Cloud-Based Video Streaming Service

Cloud Computing

School Of Engineering and Applied Science - Ahmedabad University

Group Details :

1) Het Jagani (1641007)
2) Rituraj Jain (1641020)
3) Hitarth Panchal (1641034)
4) Aayush Shah (1641065)

Abstract :

While demands on video traffic over mobile networks have been souring, the wireless link capacity cannot keep up with the traffic demand. The gap between the traffic demand and the link capacity, along with time-varying link conditions, results in poor service quality of video streaming over mobile networks such as long buffering time and intermittent disruptions.Using Cloud Computing technology we propose a solution (i.e Software as a Service(SaaS)) where the whole cloud infrastructure is developed and setup in-house. Also load balancer service is implemented internally in order to distribute the load on multiple servers.

Introduction :

Video streaming has come a long way from the old days when transferring video wasn't even possible on the Internet. Today, businesses all over the world use video streaming as a tool for marketing and communication, as well as a modern means of providing entertaining or educational content.Video streaming platforms have adopted cloud scaling in order to enable larger bandwidth and speed. These factors are necessary to handle heavier video requirements and provide a better viewing experience.

Existing Systems :

Today there are many well known video streaming services like Netflix, Amazon Prime, Youtube etc. which uses cloud as an infrastructure for computing and storage means.It includes keeping track of all the users, their preferences, the organization of the company's content, etc. Netflix uses Amazon S3(Amazon Simple Storage Service) cloud service which allows the users to save their data in cloud servers and access anytime and anywhere.

The architecture of Netflix is as follows :

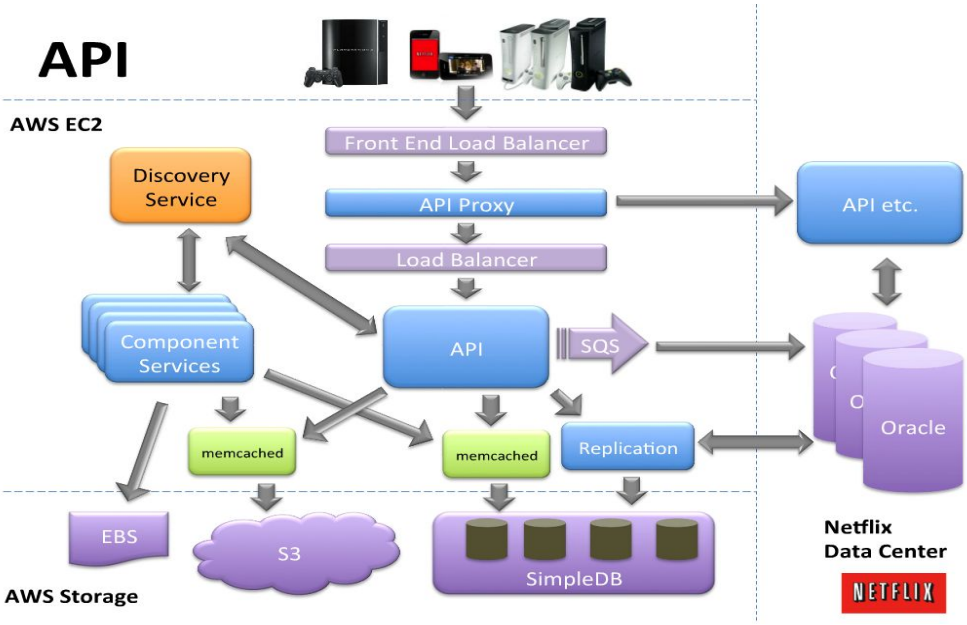

Fig: Netflix system architecture

Project Architecture :

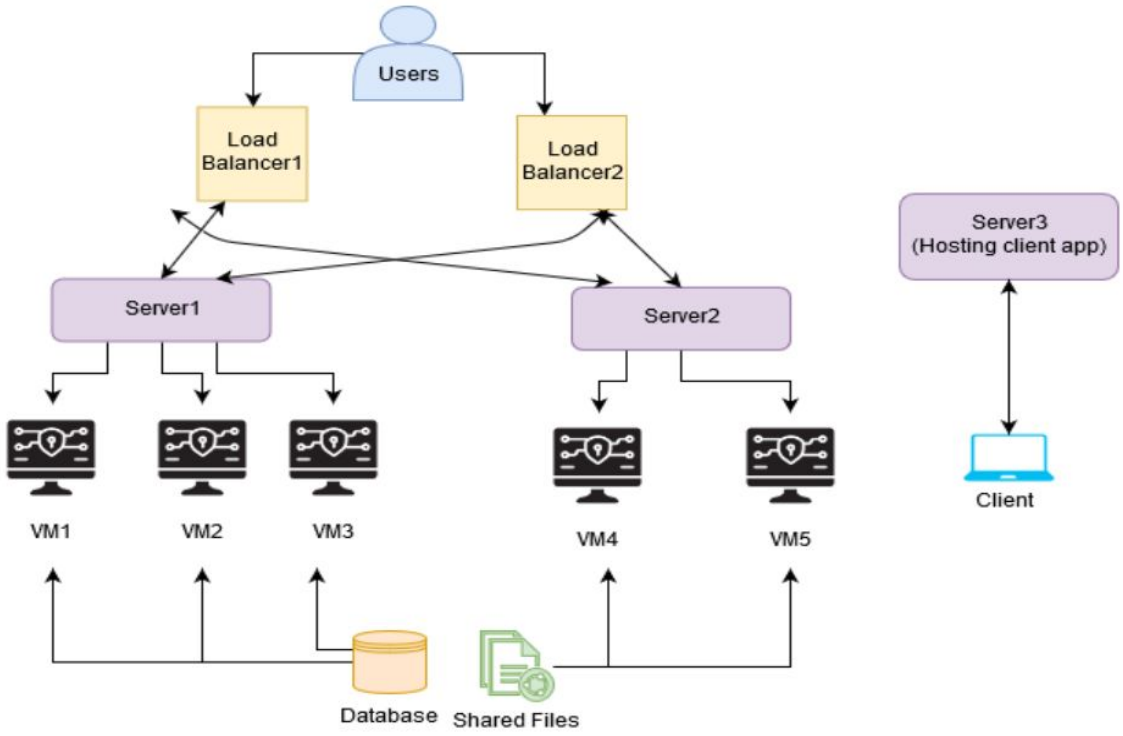

Fig: System Architecture

The architecture of our project setup is as shown in the figure above. Here, we have developed two load balancers and used the virtualization on the server machines to efficiently serve the streaming service. Following are the details of each of the components in architecture :

Load Balancer :

We have developed a load balancer service and hosted it on two servers. One of the servers will act as a primary endpoint for our video streaming service and another is the backup server which would be invoked if the primary server is not working. These load balancer services are internally connected to our service servers and distribute the requests between them in order to balance the load on each server.

Virtual Machines:

We used virtualization and created 5 virtual machines on top of two servers. These VMs will actually host the streaming service and on request will stream the video files from the storage.

Database and Shared File:

A shared file system is created between the virtual machines hosting the service. This file system stores the video files and a shared database which contains the metadata about those video files.

Client Application:

As a part of the frontend, we developed a web-based application in Python Flask, which will call the service APIs. This application is hosted on a separate server.

Infrastructure :

We have set up the whole infrastructure in-house in the Cloud Computing Lab. For virtualization, we have used Xen Hypervisor. We have chosen to do full virtualization for the project. On one server we created 3 VMs with 2GB RAM, 100GB storage and 2 VCPUs. On another server, we created 2 VMs with 3GB RAM, 150GB storage and 4 VCPUs. These virtual machines are used to host the service. Both server's specifications are: Intel i7 pro CPU, 8GB RAM and 1TB of storage. All the VMs run Ubuntu 18.04 Server OS.

To host the load balancer service, we have used the same servers but without virtualization. The servers run Ubuntu 18.04 OS. To host the client application another server with the same specifications is used. To store the metadata we used the SQLite database. A shared file system is created between all the VMs with the help of Glusterfs. All the video files

will be stored in this file system so that all the VMs can access the video files.

Implementation :

Server Side :

(1) REST API:

The server-side API is implemented using Spring framework of Java. Server-side implementation is basically a REST service which responds to three different following URL endpoints.

1. http://<server ip>:<port number>/movies

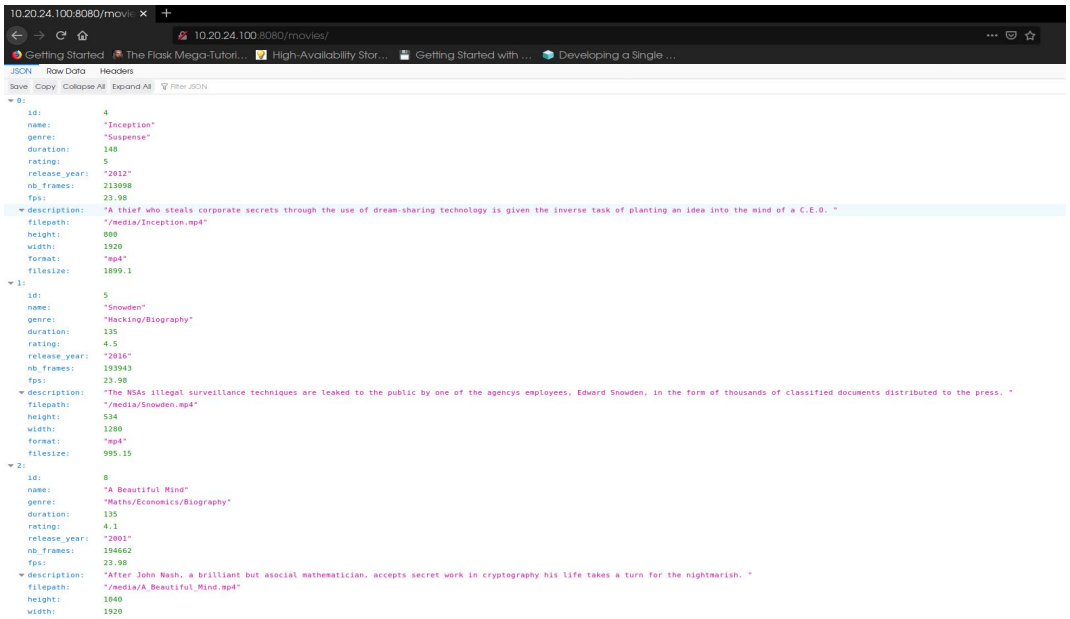

Fig: Received JSON response of all the movies

When the client goes to this URL with GET HTTP method, server provides a JSON response containing all the details about all movies. All the details are fetched from the database which is resided in a shared glusterfs volume. This glusterfs volume is shared between all the servers.

This particular shared volume also contains all the .mp4 files of the movies, whose details are stored in the database.

2. http://<server ip>:<port number>/movies/<movie id>

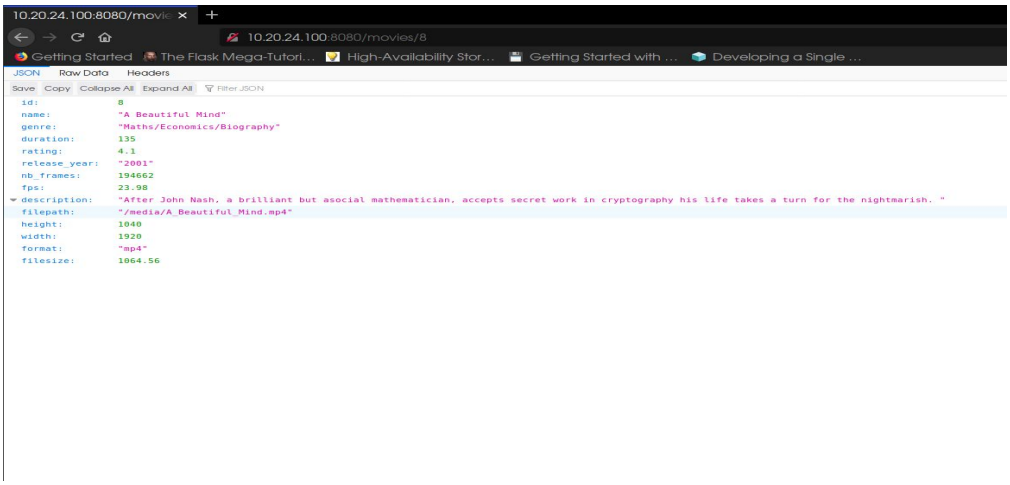

Fig: Received JSON response for particular movie id

This particular form of URL is called with GET HTTP method, server sends all the details about a movie given its id in form of JSON response. And this details are read from the database mentioned in the first point.

3. http://<server ip>:<port number>/movies/stream/<movie id>

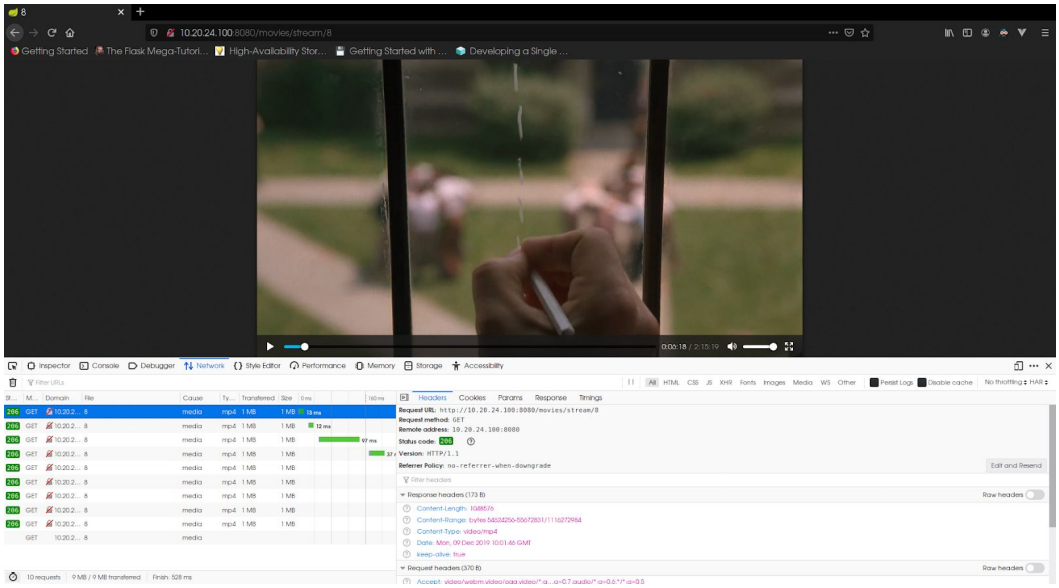

Fig: Video Streaming screen

GET HTTP method at this specific URL when entered. This HTTP request has a “Range” header set to a specific value. Range header defines what part of video it requires. E.g. if Range: 100- , then it means client requires partial video content after 100 bytes in the whole video. Server first finds the location of mp4 file of that particular movie (given movie id) from the database. After that server reads that mp4 file depending on the range header value server fetches that specific byte range of size 1 MB from the video file. And this data is sent in the body part of the HTTP response and header “Content-Type” set to “video/mp4”.

(2) Load Balancer API :

All the HTTP requests are sent to the load balancer as shown in system architecture figure. And the load balancer reroutes the requests to the server which has minimum system usage. Total five servers are up and running the REST API applications inside the virtual machines.

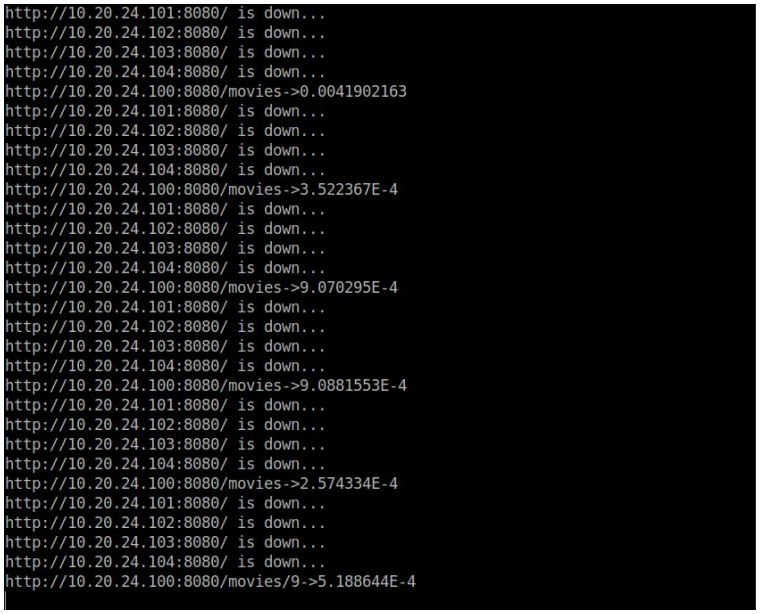

Fig: Terminal output from Load Balancer

Here the load balancer is also implemented as a REST service in java using Spring framework. Load balancer 1 is the primary load balancer and the second one is secondary load balancer. Client will first send request to the primary load balancer, if client receives 404 (Not found) or 408 (Time out) status code, then only client will send the request to the secondary load balancer. Both the load balancers are ran in two different physical machines.

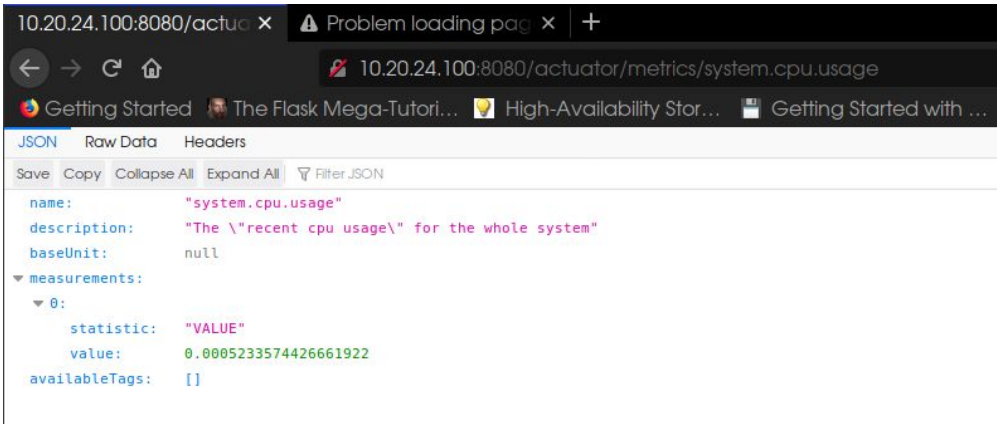

Fig: System cpu usage metrics received from API call

All the ip addresses of the servers are known at the load balancer. Also all the servers uses a functionality called Spring actuators. When this functionality is implemented, a series of URL endpoints are exposed by the server application, which responds (HTTP response JSON) to the caller (HTTP request) when requested with the system usage details of the system in which the application is running. This functionality is crucial while implementing the load balancer program. Now load balancer program can send requests to all the servers at a particular URL endpoint, in response load balancer will get system usage details of all the server machines.

After getting system usage from all the servers, it finds out which server among all the five servers has minimum system utilization value. And the client request is rerouted to the server with minimum system

usage. If one or more servers are not running load balancer will ignore them and will only consider the ones which are working.

HTTP request rerouting is done by using the “Location” header in the HTTP response with status code 307 (Temporary redirect). Load balancer simply sets “Location” header to a server URL endpoint (server with minimum system usage). After rerouting the server application responds directly to the client. Both the load balancer work in the same way as explained above.

(3) Database and Media :

All the media files (movie mp4 files) and a database file are stored in the shared glusterfs volume. Here SQLite database is used for its efficiency, reliability, and simplicity. SQLite databases are lighter compared to other databases MySQL, Postgre SQL etc.

GlusterFs volume is shared between 6 machines. Five virtual machines running the server applications and one content creator machine. Content creator program is running in a separate physical machines, its purpose is to upload media files and add entries in the sqlite database which are stored in the shared gluster volume. Content creation script is basically a python program running indefinitely on a machine. It will monitor a directory in its system.

Content is added in the shared gluster directory by the script. But first admin will create one folder containing a .mp4 video file and json file. JSON file will have all the details about the movie (e.g. name, genre, duration, description etc) inside a directory which has been monitored by the Python script. When it detects a folder in the monitor_directory, it will first copy the .mp4 file to the shared gluster directory and then will add an entry in the sqlite database of that particular movie on successful copy operation. When file copying is not successful entry won’t be added inside the database. A database entry contains movie name, genre, description,

rating, duration, frame width, frame height, video file size, etc about a particular movie. This is all done by the python script, it checks the monitor_directory every 5 mins to know if new content is uploaded by the admin. Since gluster volume is made using replication functionality, all the data inside it is replicated to all the six machines. Upon database entry the database file is locked hence no other machine can access that file during that period. Also all VMs running the server application can open the same video files independently because replication method creates six replicas of the same file inside each of the six machines. Synchronization and replication is done by the glusterfs service running inside each of them.

Client-Side :

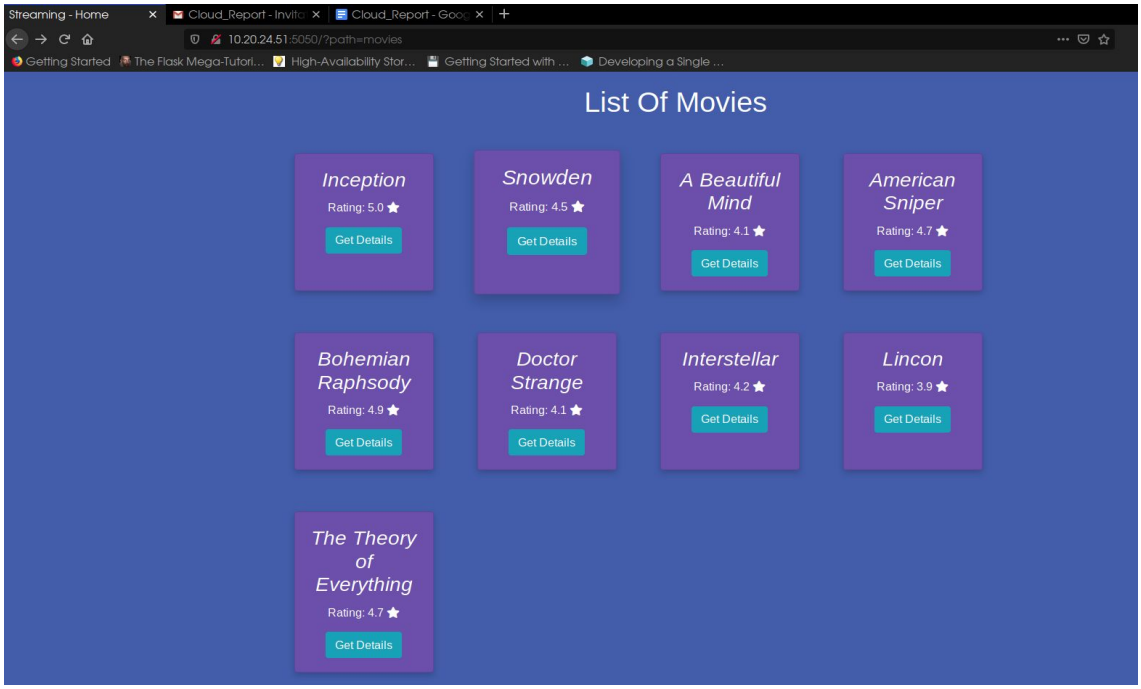

Fig: Homepage of the application

The client-side web application is implemented using the Flask framework of Python3. The application makes an HTTP request to the Rest service to get the list of available movies and receives a JSON response, parses the response and displays the list on the homepage of the application (as shown in figure). Once the user selects one of the movies,

the application makes a request to get the details of that movie, receives a JSON response and displays the details. If the user decides to watch that movie, the user is redirected to a video player in the browser. Then, the application sends HTTP 206 Partial Content requests for the movie content and receives data in chunks of 1mb. The data is then decoded and displayed in the video player. If at any point, the master load balancer is down, the application automatically makes requests to the backup load balancer.

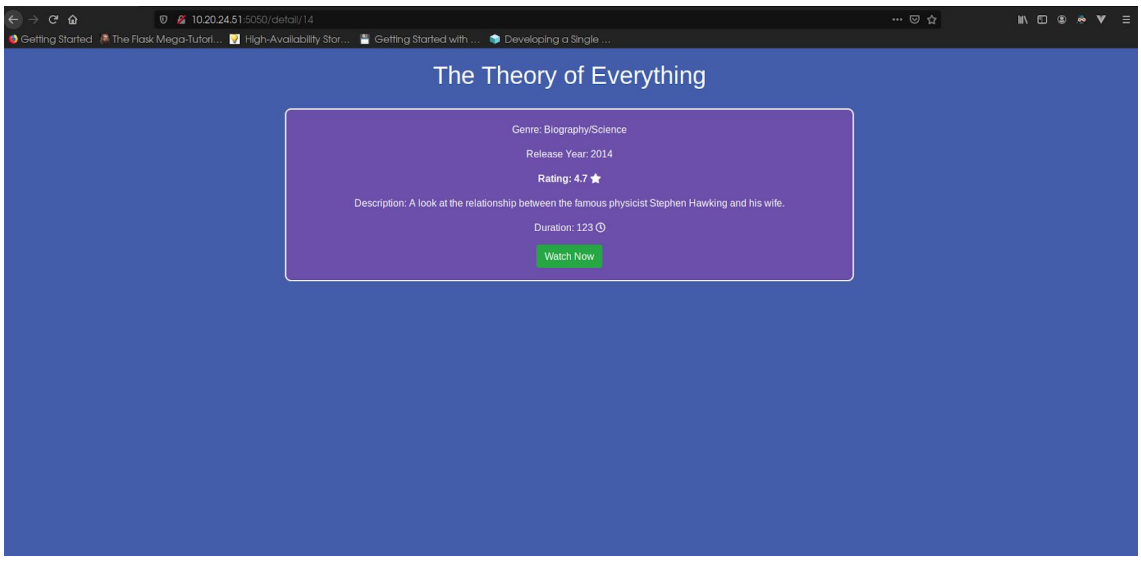

Fig: Movie details page of the application

Conclusion :

In this project, we were able to install and configure Xen Hypervisor and achieve full virtualization. 5 servers were set up (on virtual machines) on two machines in the lab so that multiple requests can be served concurrently. These servers were used for delivering video content in the web application. The movies were stored in shared storage between the servers which are served as per the request. We also set up two load balancers on the other two machines (one primary and one backup in case the primary load balancer is down). The load balancer checks the load of each server and redirects user requests to the server with the minimum load. We also implemented a client-side web application for our streaming service in which the user can get the list of available movies, their details and watch any movie.

Future Scope :

As the future scope in this project, we could further implement the load migration techniques. This would help to reduce the downtime of the service. Whenever a server is down the load can be migrated to another server. The current load balancing algorithm distributes the load based on the current load on the system. This algorithm can be improved by considering the parameters such as latency, number of requests, etc.

The network speed in different areas is unreliable. Hence we can serve the videos of different sizes and codecs based on the latency between client and server. Such a feature is implemented in most of the video streaming platforms such as Netflix, Youtube, etc. The service developed in the project can be extended by adding more features to it.

References :

Live Streaming - Media & Entertainment Solutions | Google Cloud. (n.d.). Retrieved from https://cloud.google.com/solutions/media-entertainment/use-cases/live-streaming/.

A Practical Guide to Deploying Cloud-Based Video Services. (n.d.). Retrieved from https://pages.awscloud.com/GLOBAL_IND_practical_guide_whitepaper_2019.html

Load Balancing in Cloud Computing. (2018). International Journal of Recent Trends in Engineering and Research, 4(3), 118–125. doi: 10.23883/ijrter.2018.4105.vqpyq

Virtualization. (2012). Reliability and Availability of Cloud Computing, 16–28. doi: 10.1002/9781118393994.ch2

Kennedy, A. (2019, August 30). REST API: What It Is and How to Use One for Streaming: Wowza. Retrieved from https://www.wowza.com/blog/rest-api-streaming.

Building a RESTful Web Service. (n.d.). Retrieved from https://spring.io/guides/gs/rest-service/.

Grinberg, M. (n.d.). Designing a RESTful API with Python and Flask. Retrieved from https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask.