Case Study

Cloud Design & Migration

Dated: Feb-12-2024

Executive Summary

The customer, a well-known E-Commerce B2B information portal, was facing performance problems for their web portal. Additionally, their current architecture was based on the dedicated servers with fixed resources.

When they approached us for a solution, we used our proven mechanism of:

1. Getting the requirements. We call this step 'Customer's goal list'
2. Review of the current system. This step is called 'Establish the baseline'.
3. Design and implement the solution. Another name for this step is 'Implement, test, close'

We created an auto-scaling solution on the cloud that cost half of what the client was paying before. It also allowed them to handle much more traffic optimally. Following table shows a quick comparison of different metics between before and after:

Metric	Before Implementation	After Implementation
Cost per month	$4,800 + Tax	$2,370 + Tax
Load time (static page)	7 seconds	2.5 seconds
Load time (DB-heavy page)	23 seconds	4 seconds
Max visitors concurrent users	50	400

Problem Statement

Our customer is a B2B Ecommerce information portal specializing in robotics and automated systems for assorted verticals. Company was established in 2020 and is headquartered in the United States of America.

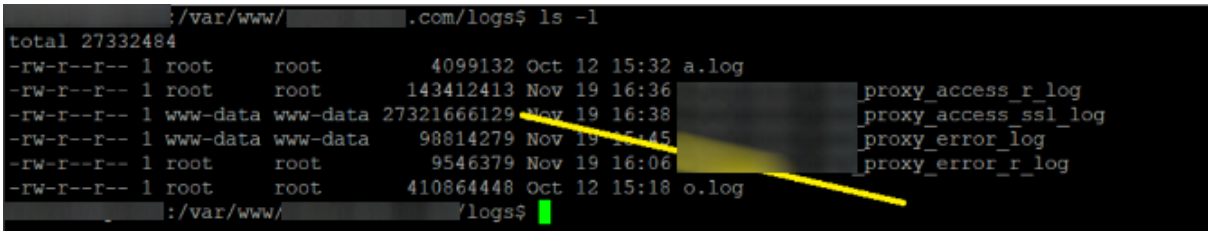
Their information portal was based on Drupal and deployed on an architecture composed of dedicated servers. However, during the periods of high traffic, the portal would be sluggish and frequently would result in 50x errors (meaning the requests were not processed completely).

These machines were also expensive to run and adding additional machines would result in considerable OPEX (operating expenses).

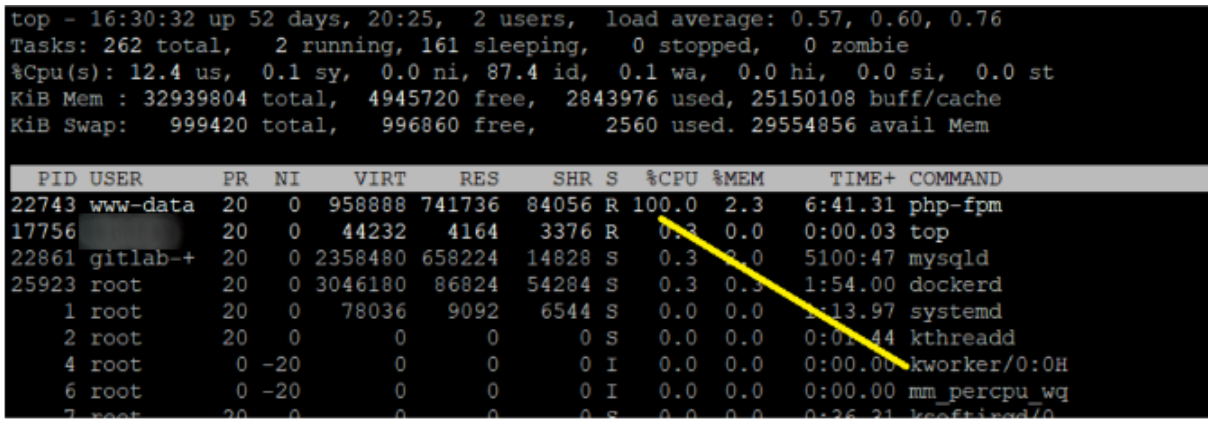
Key points can be summarized as follows:

1. Drupal is a database heavy technology and the existing portal based on an older version of Drupal
2. Static content was not served using a CDN

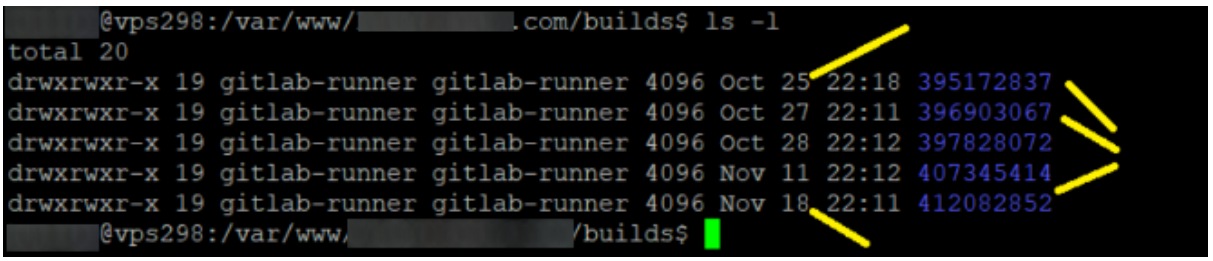
3. Caching was not employed
4. Architecture based on the dedicated machines were not capable of auto-scaling*
5. The OS of the Dedicated machines was old (at least by one major release version). Older version of PHP and libraries were also found
6. Static page load time was around 7 seconds
7. Heaviest web application page had a loading time of 23 seconds
8. Cost per month was over $4,800 + tax
9. Static CSS + JS files were not optimized (over 20 includes)
10. Images were not optimized and sprites were not used (development related optimizations).
11. CI/CD Deployment process was not optimized. Deployment to production would result in a 20-30 minute outage.
12. MySQL Memory consumption is 2% and remains constant (looks like DB optimization is missing). It means DB would chew up CPU by running queries instead of serving content out of its cache.
13. Production web server and database server were coupled on the same machine. This would result in RAM starvation as load increases
14. Log rotation was off and this resulted in ridiculously large logs.



15. PHP configuration for the FPM component was not optimized. It was taking too much of the CPU power.



16. Old builds were not removed and hence took up a lot of space.



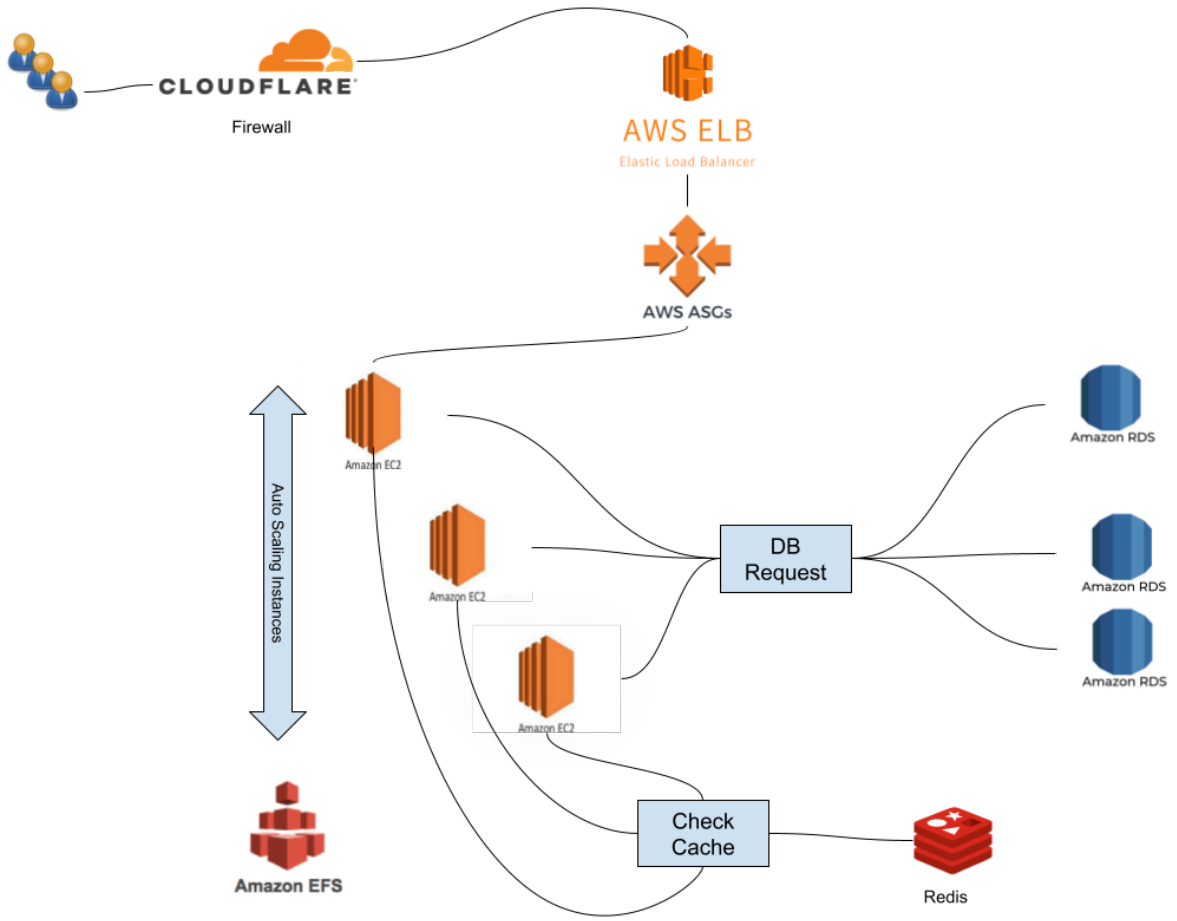
Our Solution

In order to address issues listed above, we devised a comprehensive plan for the creation of new architecture and migration of the existing website, email and media. During the design process, we considered following key items:

1. Robustness
2. Scalability
3. Security
4. Dynamic scaling
5. Cost effectiveness
6. Disaster Recovery
7. Using managed services
8. Comprehensive alerting mechanism
9. Agile Project management approach

High Level Architecture

The following diagram shows a high level architecture for our solution.



Key components of New Architecture

CloudFlare: Firewall / DDOS Protection

We opted to use Cloudflare as a Firewall / DDOS protection. Additionally, we used Cloudflare also to manage our DNS settings. Key factor involving this selection was the cost of Cloudflare as opposed to AWS WAF. SSL certificate is also provided by Cloudflare.

AWS Elastic Load Balancer (ELB)

This component accepts the incoming request from Cloudflare and forwards it to the auto-scaling group. This is a managed component. It hides our running VM from the outside world. This allows us to provide segregation from the outside world making it further secure.

AWS Auto-scaling Group (ASG)

This component manages the number of VMs (called EC2 in AWS lingo) depending on the heuristics defined. As traffic increases, ASG will spool up more instances to meet the additional demand. Similarly as load decreases additional instances will shut down automatically. At a minimum two instances will always be running.

AWS Elastic Compute (EC2)

This component is the VM machine that runs Software code. This component will host a copy of the Drupal application. A minimum of 2 such components will always be running. We decided to go with Ubuntu 20.04 OS as it had most compatibility with the Drupal version

AWS Relational Database (RDS)

This component is the managed Database Service. We used RDS running MySQL. We used Master-Slave architecture. All write operations would use Master and Read read operations will be served by the Slaves. This helps in increasing the concurrent number of users.

AWS Elastic File System (EFS)

This component is used for providing a uniform disk space for reading and writing media files. This is a managed service and offers automatic scalability and easy backup configuration.

Redis

This component is used for caching frequently used information. Key purpose is to reduce load on Database. Redis is an open source tool mostly used for this purpose. We used AWS's Elastic Cache with Redis engine for this purpose.

CI/CD Pipelines

We used AWS Pipelines that pulls updated code from Github repository and creates the deployment package and pushes the code to EC2 instances.

Improvements Observed

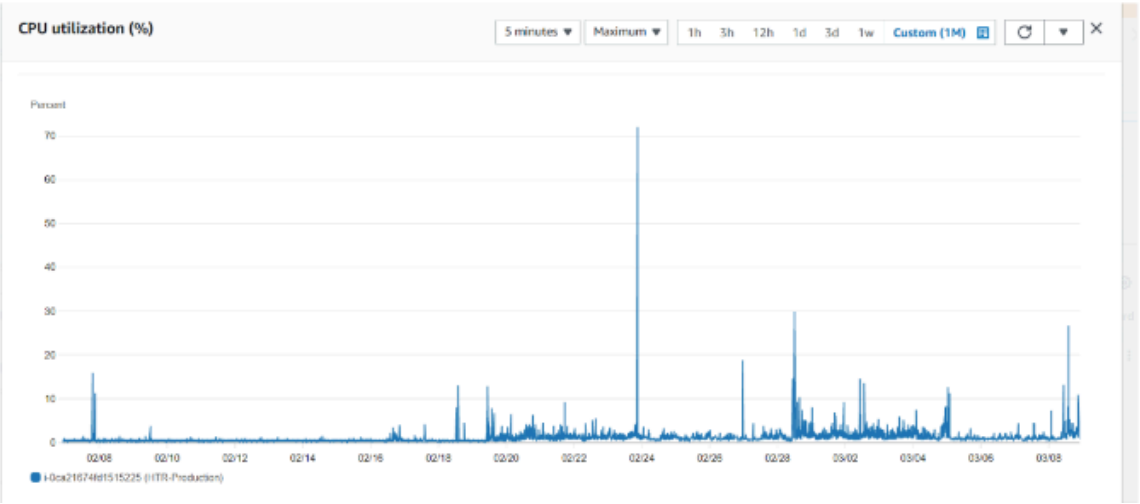
After migration to the new architecture, we observed various metrics under various load conditions and noticed that there was barely any load.

Memory Utilization

Following graph shows memory consumption of the running instance(s).

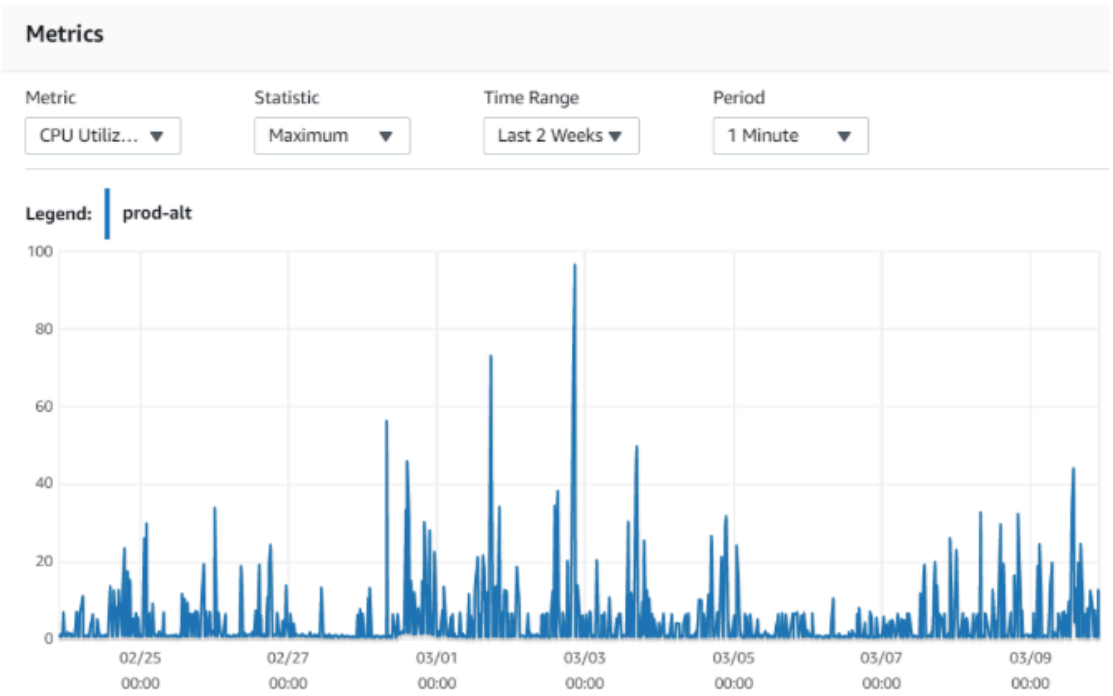
As shown in the above graph, memory consumption barely goes over 50%

CPU Utilization



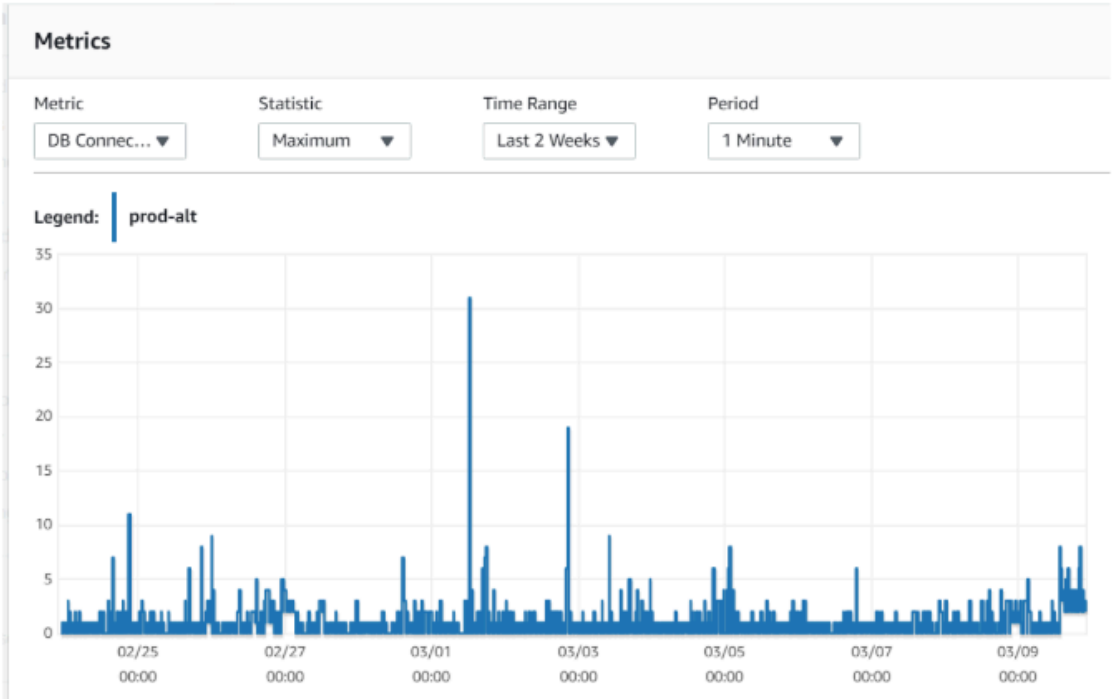
The above graph shows the CPU utilization over the period of 10 days. It is evident, mostly utilization remains less than 50%.

Database CPU Utilization



Above graph shows the CPU utilization of the RDS database instance. Average utilization remains less than 40%

Database RDS Connections



Above graph shows the number of RDS connections. Low number of connections shows the effectiveness of cache, as most of the requests were served by cache.

Conclusion

This case study illustrates how the FAMR0-LLC team approached the client's project. We thoroughly reviewed, discussed and experienced the problem(s) faced by the client. We interviewed their engineers, content writers and top management and compiled their feedback.

We used collected information to create a solution that addresses customer's current pain areas. After implementation we thoroughly tested the solution beyond the current load predictions to ensure that the solution is future ready.

Our solution helped our client to save up to **50.1%** in infrastructure costs per month. Loading time for static page went down by **64.3%**. Loading time for database heavy page went down by **82.6%**

Need Server / Cloud Optimization or Management

If you are experiencing similar issues or looking for a team to manage your infrastructure - please don't hesitate to contact us.

You can:

Phone: +971-505-208-240

Email: farhan@famro-llc.com

LinkedIn: https://www.linkedin.com/company/famro-llc/about