ELEC 6861: HIGHER LAYER TELECOMMUNICATION PROTOCOL

FINAL PROJECT REPORT

Instructor:
Prof. Roch H. Glitho

Teaching Assistant:
Yassine Jebbar

April 7, 2020

Jad Saad	**40084673**
Arihant Jain	**40067961**
Arqam Babar	**40081897**
Nashid Sultana	**40068634**
Neerav Bhatia	**40090577**

CONTENTS

Content Delivery Network

Jad Saad, Arihant Jain, Arqam Nawaz Babar, Nashid Sultana, Neerav Bhatia

***Abstract*— This report includes the discussion and the details of an implemented Content-Delivery Network. Using Amazon Web Services, 5 servers have been setup. An origin server which hosts the front-end part i.e. the webapp that allows streaming of videos to end users which was implemented using Wordpress, and 4 edge servers have been setup. 2 edge servers are situated in North America and the other 2 in Europe. A global load balancer based on latency rule was created to route traffic to the two regions and in each region an AWS Elastic Load Balancer was setup to balance the streaming requests between the two edge servers.**

***Keywords*— CDN, AWS, HTTP, GLB, ELB, NGINX, Cronjob, WordPress, latency, DNS, Servers**

I. Introduction

The demand for streaming of high quality content is ever in increase. It is crucial for platforms that offer this content to its end-users and depends on its availability that the content is easily, quickly and globally accessible. Consequently, addressing this need has become a business solution to third parties that offer a Content Delivery Network to streaming services. Such networks would offer the cache of the media content in multiple servers, spread to meet the demand for end-users in different parts of the world. The faster, the bigger and the more secure these networks can be, the more they are popular to streaming services.

Moreover, the CDN has to support the streaming protocols of the clients (the streaming platforms). Several protocols have been developed. Real-Time Messaging Protocol (RTMP/Flash) was developed by Adobe. It worked on top of Transport Control Protocol was popular for live streaming. However it has recently fallen out of favor due new protocols offering crucial features like adaptive bit-rate [1].

Apple developed their own protocol Apple HTTP Live Streaming which supports adaptive bit-rate streaming and was eventually supported by the majority of devices,browsers and operating systems. It is built using the HTTP protocol.

In an effort to provide an open-source option the Moving Pictures Expert Group created Dynamic Adaptive Streaming over HTTP (DASH) dubbed MPEG-DASH. Which also support adaptive bit-rate streaming and achieved similar latency figures to Apple's HLS [1].

The main goal of the project is to develop a CDN by which a user can access the video from the website having multiple replica servers. In our project, we stream using the HTTP protocol. We have implemented the streaming using an open source streaming software NGINX. The front end is designed in WordPress which offers the open source content management. With WordPress, managing everything from a single computer is often easy and the website capabilities can be enhanced or decreased based on the number of users. A website developed in WordPress offers the plugins and themes with specific features as per users' requirement and demands.

* *Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada*

II. Method & Implementation

In our architecture 2, we have split our servers into functions and regions. We have created our Origin Server in AWS's Canada-Central region and we hosted our front-end WordPress website on it. We have assigned the DNS name joanatech.com to it using AWS Route 53. Any request to that DNS name will resolve to our origin server which will serve the user interface website where users can browse the different video contents and stream on demand. For video streaming purposes we have built our CDN using 4 edge-servers located in AWS Ohio region and AWS Frankfurt (Germany) region.

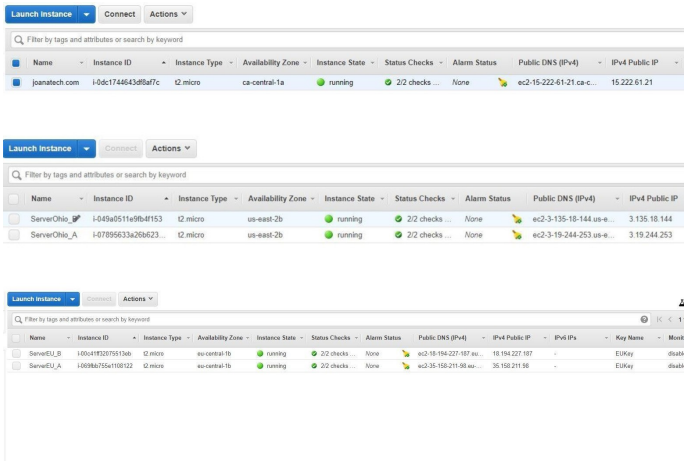

Fig. 1: Origin and Edge Servers

The origin server and four edge servers 1 were implemented using Amazon Web Services (AWS). The servers were first initiated and nginx 1.13.0 with Video on Demand (VoD) module was installed from open source website nginx.net to stream with HTTP protocol. It will be further deployed to all the servers. The DNS name of the origin server is "joanatech.com". The following figures show the origin and the edge servers running in North America (NA) and Europe (EU).

A. Streaming

Nginx is an open source media streaming software. It is the server which uses IP desktops, laptops and tablets,

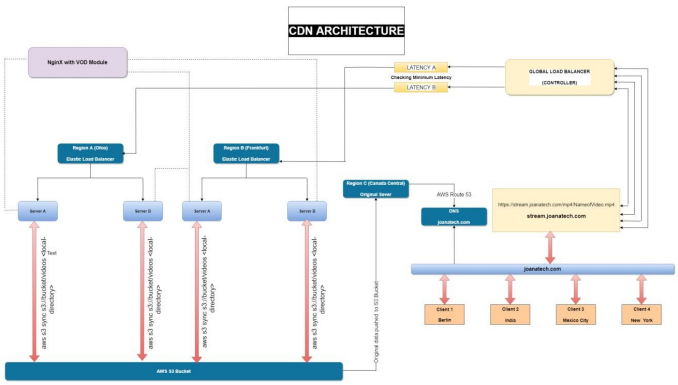

Fig. 2: CDN Archeticture

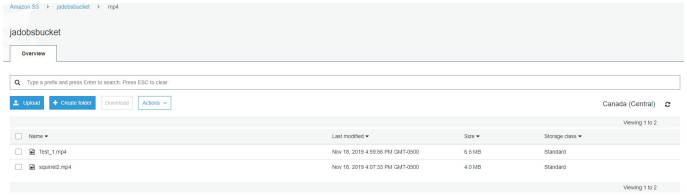

Fig. 3: S3 Bucket

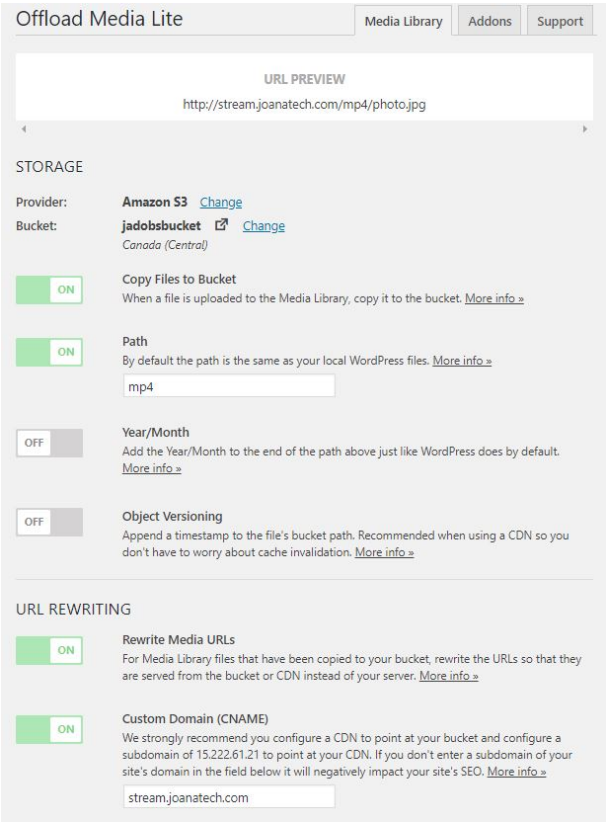

Fig. 4: Offload Media Lite

mobile apps, IPTV set-top boxes, Internet connected TV sets, game consultation systems and other networking devices for streaming live and On Demand Video, Audio, and rich Internet applications. The server can be used on most operating systems with the Java application. We used the open source website to install the Nginx software from https://nginx.org/ [3].We Downloaded nginx-vod-module open source package from github [2]. Installed nd module. Changed the configuration file of nginx to fit our application [4]. In configuring the NGINX our video on demand application name was dubbed "mp4" .

B. Content-Syncing

In order for the CDN to function correctly, the video content that is uploaded to the origin server should be pushed to all the edge servers. After that, the streaming requests would be directed to the edge servers i.e. the videos given to the origin server will be streamed back from the edge servers. While moving forward to implement this, we proposed an idea. In a live environment the origin servers would be already handling millions of requests to end-users trying to access the website and browse for content. We believed that we didn't want on top of that load the server with the task of syncing between the edge servers which in live environment could be hundreds and the video content in Terra Bytes. Therefore, we proposed that an intermediary would be present between the origin server and the edge servers where the origin server will sync the content only to this one intermediary and it would sync the content with the edge servers, thus relieving the origin server and allowing it to focus on end-users requests. In our implementation, we used an S3 bucket 3 as the intermediary. The video content uploaded to the origin server will be pushed by it to the S3 bucket and the S3 bucket will sync with the edge servers.

1) Origin to S3 Bucket: The website hosted on our origin server was done using WordPress. And for syncing with the S3 bucket we used a WordPress plugin: WP Offload Media Lite [5]. The plugin automatically pushes any video uploaded to the origin server to the specified S3 bucket. Then we use it to replace the URL for our videos with "stream.joanatech.com" which will resolve to our edge servers as will be discussed later in the report.

2) S3 bucket to Edge Servers: At the other end, we needed to sync the S3 bucket with our edge servers. To do so, we used Amazon Web Service Command Line Interface [6]. This tool allows us to manage our AWS services (servers,buckets, etc.) using the command line with scripts. We used the "sync" command which compares the source and target directories and uploads the new content. To insure that all new video contents will keep syncing with the edge servers we created a cronjob 5 that runs the sync command every specified duration. We specified it as 1 minute for our project 6.

A cronjob (cron table) file, which specifies shell commands to run on a schedule on a regular basis.

C. Network & Load Balancers

1) Global Load Balancing: The streaming requests of the end-users need to be resolved eventually to an edge server. To do so we implemented two load balancing policies. We used Amazon Route 53 tool to assign a DNS name for

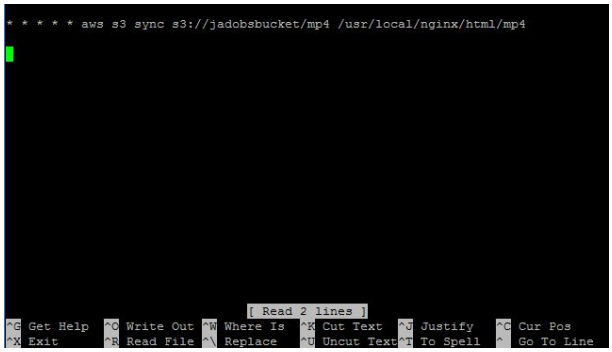

Fig. 5: AWS Sync Cronjob

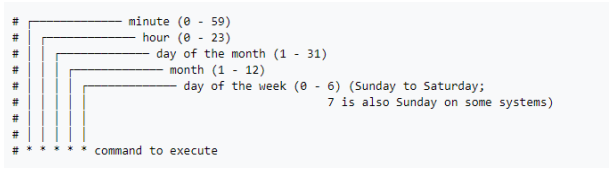

Fig. 6: cronjob format [7]

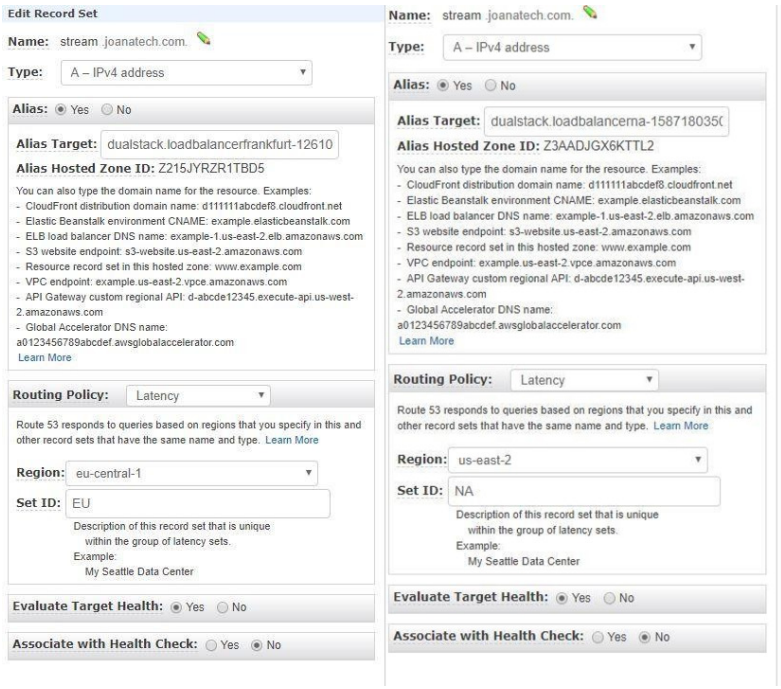

Fig. 7: Routing Policy in Target Groups

the streaming purpose and we created routing policies that would resolve that address to the edge servers. First, we load balance on a global level i.e. any requests from any region (Europe,North America,Asia, etc.) will be directed to either our North American servers (Edge servers in Ohio region) or European servers (Edge servers in Frankfurt region) according to the latency rule. The controller (AWS Route 53) 8 would calculate the latency between the edge-user and the regions (Ohio,Frankfurt in our example), and the region that would give less latency would serve the videos to the user. The DNS name assigned is stream.joanatech.com. All streaming requests will be made to that URL and the controller will direct the request accordingly to the region the would provide less latency.

2) Region Load Balancing: In a region itself the streaming requests should be balanced between the regional servers. To implement that we used AWS Elastic Load Balancer. An ELB is an entity with a specified DNS name within the AWS network. It would receive the user requests and direct them to the servers within its target group. Therefore, in the Ohio region we created a target group that includes the Ohio servers and assigned it to the Ohio ELB we created. The same implementation was done in the Europe region (Frankfurt) where we had our other edge servers. The DNS names of the two ELBs were given to the Global Load Balancer. After that step, any stream request to stream.joanatech.com would be resolved by the global load balancer to either ELBs (Ohio or Frankfurt) according to the latency rule and the ELBs would direct the request to the subsequent edge servers in its region and would balance the requests to them.

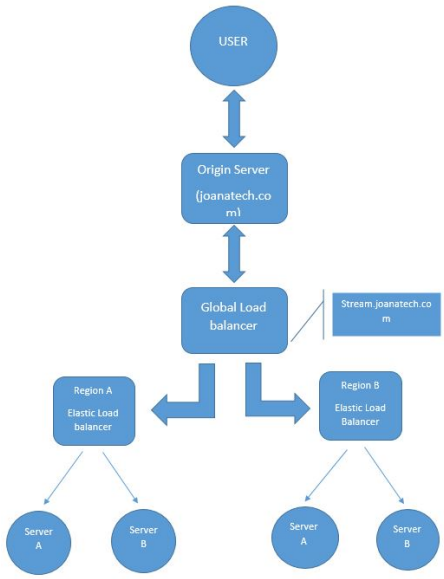

Fig. 8: The Streaming Process

III. Website

The front-end part of our project was implemented using WordPress. WordPress software was deployed in our Origin Server and was used to create the user interface . The website created was themed as a membership based webapp where one of the pages will list the videos 12 which are uploaded by the admin and are available for streaming. We used a WordPress plugin named FV Player [8] to embed our videos to the website.

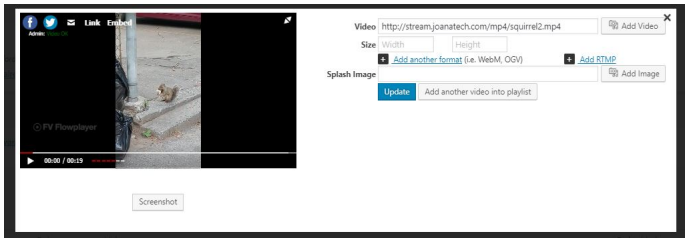

Fig. 9: Embedding the Video

The URL has the following format http://<Global Load Balancer DNS Name>/<VOD Application Name>/<mp4 Video Name>

IV. Results

To test our CDN we used a VPN service to create streaming requests from different regions. When the requests were done from Europe the Frankfurt servers' 11 egress packets spiked indicating that they were handling the streaming request. And when the graph was examined we saw that each request was divided between the servers. Similar values were observed for the Ohio servers 10 when the requests were done from North America. The results were as expected since we assumed that in most cases the requests that are closer to the edge server would have less latency and thus requests in North America would probably resolve to the Ohio Servers and the requests from Europe would resolve to the Frankfurt servers. However, it was not easy to predict which servers would handle the requests from other regions (Asia, Africa, etc.) but still the load balancer directed the traffic to the region which generated less latency with the end-user at the time.

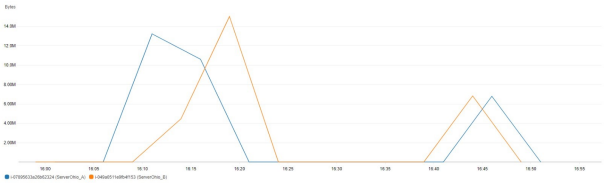

Fig. 10: Server Response - Ohio

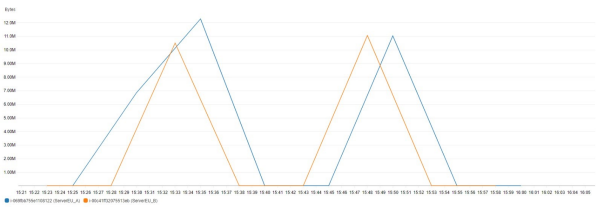

Fig. 11: Server Response - Frankfurt

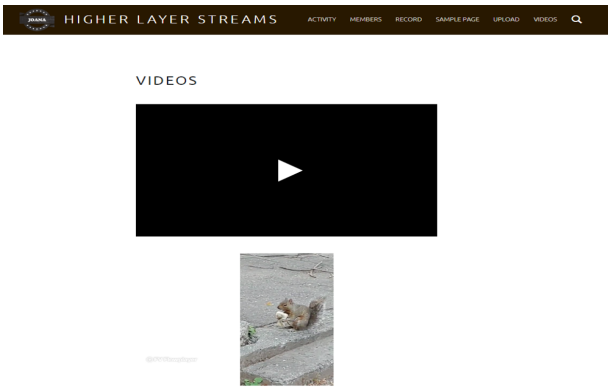

Fig. 12: Videos on DNS

V. Conclusion

We were able to implement a functioning Content-Delivery Network that hosts our video content which are uploaded first to the origin server and pushed later to the edge servers. The load balancing controllers we implemented allowed our requests to be handled with minimal latency relative to our servers and with a balanced approach for the servers. We were able to sync the video contents of the edge servers and allow any new video uploaded to the origin server to be accessed from the edge servers. The routing policies we created allowed for our origin server to have a DNS name 12 as well as our streaming service which was able to be accessed with its specified sub-domain name. The implemented website allowed our service to have a user interface where the user can browse for content and make streaming requests. Through several tools and methods we were able to create a service that is able to go live.

REFERENCES

[1] Ruether, Traci. "Streaming Protocols: Everything You Need To Know". Wowza Product Resources Center, 2019, https://www.wowza.com/blog/streaming-protocols.

[2] "Kaltura/Nginx-Vod-Module". Github, 2019, https://github.com/kaltura/nginx-vod-module. Accessed 16 Dec 2019.

[3] "Index Of /Download/". Nginx.Org, 2019, http://nginx.org/download/.

[4] "Nginx Conf For Nginx-Vod-Module On Ubuntu 16.04 With Nginx Secure_Link_Module". Gist, 2019, https://gist.github.com/lvillarino/c341b4cefc667dc616d64aa93538ee2d.

[5] "WP Offload Media Lite For Amazon S3, Digitalocean Spaces, And Google Cloud Storage". Wordpress.Org English (Canada), 2019, https://en-ca.wordpress.org/plugins/amazon-s3-and-cloudfront/.

[6] "AWS Command Line Interface". Amazon Web Services, Inc., 2019, https://aws.amazon.com/cli/.

[7] "A Beginners Guide To Cron Jobs - Ostechnix". Ostechnix, 2019, https://www.ostechnix.com/a-beginners-guide-to-cron-jobs/.

[8] "FV Flowplayer Video Player". Wordpress.Org English (Canada), 2019, https://en-ca.wordpress.org/plugins/fv-wordpress-flowplayer/. Accessed 17 Dec 2019.