# What is it about?

To prepare for large and changing user demand we decide to use a load balancer. It helps distributes the network traffic out to the web server infrastructure. As the strain on the website increases, new servers are added to a group that the load balancer sends traffic too. This solution can deal with changing demands from the users by being able to dynamically add webservers to the resource pool and send the network traffic to the new server.

# Context

To reduce the burden on a single web server by using redundancy to distribute the traffic between multiple servers. When one of the servers hit the limit the load balancer sends the traffic to the next open server in the group. The system should be able to dynamically open more servers to accommodate the growing traffic by sending traffic using an algorithm.

# DevOps

This gives DevOps the ground work for a flexible network that can satisfy the growing demand for performance and security. This allows for the system to scale up and down the number of servers needed for the current traffic. Load balancer solves the issue of reliability and availability through redundancy by increasing the amount of web servers for the user traffic to go.

# How can it be implemented?

This can be implemented by having the load balancer sit before the group of servers to direct the request to those servers that maximizes the capacity and prevents servers from hitting the limit. The new servers can be added by creating a new instance using a template that can install the required software to run the website. They also should use an external database so that they all share the same content. Then the server address is added to the resource group list located on the frontend or load balancer instance.

# Challenges

There are many challenges load balancing faces such as:

* Geo-location of the nodes – The locations of each server can affect the response and performance of each request sent to it.
* Static vs dynamic algorithms – the static algorithms work on old system information but if the server fails in the resource group then the algorithm does not work. The dynamic algorithms keeps up to date of the system by dynamically adding and keeping track of the system state.

# What/how have we implemented

We implemented a load balancer with multiple render servers. So our Web-based System is a big system that could afford a huge amount of requests.

We used NginX to create our load balancer, which is a good program to handling high concurrent requests. We used NginX to be as a reverse proxy, forwarding users’ requests to different render servers, the render servers would generate HTML page and return them to users’ browsers. So we need a shared database to make sure that different render server would return same content to users no matter which render server rendered their request.

We used Google Cloud SQL for the centralized Database system, which has better session persistence than just install MySQL on the backend servers. This way the content is the same on every backend.

We edited the PHP code to show that which server rendered the page you opened to tell you which server rendered your request, what’s more, it could prove our load balancer works.