



# VIT<sup>®</sup>

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

**School of Information Technology and Engineering**

**Winter Semester 2022-23**

**B.Tech (Information Technology)**

**Course code: ITE3007**

**Course Name: Cloud Computing**

**Slot : C1**

**Digital Assignment 2**

**Submitted by:**

Yogender

20BIT0133

**Submitted to:**

Dr. Siva Rama Krishnan S

## **Problem: Implement a failure over mechanism using virtualization**

In the case of virtualization, a failover mechanism involves replicating a virtual machine (VM) and its associated data to a secondary location, such as another server or data center. This secondary location is designed to be able to take over the operations of the primary system in case of a failure, allowing for business continuity and preventing data loss.

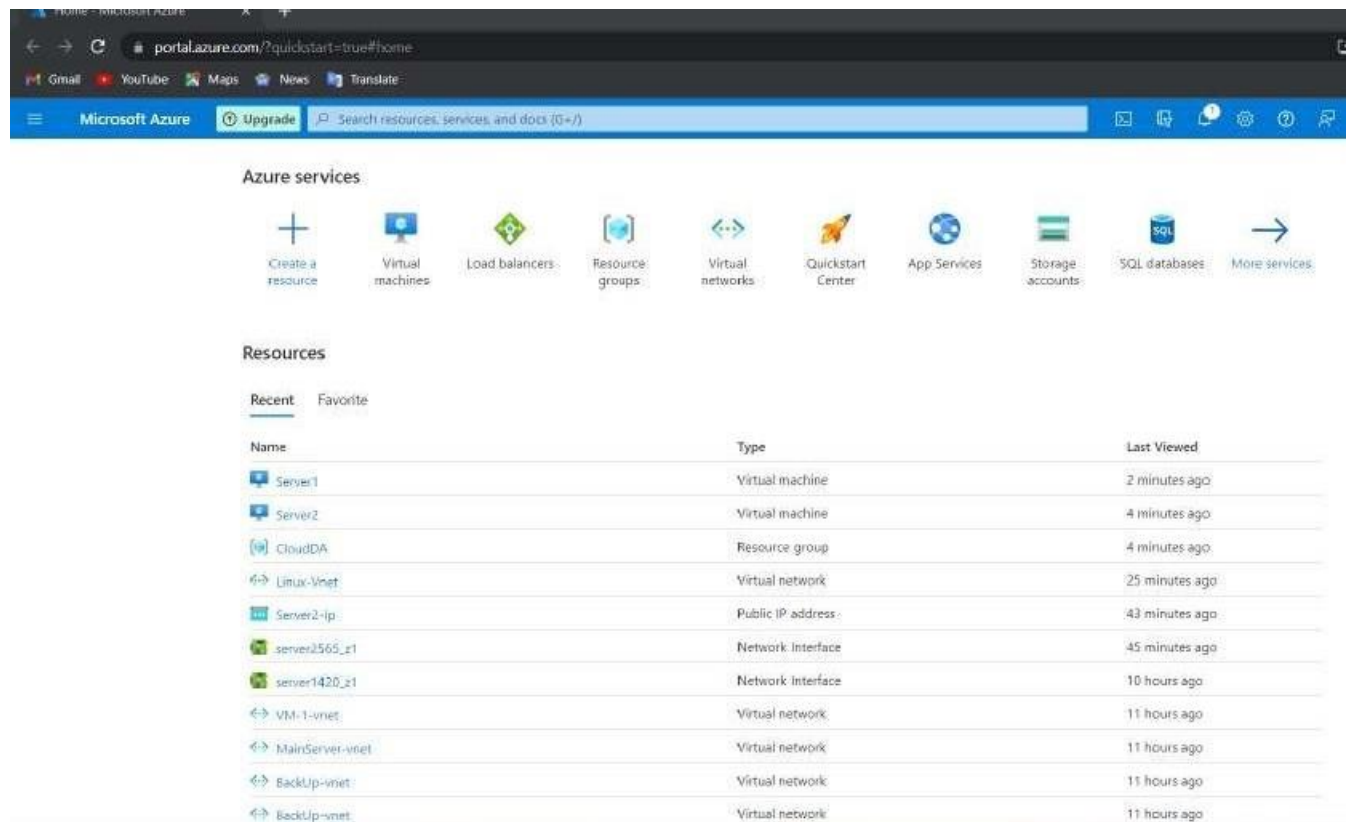
Failover mechanisms are an important part of disaster recovery planning and can help minimize downtime and disruption to business operations. They are particularly important for mission-critical systems that cannot afford any interruptions or downtime, such as those used in financial institutions, healthcare organizations, and other industries where data integrity and availability are crucial.

### **What is a failover?**

Failover refers to the process of switching over to a backup system or component in the event of a failure or outage of the primary system. In the context of virtualization, failover typically involves automatically shifting virtual machines from a primary host or server to a secondary host or server when the primary one experiences an issue or failure. The failover process is intended to minimize downtime and ensure business continuity by quickly and seamlessly transferring operations to the backup system or component. Failover mechanisms may involve the use of redundant hardware, virtualization clustering, or other techniques to provide high availability and minimize the impact of failures.

### **Solution:**

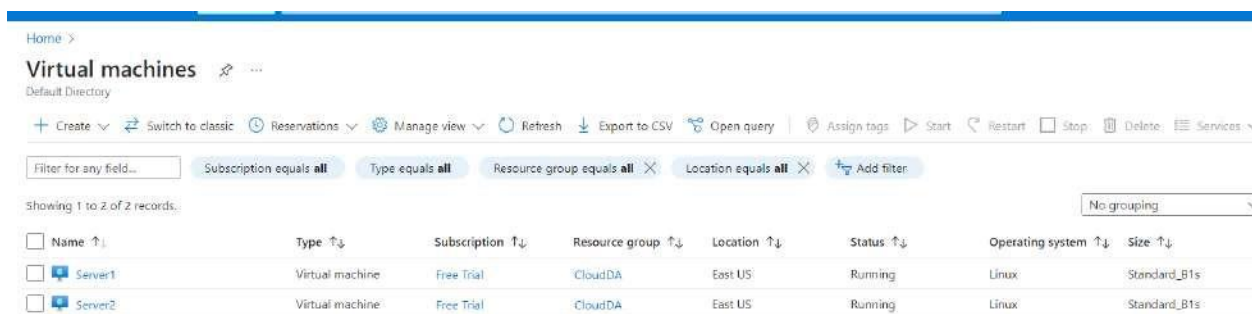
1. **Scalability:** Azure allows organizations to quickly and easily scale up or down their resources as needed, allowing them to quickly adapt to changing business needs and save costs by avoiding overprovisioning.
2. **Flexibility:** Azure offers a wide range of cloud services, including computing, storage, networking, databases, machine learning, AI, and more, which can be used to meet the needs of a broad range of industries and use cases.
3. **Hybrid cloud capabilities:** Azure offers strong hybrid cloud capabilities, allowing organizations to seamlessly integrate on-premises infrastructure with Azure services. This can be particularly useful for organizations that want to gradually transition to the cloud, or have regulatory or compliance requirements that necessitate keeping some data or applications on-premises.
4. **Security and compliance:** Azure has a strong track record of security and compliance, with numerous certifications and compliance frameworks, such as ISO 27001, SOC 2, HIPAA, and GDPR, which can provide peace of mind for organizations that deal with sensitive or regulated data.
5. **Integrated with Microsoft ecosystem:** Azure is tightly integrated with Microsoft's suite of productivity tools and services, such as Office 365, Dynamics 365, and Power BI, which can simplify management and streamline workflows for organizations already using Microsoft products.



The above shown is the dashboard of Microsoft Azure.

We have created Linux-Vnet in the resource group CloudDA.

Next, we will create two Virtual Machines under the same V-Net and same Resource Group.



We have created **Server 1** and **Server 2** under same Resource Group **CloudDA** and in the Virtual Network **LinuxV-Net** that we created above

Setting up Server1 (as Primary Server):

**Private IP address: 10.0.0.6**

```
azureuser@Server1:~$ hostname
Server1
azureuser@Server1:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.0.0.6  netmask 255.255.255.0  broadcast 10.0.0.255
    inet6 fe80::6245:bdff:feda:8f1f  prefixlen 64  scopeid 0x20<link>
    ether 60:45:bd:da:8f:1f  txqueuelen 1000  (Ethernet)
    RX packets 4436  bytes 5462459 (5.4 MB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 996  bytes 369990 (369.9 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
>>
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop txqueuelen 1000  (Local Loopback)
    RX packets 188  bytes 18634 (18.6 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 188  bytes 18634 (18.6 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

## Setting up Server2 (as BackUp Server)

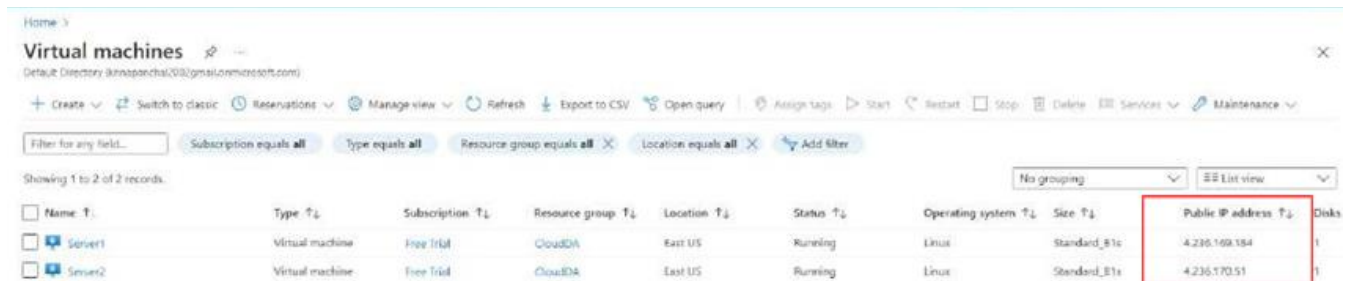
**Private IP:10.0.0.7**

```
azureuser@Server2:~$ hostname
Server2
azureuser@Server2:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.7 netmask 255.255.255.0 broadcast 10.0.0.255
    inet6 fe80::6245:bdff:fed4:5ff8 prefixlen 64 scopeid 0x20<link>
    ether 60:45:bd:d4:5f:f8 txqueuelen 1000 (Ethernet)
    RX packets 4458 bytes 5467441 (5.4 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1024 bytes 379732 (379.7 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 188 bytes 18626 (18.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 188 bytes 18626 (18.6 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

azureuser@Server2:~$
```

In the below screenshot we can see the Public IP of both the Virtual Machines that we have created.

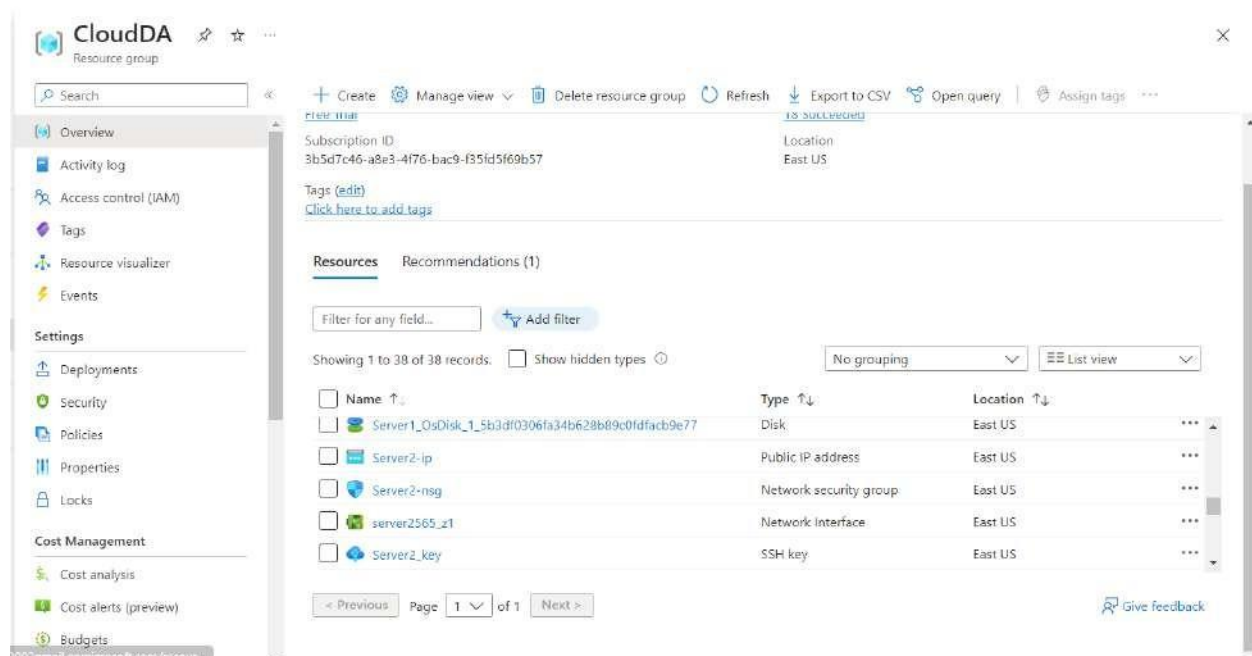


Name	Type	Subscription	Resource group	Location	Status	Operating system	Size	Public IP address	Disks
Server1	Virtual machine	Free Trial	CloudOn	East US	Running	Linux	Standard_B1s	4.236.169.184	1
Server2	Virtual machine	Free Trial	CloudOn	East US	Running	Linux	Standard_B1s	4.236.170.51	1

**Public IP of Server1- 4.236.169.184**

**Public IP of Server2- 4.236.170.51**

Two Virtual Machines are created under same V-net that is Linux-VNet  
In the resource group CloudDA



We will be using two ways to implement the failover mechanism one is using haproxy and other using Load Balancing of Microsoft Azure

Now, we install haproxy on server 2.

```
azureuser@Server2:~$ sudo systemctl status haproxy
● haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2023-03-30 21:02:16 UTC; 30s ago
     Docs: man:haproxy(1)
           file:/usr/share/doc/haproxy/configuration.txt.gz
  Process: 2090 ExecStartPre=/usr/sbin/haproxy -f $CONFIG -c -q $EXTRA_OPTS (code=exited, status=0/SUCCESS)
 Main PID: 2101 (haproxy)
    Tasks: 2 (limit: 990)
   CGroup: /system.slice/haproxy.service
           └─2101 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
             └─2102 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid

Mar 30 21:02:16 Server2 systemd[1]: Starting HAProxy Load Balancer...
Mar 30 21:02:16 Server2 haproxy[2101]: Proxy http-in started.
Mar 30 21:02:16 Server2 haproxy[2101]: Proxy http-in started.
Mar 30 21:02:16 Server2 haproxy[2101]: Proxy servers started.
Mar 30 21:02:16 Server2 haproxy[2101]: Proxy servers started.
Mar 30 21:02:16 Server2 systemd[1]: Started HAProxy Load Balancer.
azureuser@Server2:~$
```

### **Edit the HAProxy configuration file Command :**

`sudo nano /etc/haproxy/haproxy.cfg`

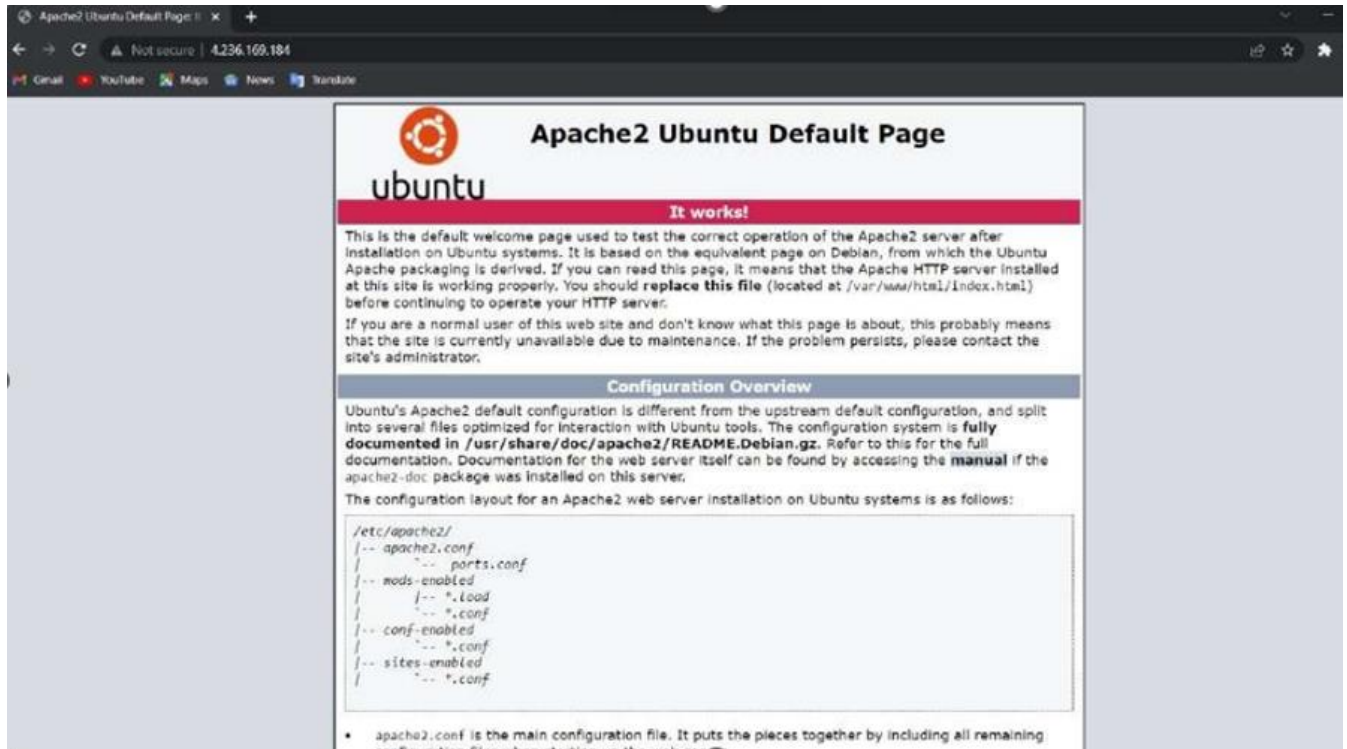
Add the following lines in the configuration file frontend :

```
frontend http-in
    bind *:80
    mode http
    default_backend servers
backend servers
    mode http
    balance roundrobin
    option httpchk GET /index.html
    server web1 10.0.0.6:80 check
    server web2 10.0.0.7:80 check
```

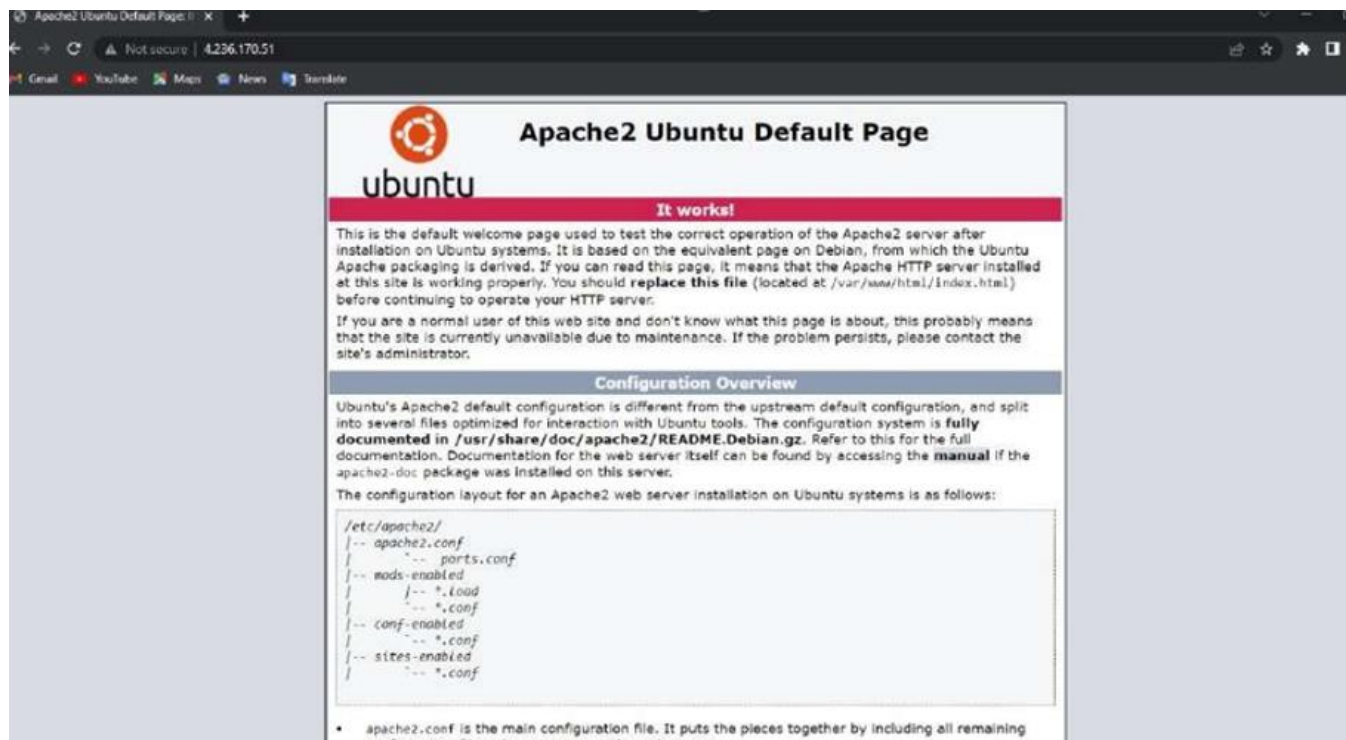




Now we open Apache using Public IP of Server1:



Screenshot of Apache using Public IP of Server2:

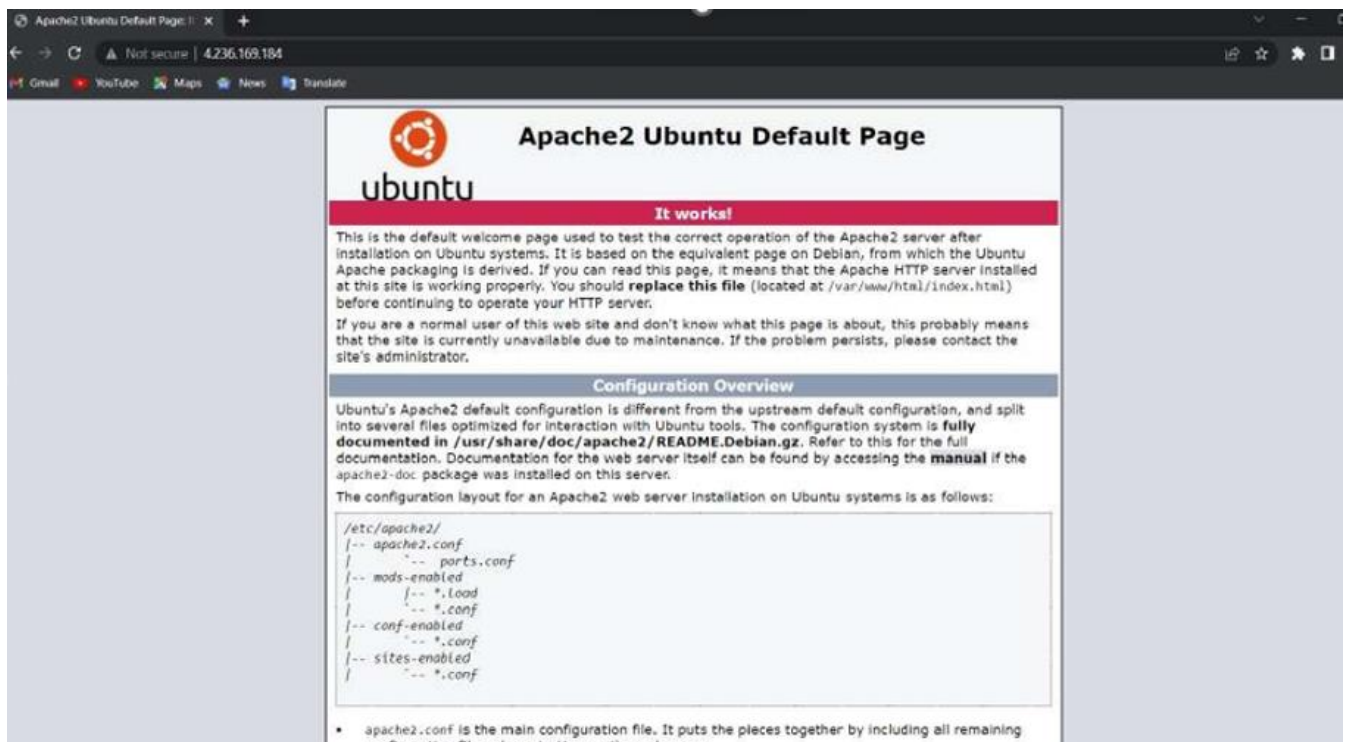


Now, we will stop the apache 2

```
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.
```

```
azureuser@Server1:~$ sudo systemctl stop apache2
```

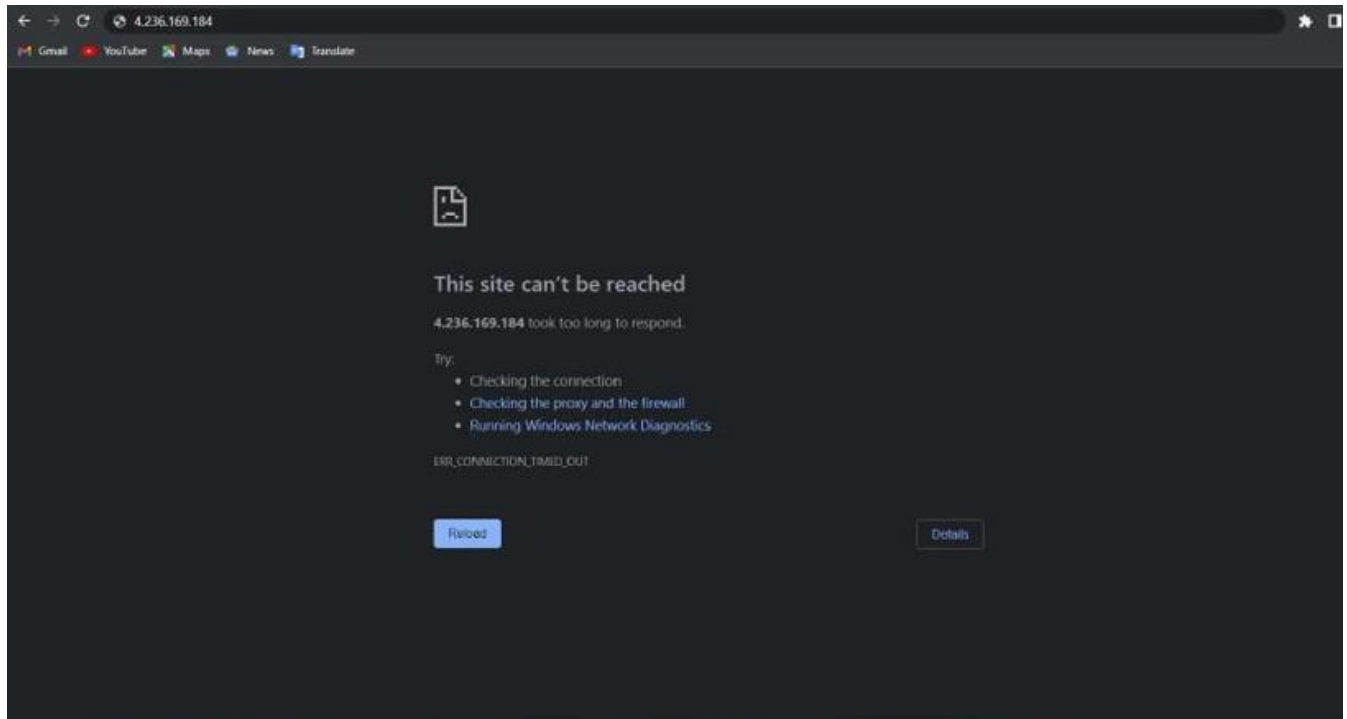
Now , I will open the IP of Server1 (whose apache has been stopped) in my chrome browser.



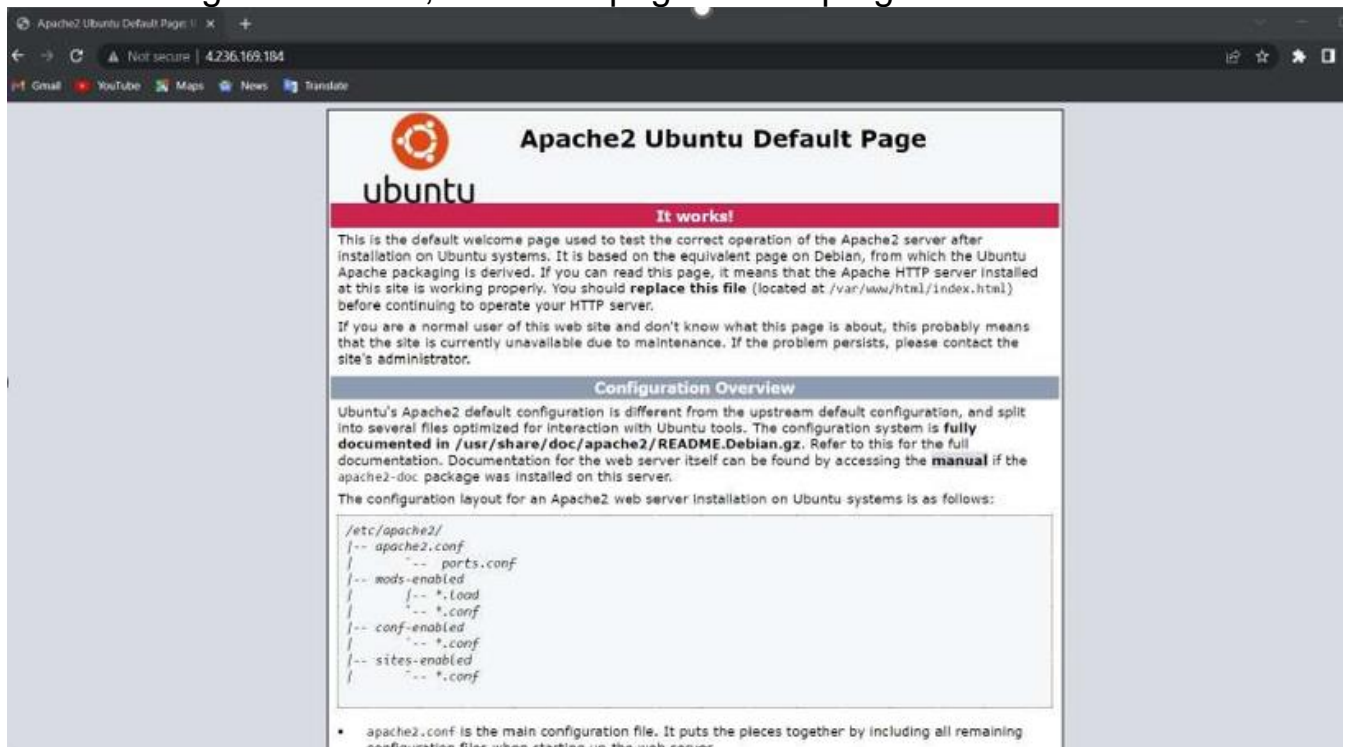
Now, we will stop apache in Server2 as well:

```
azureuser@Server2:~$ sudo systemctl stop apache2  
azureuser@Server2:~$
```

So, once the Server2 is stopped apache won't work



On restarting the Server1, the home page loads up again :

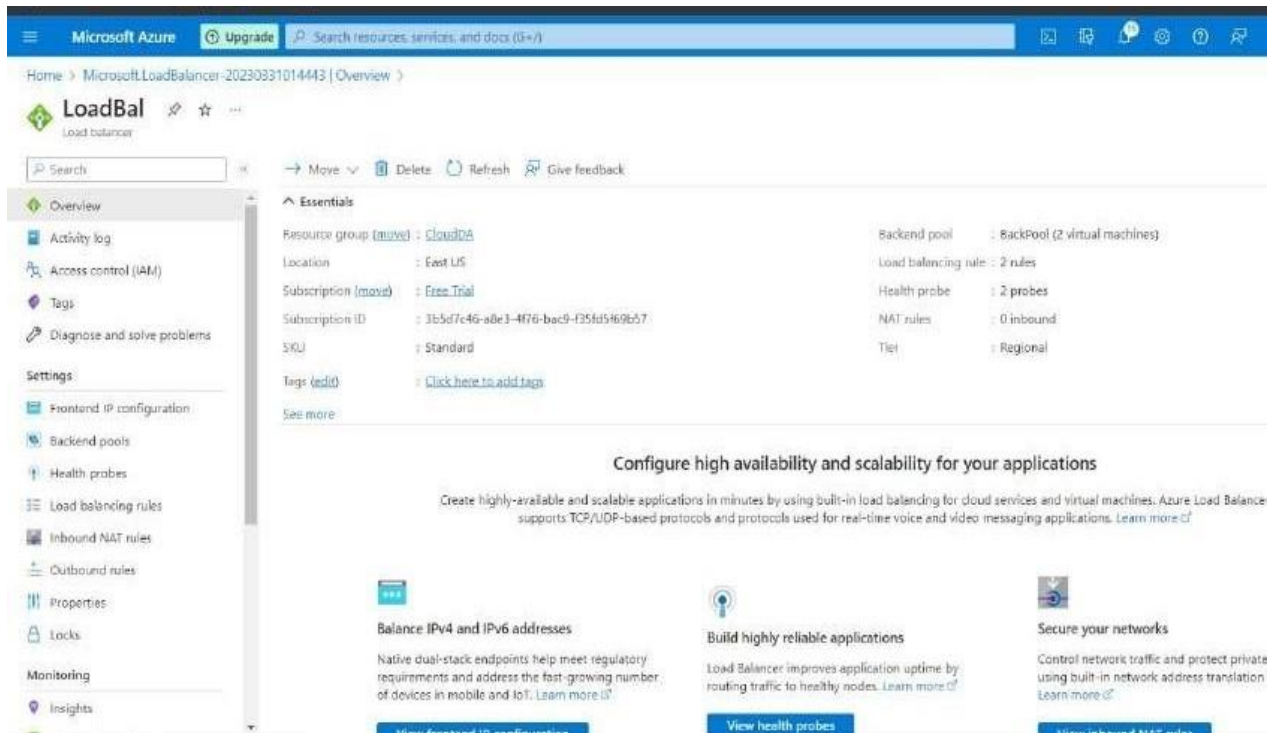


Hence our server failure over mechanism works.

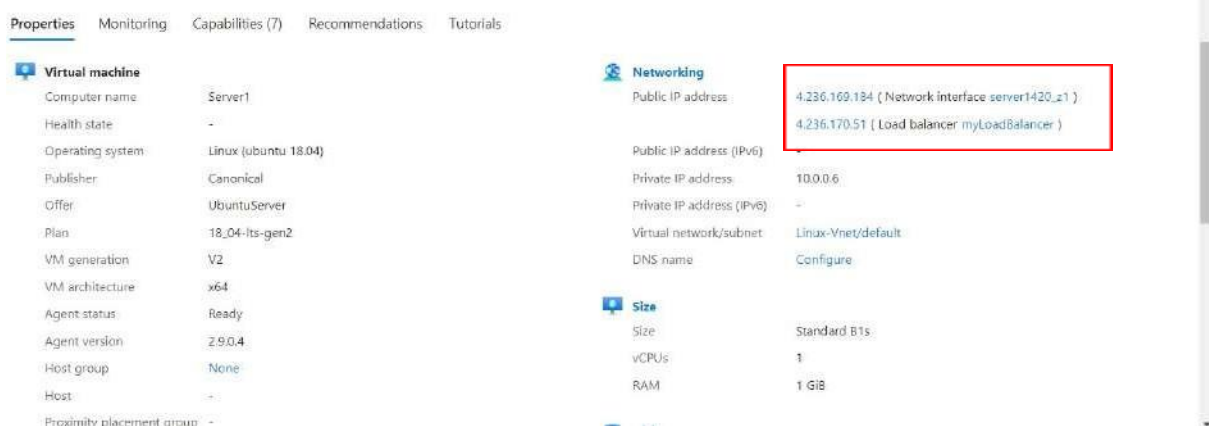
## Method 2: Using Load Balancing

Now, We will also see one more method of LoadBalancing

We have created a Load Balancer in the Azure:



In the Load Balancer **LoadBal** , I have used IP of server 2 by disassociating its Public IP from its Network Interface and that Public IP which is **4.236.170.51**.

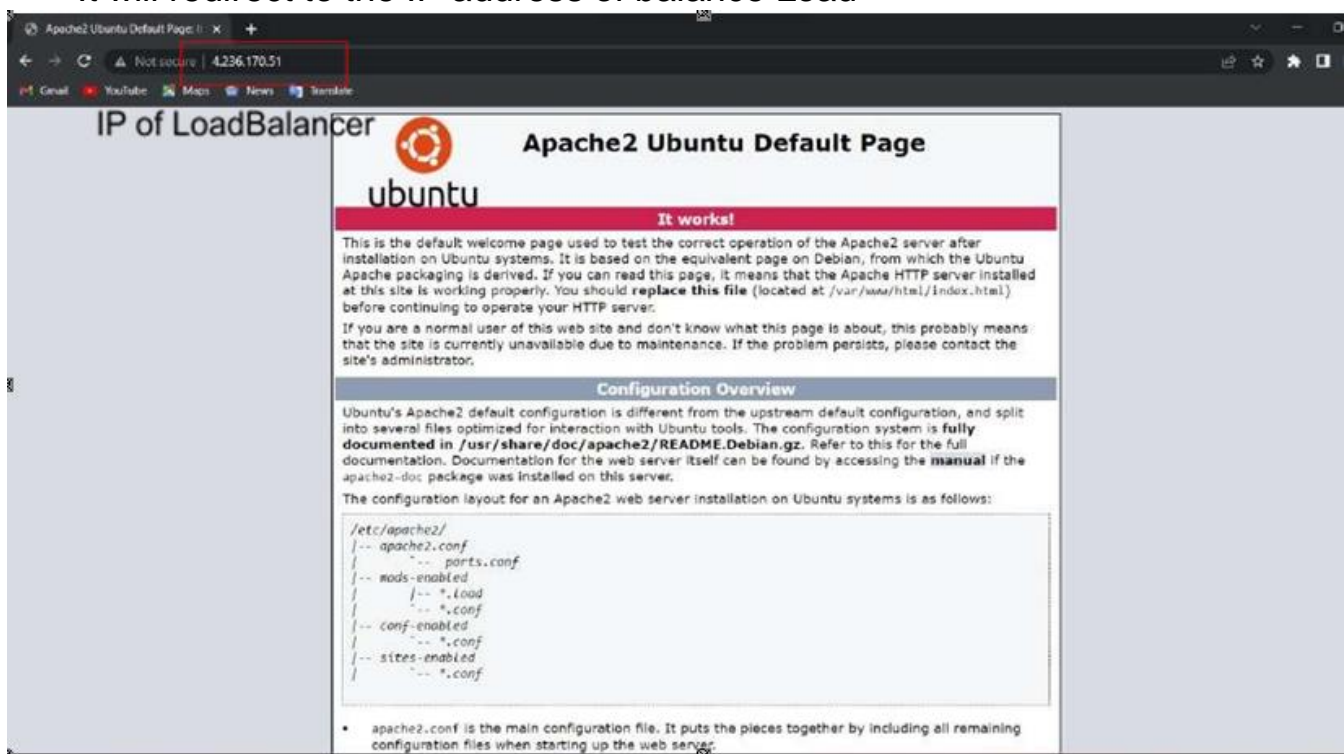


As shown above in the networking we have attached two IPs to the Server1. So when the First IP won't work it will redirect it to the LoadBalancer IP.

Now, we will shutdown Server1.

```
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
azureuser@Server1:~$ sudo shutdown now
```

It will redirect to the IP address of balance Load



Hence in this method also Failover mechanism works.