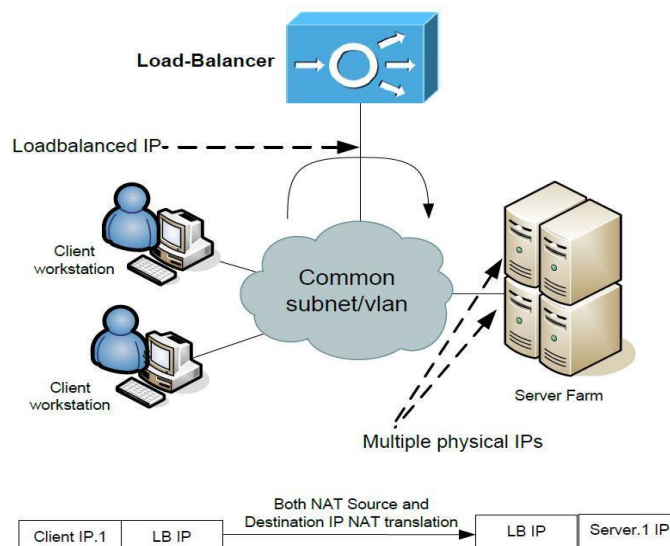


Fig: One-Arm Load-Balancer (Havrila, 2011)

To ensure that the server reply is sent through the load balancer rather than the client—who isn't expecting a reply straight from the physical server IP—the load balancer must additionally perform source IP NAT. Every request originates from the load balancer, according to the actual servers.

However, traffic flows become more convoluted because clients will naturally attempt communicating back directly with servers that are layer two adjacent to the LAN after the initial requests distributed by the load balancer itself. Return packets would bypass the load balancer, defeating its availability and performance optimizations.

This necessitates configuring one-arm load balancer deployments to use a technique called Direct Server Return (DSR), which relies on Network Address Translation (NAT) to prompt return traffic to flow back up the request chain via the load balancer again instead of trying to reach actual server IP addresses hidden behind virtual ones. Client connections are terminated at the load balancer, while only server-originated traffic uses the real IP scheme.

A downside is that without having an end-to-end picture of both request and response packets traversing the load balancer, it becomes much harder to monitor, analyze, and troubleshoot application traffic as easily as with a traditional two-arm model.

3.2.3. Direct Server Response (Also known as Direct Server Return)

The final common load-balancer situation is called "Direct Server Response," which requires adding a switch to the topology to comprehend. The Direct Server Return load balancer architecture makes backend servers directly responsible for returning traffic to clients after inbound distribution handled by the initial request broker. This

minimizes load balancer bottlenecks since response packets don't have to make the extra lateral hop through it on their way back to the requester. Especially for large downloads or media streams, a DSR model can leverage far more available server bandwidth than the constrained load balancer uplinks in legacy two-arm topologies.

As frames arrive on ports with source MACs, switches pick up information about MAC addresses. Additionally, consider a router that must know the load-balanced IP's MAC address on the final L3 hop.

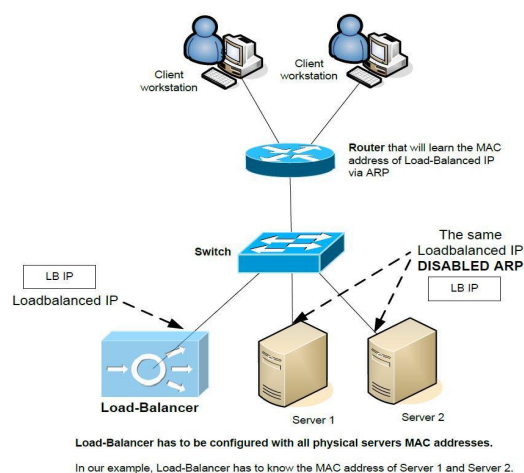


Fig: *Direct Server Response (Havrila, 2011)*

The only client-server traffic that the load balancer sees in this instance is the inbound portion and all client-IP traffic returns directly from physical servers. The primary benefits of this approach are that there is no network address translation (NAT), and the load balancer system only uses its throughput once, resulting in a reduced performance effect. On a physical server, turning off ARP is not a challenging process. Unfortunately, the Load-Balancer only sees one-way traffic throughout the whole L2 segment, which makes troubleshooting more challenging. You must also manually configure the Load-Balancer with all the server MAC addresses.

Another issue is servers must disable ARP to avoid leaking their real MAC addresses to clients, which would allow bypassing the load balancer on subsequent requests. Additional configurations like static host routes are also involved to steer return traffic correctly. Dedicated server switches might even be deployed solely interacting with load balancers and real hosts to simplify cabling and interfaces for traffic direction.

Hybrid Designs

The above represents the most common standardized load-balancing network blueprints. However, real-world deployments often combine elements of multiple models to deliver infrastructure tailored explicitly to application-specific traffic patterns and dependencies.

This hybrid approach allows for choosing the right load-balancing architecture for each application tier and use case. Standard templates meet most typical use situations, but the flexibility exists to customize further where needed.

As infrastructure scales up and gets partitioned out geographically, it also contracts different load-balancing deployment models depending on locality. For instance, traditional hardware appliances could operate regions housing user-facing web servers. However, cloud-based software load balancers integrate more smoothly with backend containers and functions distributed globally through microservices architecture.

4. Advanced Network Considerations

Additional factors like persistence requirements, SSL offloading needs, elastic scaling model and integration with modern container ingress controllers influence networking intricacies. Evaluating aspects such as:

- Effects of traffic symmetry on load balancing algorithms
- NAT configuration constraints
- Multicast and anycast protocols
- SDN integration compatibility

In cloud native infrastructure, the automation model also affects the network fabric via controllers and service discovery.

A well-architected load balancer mesh suits application delivery requirements while also improving resilience, analytics and iteration velocity.

5. Key Load Balancing Algorithms

The distribution algorithm each load balancer utilizes represents one of the most impactful design decisions because imbalance inefficiencies defeat the original goals. If one server gets overloaded while others lay idle, the fundamental capacity scaling motivation behind load balancing breaks down.

- **Round Robin**
 - This most basic strategy cycles requests equally amongst all available servers. It operates on the fairness premise that each registered host gets an equal number of connections. No external factors influence next-hop selection.
 - Round-robin strengths include simplicity and facilitation of session stickiness through stateful permutations. However, it ignores significant parameters like actual server loading and health, leading to ineffective request allocations.

- **Weighted Round Robin**

- An enhancement that assigns server weights based on capabilities to skew distribution towards more powerful machines able to handle bigger loads. It prevents imbalanced assignments by considering assigned capacity ratios.

- **Least Connections**

- This technique dynamically tracks active connections on backend hosts to calculate which option currently has the least open sessions. By picking the lightest loaded target, new requests get sent to servers with the most available capacity to process them.
- Application performance increases through lower congestion and buffering delays. But sudden traffic spikes could overwhelm nodes before adjustments kick in.

- **Least Time/Response**

- Rather than solely open connections, this approach favors servers empirically displaying the fastest response times as optimal candidates for future requests. Especially beneficial for latency-sensitive applications.
- However, it requires sufficient time for accurate measurements before identifying ideal candidates. Cold start and aged data can cause low-performance choices until stabilized.

- **Hashing Algorithms**

- Hashing uses a preset mathematical formula to allocate requests. Inputs like source IP, session cookies, or requested resource URLs get programmatically mapped to a particular server. This facilitates session affinity as clients consistently reach the same host.
- Downsides of hashing include difficulty balancing loads evenly and dependencies on inputs users can potentially manipulate to force allocations. Rehashing requirements after server additions/retirements also prove disruptive.

- **Agent-based**

- Load balancer software agents deployed on each backend host give local visibility into its current loading, capacity, and health status. This live

telemetry gets monitored by a central controller to promote real-time distribution optimization.

- Key metrics like CPU utilization, memory availability, disk I/O rates, and application-specific data like database connection usage guide intelligent allocation. However, the agents must be kept highly performant to prevent turning into a resource bottleneck themselves.

6. Implementation

While understanding the architectures and algorithms constitutes a major portion of design decisions, choosing the right hardware or software foundation for implementing load-balancing capabilities remains equally important.

The most traditional option involves proprietary hardware appliances from vendors like F5, Citrix, A10, Radware, etc. These specialized devices come pre-installed with dedicated software optimized for load balancing, acceleration, and security capabilities, leveraging custom ASICs and hardware processing. In our case, we will use Google Cloud to develop a custom load-balancing approach.

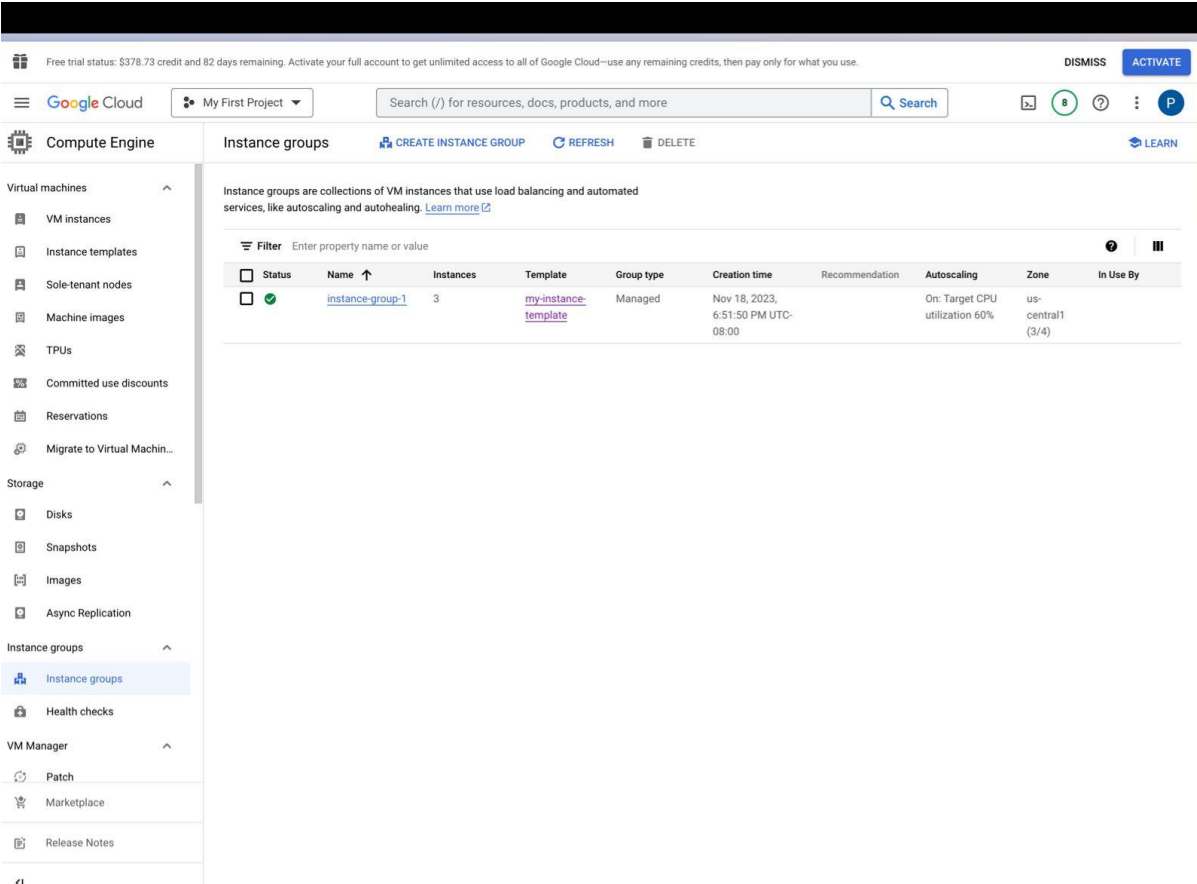
To implement load balancing, first create the account on Google Cloud and then deploy the application from local to Git Hub and then to Google Cloud.

- Go to “Compute Engine” from the dashboard and select “Instance Templates” under the “Images and Instances Templates” section. Added details like the template name, machine type, boot disk specifications, firewall configurations, and, optionally, a startup script.

The screenshot displays the Google Cloud Platform console interface. At the top, there's a navigation bar with the Google Cloud logo, a search bar, and a 'My First Project' dropdown. Below this, the 'Compute Engine' section is active, showing 'VM instances'. The 'INSTANCES' tab is selected, displaying a table of VM instances. The table has columns for Status, Name, Zone, Recommendations, In use by, Internal IP, External IP, and Connect. Three instances are listed, all with a status of 'Running'. Below the table, there are 'Related actions' such as 'Explore Backup and DR', 'View billing report', 'Monitor VMs', 'Explore VM logs', 'Set up firewall rules', 'Patch management', and 'Load balance between VMs'. At the bottom, there's a 'CLOUD SHELL' terminal window showing a welcome message and the current project ID 'empirical-envoy-405423'.

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
Running	instance-group-1-9g8r	us-central1-c		instance-gr...	10.128.0.9 (nic0)	34.41.212.148 (nic0)	SSH
Running	instance-group-1-gltj	us-central1-f		instance-gr...	10.128.0.10 (nic0)	34.70.226.223 (nic0)	SSH
Running	instance-group-1-n2ng	us-central1-b		instance-gr...	10.128.0.11 (nic0)	34.171.213.97 (nic0)	SSH

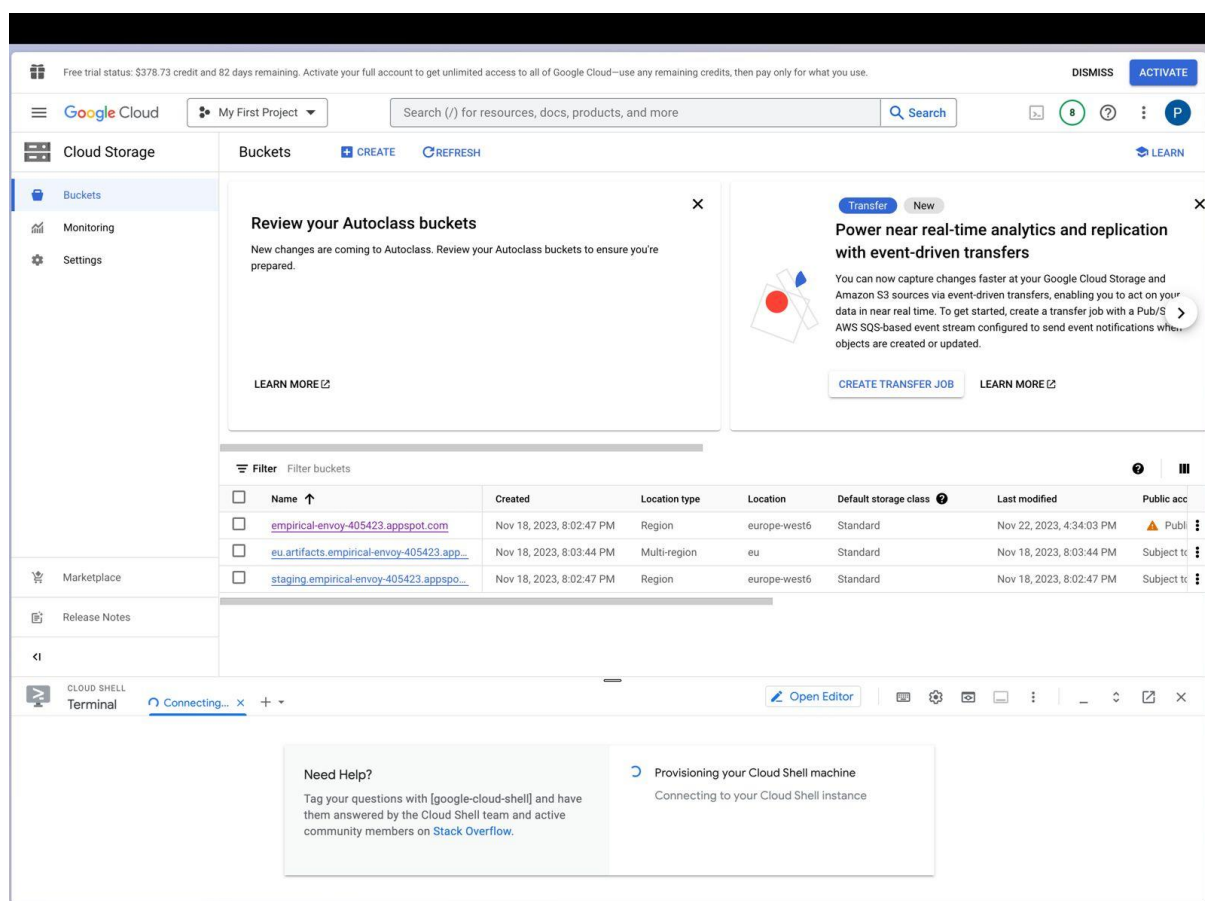
- Now select "Instance Groups." after navigating again to "Compute Engines". Once the Compute Engine is prepared, create an Instance Group by clicking the corresponding option. Provided the name and description, selected the zone, specified the region, generated an instance template, and proceeded by clicking the "Create" button.



The screenshot displays the Google Cloud console interface. At the top, there's a header with the Google Cloud logo, project name 'My First Project', a search bar, and user profile. The left sidebar contains a navigation menu with categories like 'Virtual machines', 'Storage', 'Instance groups', and 'VM Manager'. The 'Instance groups' category is currently selected, and its sub-item 'Instance groups' is highlighted. The main content area shows the 'Instance groups' page. It includes a description: 'Instance groups are collections of VM instances that use load balancing and automated services, like autoscaling and autohealing. [Learn more](#)'. Below this is a table with columns: Status, Name, Instances, Template, Group type, Creation time, Recommendation, Autoscaling, Zone, and In Use By. There is one entry in the table with the name 'instance-group-1', 3 instances, using the 'my-instance-template', created on Nov 18, 2023, with a recommendation of 'On: Target CPU utilization 60%', and located in the 'us-central1' zone.

Status	Name	Instances	Template	Group type	Creation time	Recommendation	Autoscaling	Zone	In Use By
<input checked="" type="checkbox"/>	instance-group-1	3	my-instance-template	Managed	Nov 18, 2023, 6:51:50 PM UTC-08:00		On: Target CPU utilization 60%	us-central1 (3/4)	

- Now, navigate to access the "Storage" section, and click "Create Bucket." Give a globally unique name, choose a default storage class, and select a location for the bucket. Optionally, configure advanced settings such as access control, versioning, and logging. Under the "Permissions" tab, set access permissions as needed, and we gave public access by adding “all users”.



- Now, Configure both the Backend and Frontend settings. Provide a name for the load balancer, choose the backend configuration and add the bucket created in the last step, input the required details, establish a health check, and proceed to create the backend service. The health check plays a vital role in verifying that the Compute Engine directs traffic exclusively to instances that are both in a healthy state and currently operational.
- Now navigate to Frontend configuration, input the name, and click on the box labeled "IP address" to reserve a new static IP address. Subsequently, select "Add Frontend IP and port," or we can use the default one also.
- Upon setting up the frontend in the load balancer configuration, a single IP address is obtained, on which the website operates. In the screenshot below, the website is functioning with the IP Address **34.120.116.133**.



7. Conclusion

To conclude this, we can say that a properly designed load balancer architecture provides scalability, resilience, and optimized distribution of workloads across backend systems. Considering the types and combination of hardware and software solutions, optimal load-balancing methods, high availability configurations, health checks, and comprehensive monitoring is key to building a robust, efficient and secure load balancing infrastructure.

8. References

- Havrila, P. (2011, December 2). *Basic load-balancer scenarios explained*. NetworkGeekStuff.<https://networkgeekstuff.com/networking/basic-load-balancer-scenarios-explained/>
- GeeksforGeeks. (2023, August 4). *Load balancer - system design interview question*.<https://www.geeksforgeeks.org/load-balancer-system-design-interview-question/>
- H. Jiang, A. Iyengar, E. Nahum, W. Segmuller, A. N. Tantawi and C. P. Wright, "Design, Implementation, and Performance of a Load Balancer for SIP Server Clusters," in *IEEE/ACM Transactions on Networking*, vol. 20, no. 4, pp. 1190-1202, Aug. 2012, doi: 10.1109/TNET.2012.2183612.
- Semong, T., Maupong, T., Anokye, S., Kehulakae, K., Dimakatso, S., Boipelo, G., & Sarefo, S. *Intelligent Load Balancing Techniques in Software Defined Networks: A Survey*. *Electronics*, 9(7), 1091. <https://doi.org/10.3390/electronics9071091>

- S. Ristov, M. Gusev, K. Cvetkov and G. Velkoski, "Implementation of a network based cloud load balancer," *2014 Federated Conference on Computer Science and Information Systems*, Warsaw, Poland, 2014, pp. 775-780, doi: 10.15439/2014F454.
- W. J. A. Silva, K. L. Dias and D. F. Hadj Sadok, "*A performance evaluation of Software Defined Networking load balancers implementations*," *2017 International Conference on Information Networking (ICOIN)*, Da Nang, Vietnam, 2017, pp. 132-137, doi: 10.1109/ICOIN.2017.7899491.
- abuonji, P., Rodrigues, A. J., & Raburu, G. (2018, October). *Load Balanced Network: Design, implementation and legal consideration ...* Load Balanced Network: Design, Implementation and Legal Consideration Issues.
https://www.researchgate.net/publication/328751909_Load_Balanced_Network_Design_Implementation_and_Legal_Consideration_Issues