Cover Page

Task: Cloud System Design and Evaluation

Name: Sai Kiruthika Saravanan

Student no: 219008860 (sks66)

Group no: 23

CO4219/CO7219 Internet and Cloud Computing

Cloud System Design and Evaluation

Task 1: Design and Technological choices for your private cloud

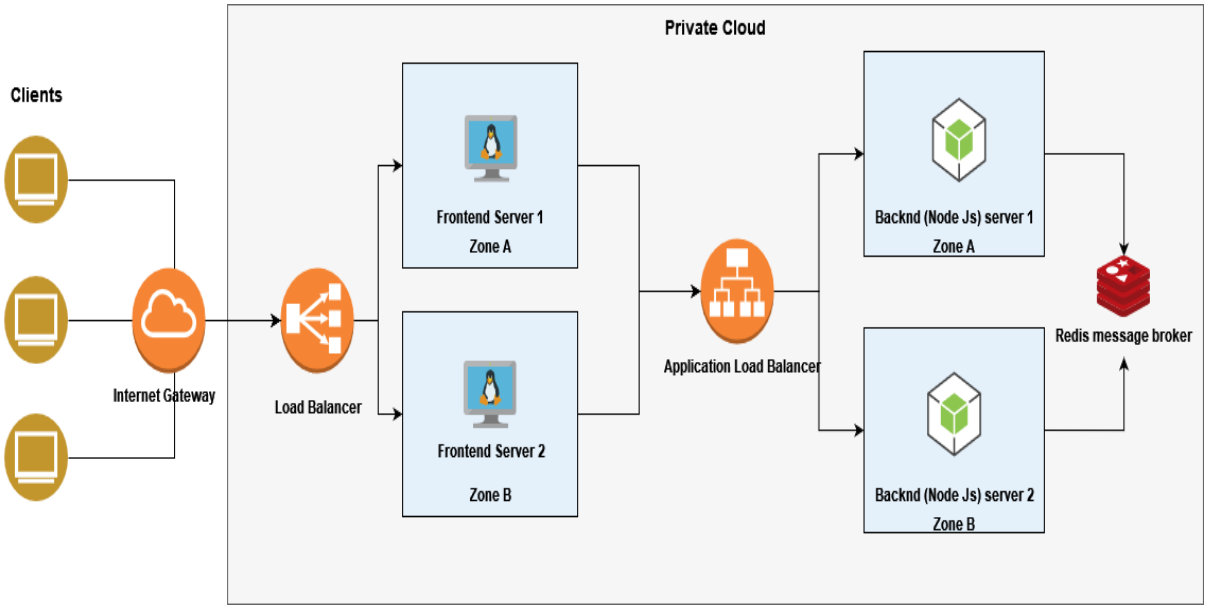

Fig 1.0 Architecture of our private cloud

Tools used in this architecture:

- Virtual Machine running on a Linux OS.
- Proxy Servers like ngnix is used.
  For Frontend we are using webservers (E.g., nginx, Apache Tomcat, Apache HTTP)
- For Backend we used Node.js.
- Load Balancing is done by Digital Ocean and Linode.
- We are using Digital Ocean and Linode Cloud Platform
- Cache memory is used (E.g, Redis Message Broker)

All these tools make up our architecture of the private cloud. Detailed Explanation of tools are given in the implementation of components part i.e.) Task 2

Consistency:

The above architecture is a consistent module because we have used Redis Message Broker for both the frontend as well as the backend. Redis will easily do data synchronization. The Redis Cache Memory will follow a Master-Slave Architecture and all the data's will be synchronized between the servers and it will be available in all the servers which are equally distributed between the load balancers. Consistency is nothing but data synchronization and our architecture is consistent.

Scalability:

In our project we have used Load balancer to achieve Scalability. We can easily scale all the servers connected to the load balancer. We have done horizonal scaling in our project. We have created multiple instances which has data in it. It is easy to duplicate all the instances which can be again scaled easily. The load balancer will then equally distribute the data between the instances. The performance of our project is good because we have achieved scalability.

Availability:

Availability is the process of processing information at least to one node if any of the other instance is not working. In our case we are using Load balancer for frontend and for the backend we are using application load balancer for transferring of data. If any instance is not working in any location due to any internal failure the load balancer will be able to send data to another instance which is connected to other different location and the data flow will not stop but in our project, we have achieved only minimal availability.

Security:

In our project we have few IP Address used all over the architecture. There is a public IP used in the project and other all IP address are not visible to anyone, and we have installed firewalls to protect them from all attacks. This is how our project is more secure.

Fault Tolerance:

Fault Tolerance is also maintained in our system for example when the client sends an request through an loan balancer and when any of the instance connectivity is lost yet the data will flow through the front end and then it will reach the backend and even if one of the instance is no longer working the system will still run and synchronize the data with the use of Redis cache memory and this is how fault tolerance is attained in our system.

Network Topology:

The Network Topology of our system works like the client sends a request to the load balancer and the load balancer is connected to two instances and the ngnix is used web server for proxy and the data are equally divided and sent to the frontend for deployment and after the deployment we have connected the front end with the application load balancer. The application load balancer is deployed in digital ocean since we got some free subscriptions in it and this load balancer is in turn is connected to node.js for backend and then the backend is furthermore connected with a cache memory called as Redis Cache memory for data synchronization. This is how the network topology works for our application.

Task 2: Implementation details of your private cloud:

- **Environment used:**

  We have used Digital Ocean to implement our virtual machines and Linode since both the website has free subscription to host our virtual machines.

- **Client:**

  We have three clients who are sending requests and searching some information in a website.

- **Load Balancer:**

  Load balancing is the process of dividing workloads and processing resources in a cloud computing environment. In our implementation the public IP is first enabled. The load balancer is used in the first step because we can equally distribute the data to the front-end server which has node.js script in it. Load balancing is used to optimize the data and to control the traffic. We also configured Nginx software for proxy, and it has a main role in this project. Since it has many SSL Certifications it is trusted and used as web server and proxy server. The cache methods are implemented in the webserver for quick transfer of files to the frontend servers. The load balancer uses Round Robin method to equally distribute the traffic of the data given by the client and the data is exposed to public IP. The requests given by the client is distributed equally in turns.

- **Frontend Server:**

  Frontend is implemented on a Linux machine which in turn runs on a Ubuntu OS and it is installed with node.js because we are using next.js for frontend and node.js for backend data. Nginx software is an opensource software which is configured in front-end to perfume reverse proxy. Same configurations i.e.) the copy of the same architecture is configured in other server in different location. For our project we have choose London for one server and Germany for another server and same architecture is followed for both the servers and for the backend we have connected node.js.

- **Backend Server:**

  Backend Server is implemented and has same architecture as Frontend Server. For Backend we have installed node.js in the Linux machine**.** For webserver we have again configured the open-source software called as nginx which can be used for reverse proxy. Same as frontend for backend also the copy of another server is implemented in the different location and both the data of the servers are synchronized by having a cache memory.

- **Application Load Balancer:**

  We have implemented our Application Load Balancer from Digital Ocean. These are the online load balancer available over the internet. We have connected this load balancer with the backend data, and it links the private IP address to the network and when-ever the requests come from the front end the Application load balancer will be able to divide the traffic equally and send it to the servers. The

Application Load Balancer is implemented in such a way that it will pass the data to the server which has minimum no. of connections.

- **Redis Message Broker:**

  Redis Cache memory is an open-source software. We have connected this cache memory with our backend servers. The Redis message broker uses a in memory cache data. The Cache Memory will store all the data in cache memory, and it is used for synchronization of data between the two servers.

- **Data Monitoring:**

  We have used Digital Ocean and Linode and the data which is stored outside is auto monitored by them and for the data which is stored internally we use logging of data methods to analyse the data which are stored internally.

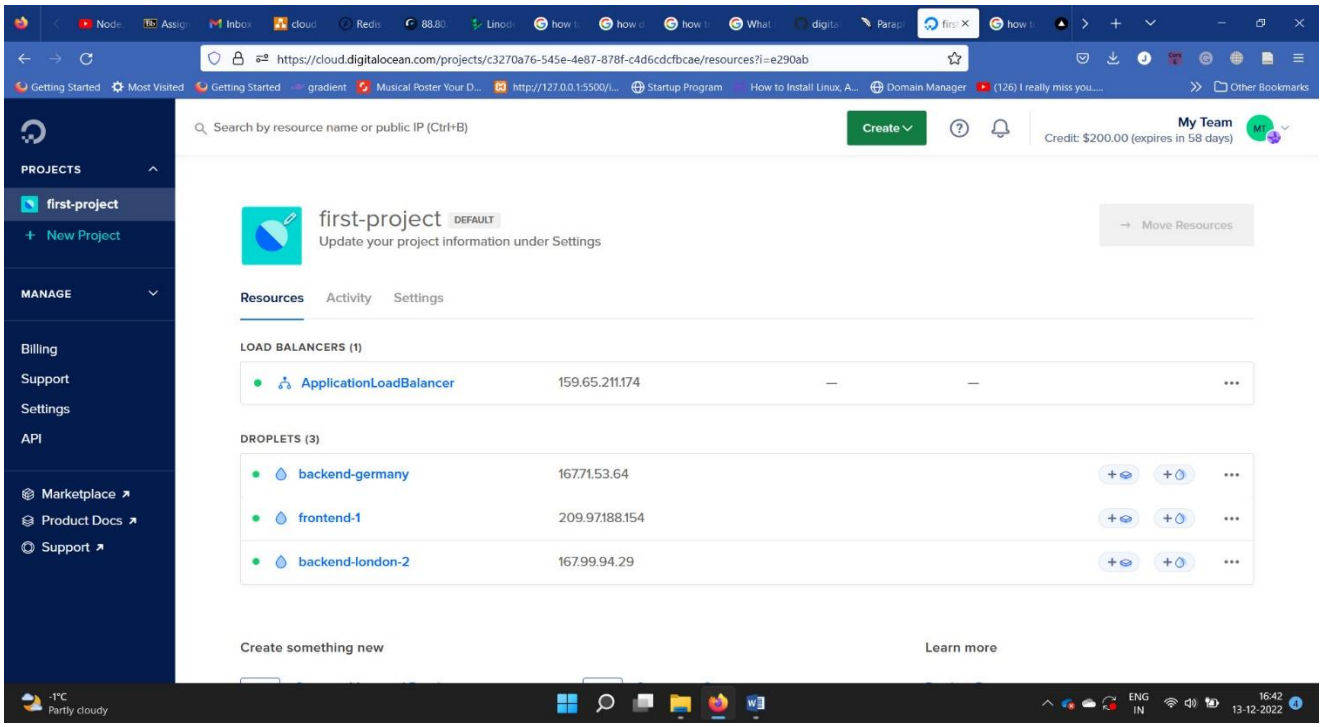

Fig 1.1 Data Monitoring Dashboard

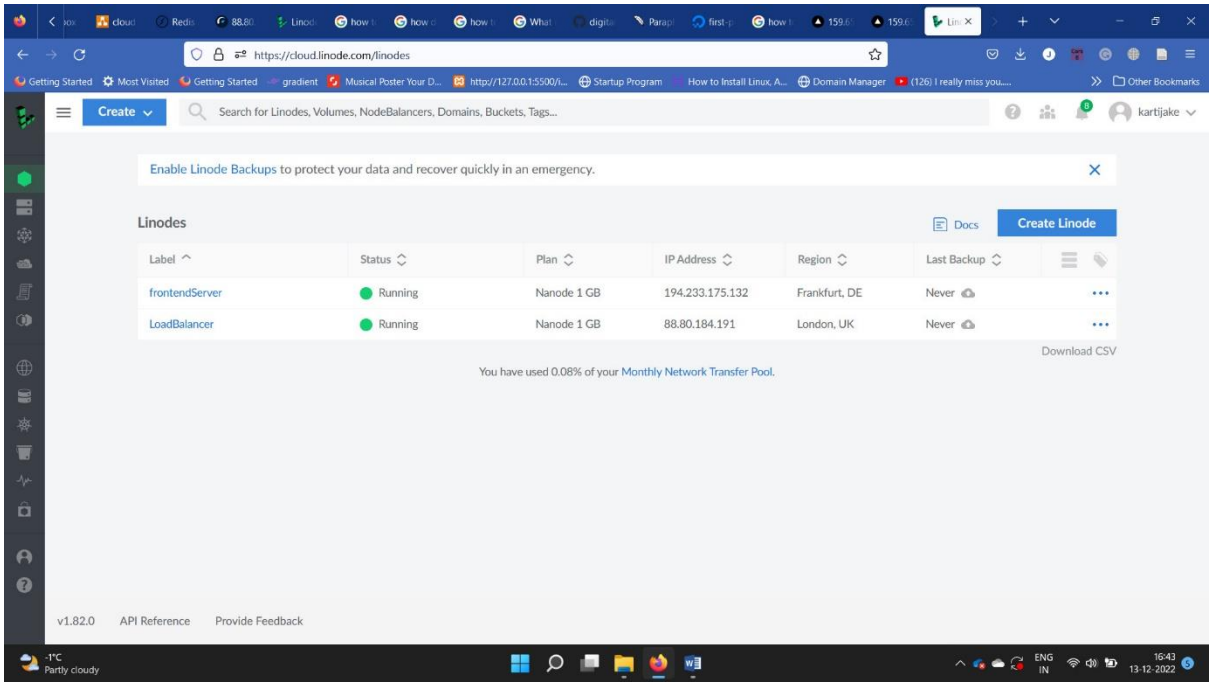

Fig 1.3 Linode Dashboard

Task 3: Development of a Distributed Whiteboard Application:

We have designed a White board Application which is a consistent model. Our white board application is connected to two nodes simultaneously with the help of a load balancer. The white board application is designed with a frontend as well as the backend. Node.js is used for storing data in the backend. We have also used an application load balancer connected to the instances of the backend. Our white board application server is stateful because we are using web sockets for our connection. The basic idea behind our White board application is it consists of two nodes which has the same access and data replication as of the original node. Our white board application is consistent because when I draw something in the first node for example circle all other nodes will get the same diagram and if I made any changes in the third node all other nodes will get an update in the diagram. This is how our white board application works. The features which we have included in our whiteboard application are:

- ✓ A pen tool
- ✓ Colour pallet
- ✓ Circles
- ✓ Rectangle
- ✓ Eraser
- ✓ Undo option

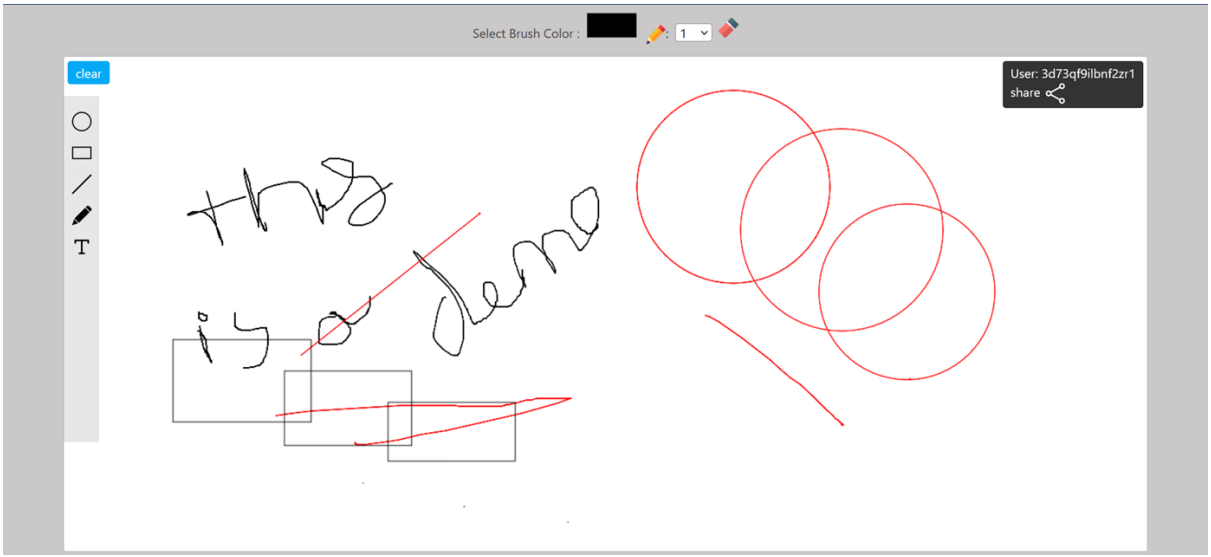

Fig no. 1.4 White Board Application

The white board application is built with the WebSocket, and sockets are used to build the connection within the backend server. We used web sockets in our real time project because it will keep the HTTP connection alive. The first step of our web socket is the handshake process when the client gives a handshake the http connection will be established between the client and the server, and the data exchange process will take place. After the connection has been built both the client and the server will now exchange the data simultaneously. This is how we have connected our application using a web socket. By this connection we can be able to see the data in all the nodes when we draw anything in the white board application. We have also connected a cache memory to our backend server which is mainly used for data synchronization. Our data integrity is maintained by it. We are also having some firewalls used in our application for security purposes because we will be using some IP addresses for connectivity. Our application is consistent and fault tolerance and I have addressed the following topics below

Fault Tolerance:

Fault Tolerance is also maintained in our system. When we draw anything in our white board application all other nodes will also get reflected by the same data. Sometimes due to any technical faults the nodes do not respond back to the servers. Yet the load balancer in our system continues to work by sending the relevant data to all other nodes and thereby all other screens will get the data. This is how fault tolerance works in our system.

Consistency:

The above application is a consistent module because we have used Redis Message Broker for both the frontend as well as the backend. Redis will easily do data synchronization. The Redis Cache Memory will follow a Master-Slave Architecture and all the data's will be synchronized between the servers and it will be available in all the servers which are equally distributed between the load balancers. Consistency is nothing but data synchronization and our architecture is consistent.

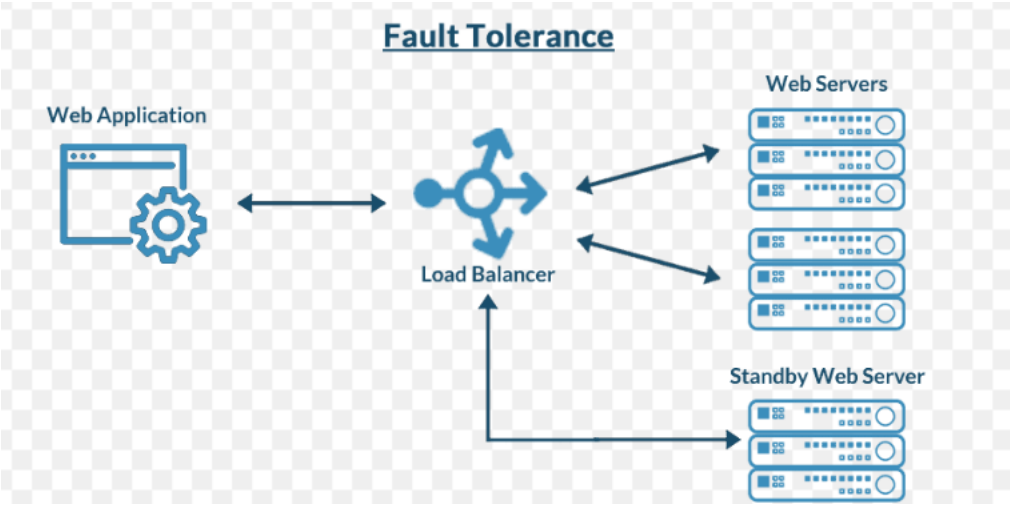

Fig 1.3 Fault tolerance of our system.

Task 4: Individual Contributions and URL of the group video:

For this project we have contributed as a group as well as an individual. We have worked on the architecture equally and we have given our outputs and ideas for it and after collecting all our ideas we designed our architecture for this system. I have personally contributed on the development of the frontend architecture and hosted my frontend architecture on London server. I have also worked on the ideas of WebSocket's and how to connect them for processing the data from frontend to the backend. We have been discussed frequently on WebSocket's since it was a new concept for many people in our group and the outcome of our discussion had a great impact in creating a consistent model with WebSocket's running in our backend. We have also worked with maximizing the availability of model, yet we were able to attain only the minimal availability of our system. Overall, we have worked as a team and contributed equally on the development as well on the deployment of system.

URL of the group video:

https://pro.panopto.com/Panopto/Pages/Viewer.aspx?tid=d6fe984f-0aea-41b7-a722-af6b00b8be5f&start=0

Task 5: Critical Review of the system

Strengths:

- **Scalability:**

  Scalability is achieved in our application by using Load Balancer which will help us to scale a large amount of data sets by duplicating the instances. This is one of the major advantages of this application.

- **Security:**

Many IP address is used in the development of this application and to improve security we have implemented few firewalls which will prevent attacks from attackers.

- **No Data Loss:**

  There is no loss of data while transitioning in frontend and backend because both the servers are attached to Redis Cache memory. This Cache memory will help in Data synchronization, and this helps is managing the data.

- **Fault Tolerance:**

  The application achieves Fault Tolerance by using the load balancer so if there is any fault in the instances, yet the system will be processing the data with the other available instances or servers.

- **Metrics:**

  Since we used Digital Ocean as our cloud provider, we will be able to view our own metrics of the system since it will auto generate it.

Weakness:

- **Minimum Availability:**

  We have tried to achieve maximum availability in our project using Load Balancer, yet we have achieved only minimum availability.

- **Digital Ocean:**

  We have used Digital Ocean as the cloud provider. Since we implemented the whole project in a free trail it will last only for few days, and I see this issue as a major weakness for project.

- **Single Cache Memory:**

  We have used only one cache memory for data synchronization so the speed of the system is little bit slow it would be better if we have multiple Cache attached to the system for fast and efficient use of data.

- **Failure of Data Centre:**

  If one datacentre gets interrupted with any failure the whole system in that area will crash. So, it would be better if we scale the entire data with load balancer.

Alternative methods of improving our system:

If I have given another chance to improve our system, we would have made our system and data more **available** to everyone who use it and we may use any other cloud platform which has more access to our data. I would have researched more on **multiple cache** memory and how to implement them on our database. We can also use **Redis pub-sub** method which is used to synchronize data for large set of channels and even we can also research more on working of **multiple load balancers** since they will give us more consistent and scalable application. The major changes I like to change if I would have given another chance are:

Multiple Load Balancer:

The first change I like to make is executing Multiple Load balancing system. In the existing system we have used one load balancer but if we use multiple load balancer, we distribute the data to n number of servers which will create a better scalability to the system.

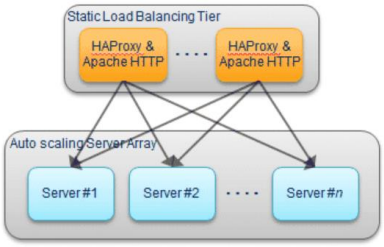

Fig 1.4 Multiple Load Balancer