Website Load Balancing Solution

by

Nick Mueller

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

Nick Mueller — Date 4/20/15

Mark Stockman, Faculty Advisor — Date 4/20/15

University of Cincinnati
College of
Education, Criminal Justice, and Human Services

April 2015

Table of Contents

Abstract

A load balancing solution enhances a network's efficiency and redundancy by balancing traffic across a network. My place of employment, whose name will remain anonymous, was in dire need of such a solution. Website uptime was being negatively affected by bad code causing memory leakage on the web servers. We took multiple website outages a week which resulted in poor customer experience, threatening business satisfaction and loyalty. This was not acceptable. My proposed plan to resolve this issue involved a virtual Kemp load balancer to alleviate stress off from the web servers. This allowed the servers to perform services properly until the code could be reviewed and patched. Aside from performance, this solution would provide us the ability to create web farms resulting in a more redundant network.

Introduction

Load balancers are necessary in any network. They specialize in making sure that traffic is balanced across a network allowing the infrastructure to be able to evenly and reliably handle network traffic load. By distributing network traffic amongst multiple web servers, it allows each server to be ran at optimal performance during high traffic periods. Another reason to use a load balancer is often overlooked. A load balancer is also very beneficial in the fact that they minimize points of failure and allow for redundancy in the instance a server in a web farm becomes unresponsive. If three servers are in a web farm and one of them fails, the remaining two can take on the workload of the downed server until it can be fixed. If not using a load balancer, the websites would be down until that one server could be brought back online. When dealing with a business where a majority of its business functions are done through its websites, it is critical that they are online at all times.

Problem

At my place of employment, one web server was being used for our primary website which is what handled most of our business functions facing the customer. Memory leakage due to bad code caused the webserver to lock up on a daily basis when the server hosting the websites with the bad code became starved of resources. Memory leakage is caused when working processes use memory from a server and when the process is finished, it doesn't give back all the memory it originally claimed from the server. Over time, the amount of available memory on the server becomes less and less until it reaches a critical point where there is not enough memory available for the web servers to function properly. When this occurred, our websites crashed and went

offline until the application pool could be recycled. We were experiencing this issue between 12am to 2am every night. We created a job to recycle the application pool at 12am every night, but this was not a permanent fix. The only real fix was to patch the poor code, but this automatic job at least kept the websites from going offline until an engineer could recycle the application pool manually. Whenever the application pool was recycled, anyone that was using the websites would lose their session. If a potential customer was filling out an application to become a customer and the app pool was recycled, that progress was lost.

Research

When I began the process of researching potential fixes that could be put in place by the networking side of the company, I proposed the use of load balancers. Depending on how many web servers we would put behind the load balancers, it'd increase the amount of time it would take before the memory leakage would reach a critical point. I researched F5 and then later Kemp. I really pushed hard for an F5 solution for the additional add-on features that F5 provides such as a firewall and IDS/IPS services, but the price was out of our budget. I was then referred to Kemp by a coworker and upon my research discovered that a Kemp solution would fulfill our needs. It provided the main features we would need such as persistence, round robin least connected routing, layer 4 routing, and SSL acceleration.

Budget

I proposed for a $40,000 budget for two physical F5 load balancers, but was approved for $5,000. The cost of a Kemp VLM-2000 virtual appliance is $4,900. Since I implemented the

load balancer, the cost of implementation was included in my salary, but hourly was about $20 per hour. If a contractor was doing the work, it would have cost an estimated $200 per hour. Configuration and implementation consisted of 8 man hours. For a large company, it may not be hard to look past $1,700, but for us it was out of our budget.

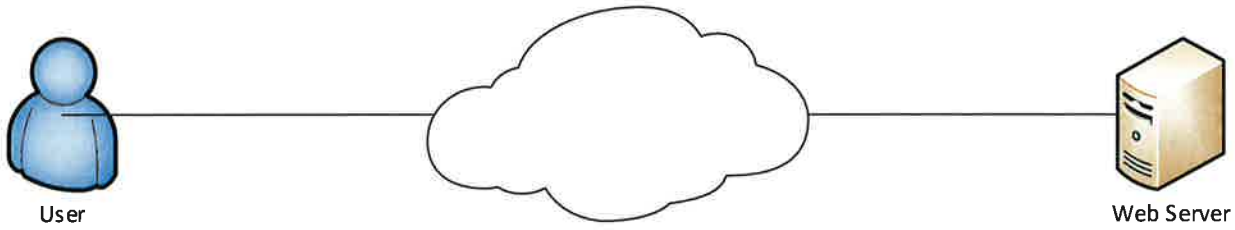
Cost of Implementation		
	Internal	**Contractor**
Kemp VLM	$4,900	$4,900
Man Hours	$20 per hour (est. 8)	$200 per hour (est. 8)*
Total:	$5,060	$6,600

*Estimate given by past contracting work

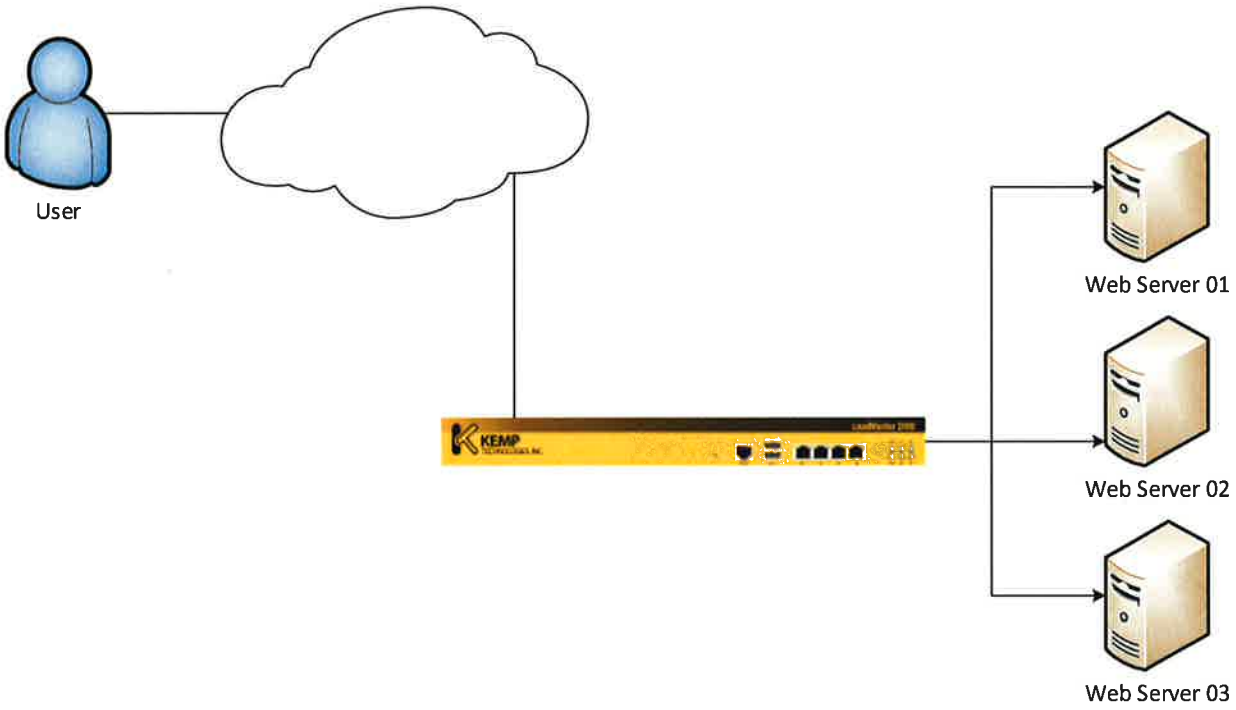
Solution

My proposed solution was to position a virtual load balancer in front of the web servers and to create a web server farm. Even though this is not a "true" fix for the memory leakage, creating two more identical web servers, for a total of three, would at least increase our outage duration from daily to a few days. Even though no outage is ideal, we needed to increase the duration between outages until our developers could fix the code. This solution also gave use the website redundancy which had been needed. The reason we decided to implement a virtual appliance was due to the fact that we were a 100% virtual shop, but also due to the fact that given the budget, we could only purchase one appliance. By purchasing a virtual appliance, we were able to create a template of our configuration and if in an instance we lost the virtual load balancer we could bring up another appliance and load the template onto it in a matter of seconds. It was, as we called it, "a poor man's disaster recovery".

Current Setup:

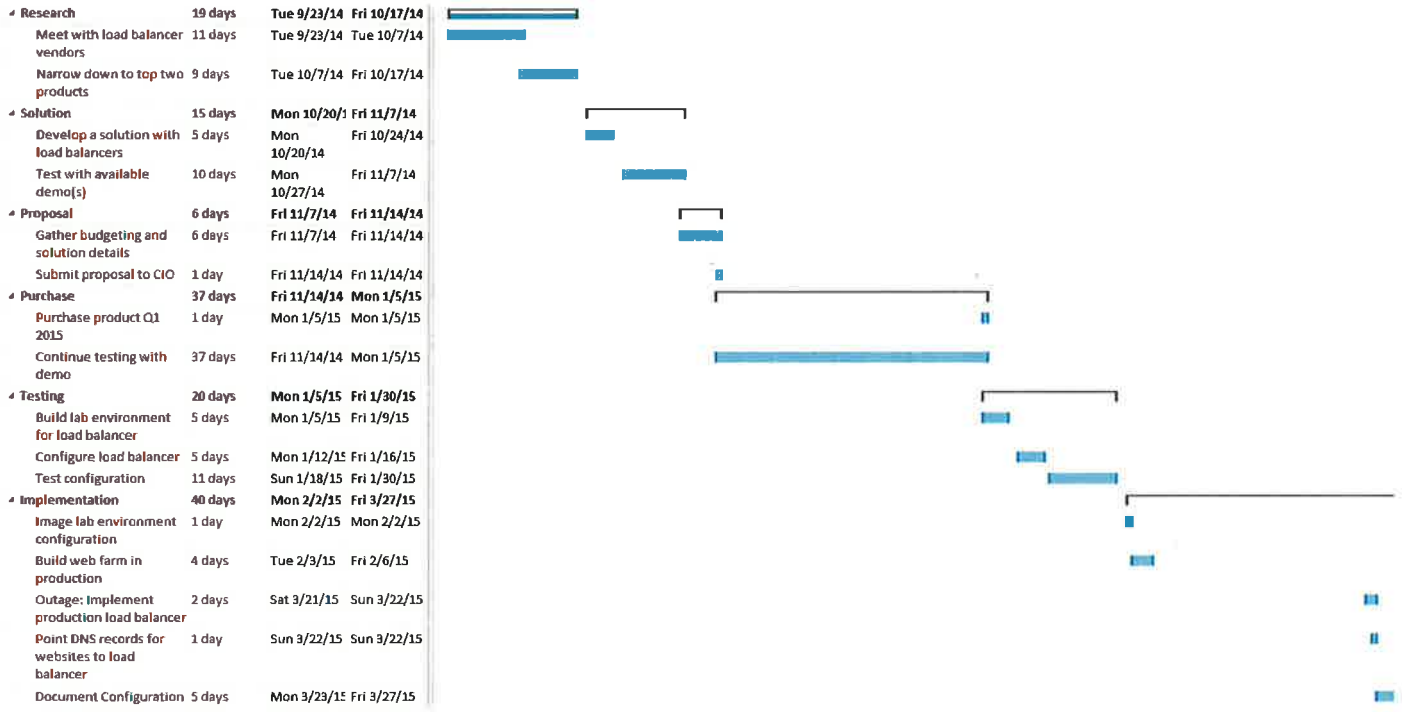


Proposed Solution:



Schedule

The following schedule shows the estimated timeline for the project.



Research	19 days	Tue 9/23/14	Fri 10/17/14
Meet with load balancer vendors	11 days	Tue 9/23/14	Tue 10/7/14
Narrow down to top two products	9 days	Tue 10/7/14	Fri 10/17/14
Solution	15 days	Mon 10/20/1	Fri 11/7/14
Develop a solution with load balancers	5 days	Mon 10/20/14	Fri 10/24/14
Test with available demo(s)	10 days	Mon 10/27/14	Fri 11/7/14
Proposal	6 days	Fri 11/7/14	Fri 11/14/14
Gather budgeting and solution details	6 days	Fri 11/7/14	Fri 11/14/14
Submit proposal to CIO	1 day	Fri 11/14/14	Fri 11/14/14
Purchase	37 days	Fri 11/14/14	Mon 1/5/15
Purchase product Q1 2015	1 day	Mon 1/5/15	Mon 1/5/15
Continue testing with demo	37 days	Fri 11/14/14	Mon 1/5/15
Testing	20 days	Mon 1/5/15	Fri 1/30/15
Build lab environment for load balancer	5 days	Mon 1/5/15	Fri 1/9/15
Configure load balancer	5 days	Mon 1/12/15	Fri 1/16/15
Test configuration	11 days	Sun 1/18/15	Fri 1/30/15
Implementation	40 days	Mon 2/2/15	Fri 3/27/15
Image lab environment configuration	1 day	Mon 2/2/15	Mon 2/2/15
Build web farm in production	4 days	Tue 2/3/15	Fri 2/6/15
Outage: Implement production load balancer	2 days	Sat 3/21/15	Sun 3/22/15
Point DNS records for websites to load balancer	1 day	Sun 3/22/15	Sun 3/22/15
Document Configuration	5 days	Mon 3/23/15	Fri 3/27/15

Testing

In order to properly test the load balancer, I created a lab environment which consisted of the virtual load balancer and three web servers with one website each with the text "IIS WEB SERVER 01", "IIS WEB SERVER 02", and "IIS WEB SERVER 03" respectively. This way, throughout my testing, I could tell what server I was hitting given a certain configuration to make sure the load balancer was working correctly. I first had to input the web servers into the load balancer so that it'd know where to route traffic. I then had to give the appliance a virtual IP so that I could point our external DNS to it. Some of the key components I configured were persistence, layer 4 routing, round robin least connect, and SSL acceleration.

Persistence is how the load balancer will handle connections from a same IP. It is advisable to have this enabled due to the fact that if an end user is working on a web server and has information stored on the back end, if that user disconnects and reconnects, you don't want that user going to a different server where that data doesn't exist.

Layer 4 routing defines the protocol that the load balancer will use to route incoming traffic. Layer 4 routing looks at a packet and hands it off to a server based on the configurations you define in the load balancer. In this case, round robin least connect was used. When a request comes in, it will be routed to the server that is being used the least at that period in time. If web servers have the same amount of connections it will route the requests sequentially. Layer 7 though is much more complex and since it wasn't used it won't be defined in detail, but layer 7 routing is how the load balancer routes packets based on the requested application.

SSL acceleration is when the load balancer uses it's resource to handle the SSL encryption rather than the web servers using processing power to process encrypted traffic. The load balancer will unencrypt the incoming secure connection and hand the HTTPS request to the web servers as HTTP.

Implementation & Results

After the load balancer's settings were tested in a lab environment, a template was created of the working configuration. Two additional web servers were brought up in our production environment mirroring the original web server and the load balancer appliance was created as well. The load balancer was placed behind our production firewall and patched to the most updated firmware for security. During a scheduled outage the template from the lab load balancer was blown onto the new load balancer in production. The virtual IP was defined for the

load balancer and the external DNS was pointed to the new production IPs for our websites. The IPs to the lab web servers were then changed to the new web servers in the production environment. Once we verified that the load balancer was talking to the web farm, we begun testing for the remainder of the outage to ensure all processes were functioning properly.
The results were as expected. We were seeing slightly more than 600 concurrent connections throughout the day on our single web server and after the implementation each server was now working with roughly 200 concurrent connections. The poor code from before had since been fixed and whenever there have been issues with one of the three new web servers we have been able to take the server offline without affecting daily business.

Conclusion

A load balancer in my opinion is a necessity in any network. My company, at the rate we have been growing, could not afford to take any more website outages due to any reason. The results have been great and after seeing the implementation, we have received approval for upgrading our load balancing environment Q1 of next year. We will be able to implement redundant load balancers so in the instance one went offline the other would take over immediately.

References

Kemp. http://kemptechnologies.com/ .Website. 2014