Assignment 02 - Report

Title: Introducing a Load Balancer in a Simple Cloud

Benchmark

We will carry out the following experiment to test out the performance of our load balancer.

Our cloud architecture support multiple pods, and there can be multiple nodes running inside of each pod. Each node in the same pod should run the same service, such that when traffic arrives at a pod, it can be load balanced acorss all the nodes inside of the pod.

For demo purpose, we will run a simple express server inside of each node. The server has an API route that takes in 2 query parameters, `n` and `k`. Upon request, the server will generate `n` random strings of length `k` and the job is to sort the string.

As you can see, with this experiment setup, we can easily adjust the `n` and `k` parameter to simulate different workload intensity.

The variables in this experiment will be the **number of nodes online per pod** and the **load balancing algorithm used** to balance the traffic.

For simplicity purpose, we will assume we are always generating 500 random strings with length 200. And we will send 1000 requests in total for each experiment.

Results

3 Nodes

Algorithm	Req/Sec	Latency/Req
Round Robin	88.21	0.38
Least Connected	87.86	0.40
Random	80.47	0.42

5 Nodes

Node Count	Req/Sec	Latency/Req
Round Robin	94.28	0.38
Least Connected	97.09	0.36
Random	91.18	0.36

8 Nodes

Node Count	Req/Sec	Latency/Req
Round Robin	97.78	0.36
Least Connected	92.84	0.40
Random	91.02	0.37

Conclusion

Overall, as we can see that for all 3 different algorithms, the latency is pretty much around the same. Why is this the case? Most likely because with the amount of traffic generated by the blaster, it was not enough to congest the network, or to cause any bottleneck in terms of bandwidth. There is another way to increase the latency, which is to overload the server such that it consumes lots of CPU and memory. Though this is a great idea, it is not trivial to execute since as the container gets overloaded, it will get OOMkilled by Docker. Therefore, it is very hard to get it exactly right at the workload that can cause the server to slow but also won't get killed by Docker.

Besides the latency, we have also measure the average amount of requests servered per seconds. As we can see, both round robin and least connected algorithm performs better than a random algorithm. This makes a lot of sense because by distributing the workload in a more even manner, it will ensure that we get the most performance out of all the nodes, instead of having various hot spots which are not ideal for a cloud computing system.