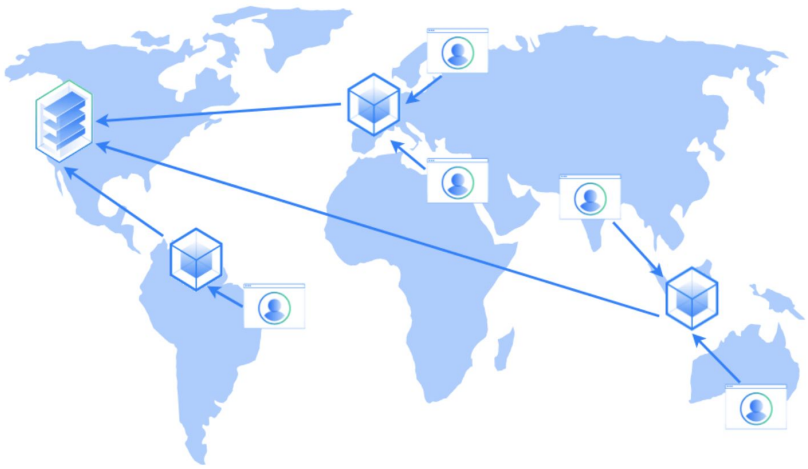
NC STATE UNIVERSITY

CONTENT DELIVERY NETWORK AS A SERVICE

Linux Networking - EcE 792



Professor: Anand Singh
EcE792
Fall 2019

Team -5
Ashish Sadanand Rajpurohit
Kashish Singh
Rishabh Katta
Sathwik Kalvakuntla

Contents:

1. Introduction
    - 1.1 Project Description
    - 1.2 Architecture overview
    - 1.3 Key components
2. Related work
3. Project Description
    - 3.1 Objective
    - 3.2 VPC architecture
    - 3.3 Features
        - 3.3.1 Functional Features
        - 3.3.2 Management Features
        - 3.3.3 K8's inspired Feature
4. Deployment and Functional Architecture
5. CDN architecture using Virtual Machines
6. CDN architecture using Containers
7. Kubernetes inspired features
8. Executional setup (codes and output snaps)
    - 8.1 In a Virtual Machine Environment
        - i. Creation of VPC
        - ii. Creation of Subnets
        - iii. Creation of VM instances
        - iv. Functional Features
        - v. Management Features
    - 8.2 In Container Environment
        - i. Creation of VPC
        - ii. Creation of Subnet
        - iii. Creation of VM instances
        - iv. Creation of Containerized instances
        - v. Functional Features
        - vi. Management Features
9. Future Scope
10. References

1. Introduction:

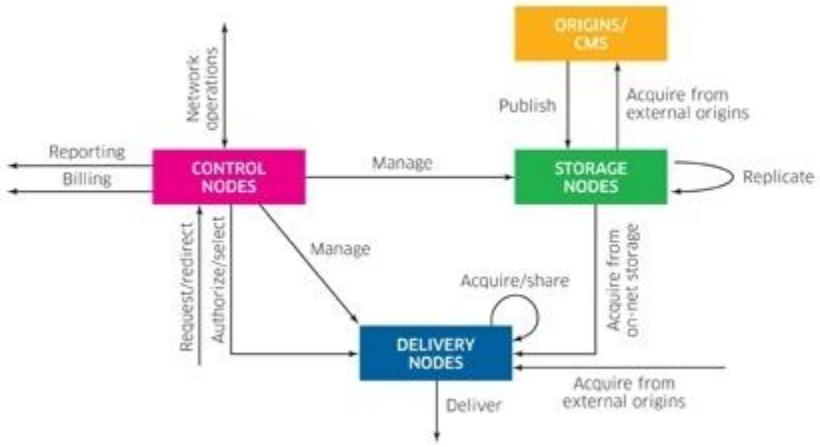
1.1 *Project Description:*

Implementing a Content Distribution Network using a Distributed group of servers that work together to provide faster delivery of content based on the requirement of the client.

The content is replicated and stored throughout the CDN so the user can access the data that is stored at a location that is geographically closest to the user. This is different (and more efficient) than the traditional method of storing content on just one, central server.

This solution focuses on improving the performance aspect among the Cloud Service Attributes. This service has been designed with the intention to take smart decisions governed by a set of predefined parameters. The idea is to leverage the information available in the form of performance metrics and maximize the performance of the complete cloud infrastructure. The main aspect of the project is that content is replicated and stored throughout the CDN so the user can access the data that is stored at a location that is geographically closest to the user. This project specifies the high usage implementation of the VM replication, Data replication, and Data hierarchy.

1.2 *Architecture overview:*



Most CDN architectures are constructed from a number of key components as seen in the above architectural diagram:

- *Delivery nodes*:

  The primary purpose is the delivery of data to consumers. It contains caches each running one or more delivery applications; these tend to be deployed as close to the edge (near the consumers) as possible.

- *Storage nodes:*

  The primary purpose is providing data to caches. These can be deployed in a hierarchical model to allow tiered caching and protection to any origin servers. These nodes can also be used where prepublishing of content is required rather than the content being acquired on-demand from origin servers.

- *Origin nodes:*

  These are the master sources for content and can be deployed within the CSP's network or more commonly within a content owner's infrastructure. A number of origins will be provided for scale and resilience.

- *Control node:*

  The primary purpose is to host the management, routing and monitoring components of a CDN. This is typically the integration point into any Operation Support Systems (OSS)/Business Support Systems (BSS) and Network Operations Centers (NOCs).

1.3 *Key Components:*

Content : The source(Ex: Youtube, Netflix) provides the digital information for distribution to the users.

Request: The user requests from the Content Provider to view or locally store data

Deliver: CDN delivers the content to the user

User: The entity requesting data (content) from the Content Provider

Client: The entity using the CDN service provided by us to provide content to his users

2. Related Work:

This service is inspired by some of the existing solutions in the industry. Companies such as Amazon, Akamai, and Google have developed their version of Content Delivery Networks.

Amazon Cloudfront:

Amazon CloudFront is a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency, high transfer speeds, all within a developer-friendly environment. CloudFront is integrated with AWS – both physical locations that are directly connected to the AWS global infrastructure, as well as other AWS services.

CloudFront's infrastructure has 117 PoPs of which 106 of them are smaller & localized PoPs and 11 are bigger, regional PoPs.

Google Cloud CDN:

Cloud CDN uses Google's global infrastructure (the same infrastructure that Google uses to deliver their end-user products like Google Search and Youtube) to cache and delivers content for their clients that are also users of the Google Cloud Platform (GCP).

GCP is a part of Google Cloud, which includes the entire public cloud infrastructure used by GCP, enterprise versions of Android and Chrome OS and much more.

Google Cloud CDN is available in 20 different regions, 61 zones and 134 different network edge locations for 200 countries and territories.

Microsoft Azure CDN:

Azure Content Delivery Network (CDN) is a global CDN solution for delivering high-bandwidth content. It can be hosted in Azure or any other location. With Azure CDN, you can cache static objects loaded from Azure Blob storage, a web application, or any publicly accessible web server, by using the closest point of presence (POP) server. Azure CDN can also accelerate dynamic content, which cannot be cached, by leveraging the various network and routing optimizations.

Akamai:

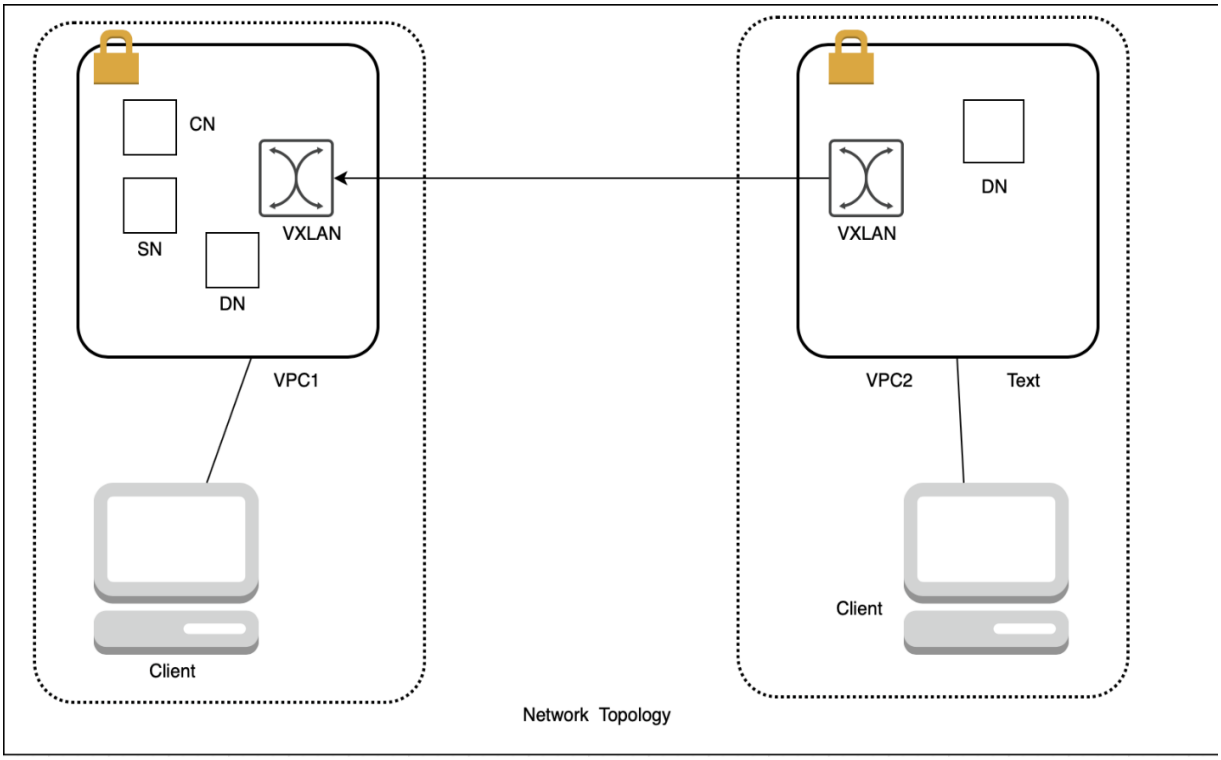
Akamai is the creator and operator of the world's most highly distributed CDN, serving 30% of all Internet traffic. With the broadest service portfolio in the industry, Akamai provides next-generation CDN solutions for operators everywhere. Akamai Aura Licensed CDN is a suite of operator CDN solutions that enable next-generation IP video services to deliver any device. Akamai's CDN infrastructure counts more than 240,000 servers located in more than 130 countries around the world. It is said that 85% of worldwide internet users are just one network jump away from one of Akamai's CDN servers. 2.5 exabytes of traffic go through Akamai's servers daily and Akamai approximately has interactions with 1.3 bn. user's devices daily as well.

3. Project Description

3.1 Objective

To provide the Content Delivery Network as a service to clients who wish to serve users belonging to different geographical regions. Our objective is to automate the VPC Infrastructure creation and the features we provide in our version of Content Delivery Network implementation. Also, provide the user with management and functional features.

3.2 VPC Architecture:



In this case of VPC based environment, the network infrastructure comprises of two hosts, host-1 and host-2. Host 1 represents the main VPC representing the node

architecture and the Host 2 represents the client architecture VPC. Each tenant is provided a Controller VM with a bridge in NAT/DHCP mode. The tenant is provided with the functionalities of creating his own topology using the input json/yml file. There are four key aspects in this overview topology: L3 Isolation using Namespaces, L2 Isolation using Bridges within a Tenant's subnets, VxLAN connectivity to provide tunneling across hypervisors and a management network for management activities.

➢ *L3 Isolation between Tenants:*

To provide L3 isolation for each tenant a dedicated namespace is created and all network bridges of the same tenant are connected to this namespace. The namespace also acts as a load-balancer for each tenant. This is accomplished using iptables in the namespace.

➢ *L2 Isolation between Subnets*:

To provide L2 isolation between subnets of the same tenant, a dedicated bridge is used for each subnet of the tenant.

➢ *VxLAN Connectivity*:

Vxlan connectivity across the hypervisors connected to the bridges that are connecting a controller VM to both the hypervisors.

3.3 Features:

3.3.1 Functional Features

- ***Data Hierarchy:***

  The Nodes are segregated on the basis of the specific function they implement and a hierarchical design is provided for the client to store data based on the client's implementation criteria; such as geographical-demand based storage.

  The Delivery node is closer to the users of the client and serves content the user requests either directly or by redirecting the client to fetch it from the Storage node.

  The Storage node is the main node and is at the top of the hierarchy which distributes files initially to the delivery nodes and later as per request from the delivery node.

  By implementing this feature, we solve the problem of wasting storage space by storing all files in all the nodes of the client.

- ***Data Replication:***

  The client's users' requests are expected to vary dynamically over time. This also creates a need to store the frequently requested data in the delivery node and remove the data which is not being used by the users from the delivery nodes for the efficient management of the storage space in the delivery nodes.

Logs are generated for all the user requests and based on these logs, the files which are frequently requested to a particular delivery node are transferred from the storage node to the particular delivery node.

By implementing this feature, we solve the problem of the Storage node serving many users for the same file when the file is not present at the delivery node. Also, we keep track of files that are not being used or are obsolete so that they can be deleted for efficient management of all the nodes.

- ***Node Replication:***

  When the requests to a particular node exceed a limit(which can be set by the client) a new delivery node is spawned which also starts serving the users and reduces the load on a single node.

  Implementing this feature solves the problem of a node crashing or being slow due to high user requests.

3.3.2 Management Features:

- ***High Availability***:

We have implemented the high availability feature in our project in two ways.

➢ *Stand-By Node:*

When a delivery node is created the client has the option of choosing to implement the high-availability feature. If chosen to implement it, a standby node is created along with the delivery node and it functions as a delivery node such that it isn't idle.

By this feature, if there is an unexpected failure of the Delivery node, then immediately the Standby node takes over and serves the users with the content requested.

By implementing this feature we solve the problem of reducing the down-times at the delivery node.

➢ *Self-healing:*

This is a Kubernetes inspired feature, which, if the client chooses to implement it, when the nodes get stopped unexpectedly, they immediately get spawned again.

- *Accounting*:

We have implemented the accounting management functionality by logging the number of times the system was accessed with the timestamp and saved it in CSV format, which can later be seen by the client for further processing and projection of estimated costs etc. The csv files specify the number of times a file was tried to be fetched from that specific Delivery nodes.

- *Configuration Management*

We implemented Configurational management in four key steps for our project:

➢ *Planning* - We are implementing configuration management plan details as to how a tenant will create, control, and audit configuration. This documentation is often a part of the project quality management plan.

➢ *Identification* - All the configuration requirements on a project are identified and recorded. That includes functionality requirements, design requirements, and any other specifications. The completion of this process results in the configuration baseline for the project.

➢ *Control* - As the project scope is altered, the impact on the configuration is documented in the form of the CSV file. This is normally done within the project change control process.

➢ *Status accounting* - We are tracking our project's container status at all times. We are able to tell what version your configuration is on, and when the containers are down using the Self_Healing.py script, the container is made active.

3.3.3 Kubernetes inspired feature:

- ***Self-healing*** (High availability and automatic recovery of containers):

We provide automatic recovery from arbitrary exits in our design by running a python script for each tenant's containers. This script executes every T seconds (10 by default) and checks the status of the executing containers. If it finds any of them in the stopped state, i.e. the user has accidentally exited from them, it starts those containers again.

- Inspired by Kubernetes - is a service that helps restart Containers when they fail the health check.
- This service monitors all the containers on all hypervisors to identify failed containers and respawns them on a free host.
- This service will detect any container that does not have a valid performance metric value. This will happen usually when the container is shut down or deleted.
- This will also respawn all containers on an affected Host onto other Hosts. This is because when the Host is down, the Containers will not have a valid performance metric value and will be respawned.

Demo:

```
ece792@t11-vm5:~/Container$ cat CID.txt
2fd5a434e776
ece792@t11-vm5:~/Container$ sudo docker container ls
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
[sudo] password for ece792:
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS                    NAMES
8145d5380721        mycentos               "/entrypoint /usr/sb…"   30 hours ago        Up 30 hours                                  DNRep
50b90d6c5bb7        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 23 hours                                  DN1
b8eab9250383        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  SN
4519a0862a05        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DN2
5767d902c238        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DNSB
2fd5a434e776        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  CN
2b225eb18c69        5e35e350aded           "/bin/bash"              44 hours ago        Up 44 hours                                  c1
f4d2df1a9027        k8s.gcr.io/pause:3.1   "/pause"                 2 days ago          Up 2 days                                    k8s_POD_etcd-t11-vm5_ku
efaf2d62df7d_3
03d0f1a9052e        ubnare/centos-with-ssh "/bin/sh -c 'bash /d…"   2 days ago          Up 2 days           0.0.0.0:9090->22/tcp     Kashish
7359d3c8cedd        k8s.gcr.io/pause:3.1   "/pause"                 2 days ago          Up 2 days                                    k8s_POD_kube-scheduler-
1d60d131353157588ab020_2
6346c0f72cf4        k8s.gcr.io/pause:3.1   "/pause"                 2 days ago          Up 2 days                                    k8s_POD_kube-apiserver-
a6e1ce435d78825d10cf2c_1
bf43d9cc8e5f        k8s.gcr.io/pause:3.1   "/pause"                 2 days ago          Up 2 days                                    k8s_POD_kube-controller
71e8d4ae0c74918307381f166eaac16_1
ece792@t11-vm5:~/Container$ sudo docker stop ^C
ece792@t11-vm5:~/Container$ sudo docker stop 2fd5a434e776
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
2fd5a434e776
ece792@t11-vm5:~/Container$ sudo docker container ls
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS                    NAMES
8145d5380721        mycentos               "/entrypoint /usr/sb…"   30 hours ago        Up 30 hours                                  DNRep
50b90d6c5bb7        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 23 hours                                  DN1
b8eab9250383        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  SN
4519a0862a05        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DN2
5767d902c238        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DNSB
2b225eb18c69        5e35e350aded           "/bin/bash"              44 hours ago        Up 44 hours                                  c1
```

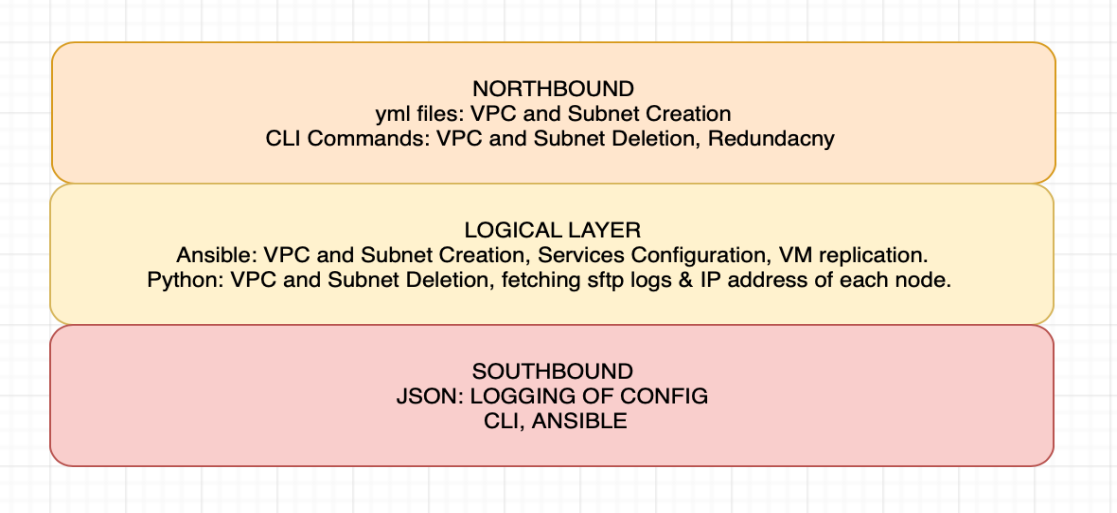
Self-healing of stopped container CN using SelfHealing.py script:

```
ece792@t11-vm5:~/Container$ sudo python SelfHealing.py CID.txt
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
The Container is back in action  with id  2fd5a434e776dd48f6a796da65a838d3fa6b5a789b6519fc1bdaadcfac34bfb5
^CTraceback (most recent call last):
  File "SelfHealing.py", line 17, in <module>
    sleep( 10 )
KeyboardInterrupt
ece792@t11-vm5:~/Container$ sudo docker container ls
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS                    NAMES
8145d5380721        mycentos               "/entrypoint /usr/sb…"   30 hours ago        Up 30 hours                                  DNRep
50b90d6c5bb7        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 23 hours                                  DN1
b8eab9250383        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  SN
4519a0862a05        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DN2
5767d902c238        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DNSB
2fd5a434e776        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 31 seconds                                CN
2b225eb18c69        5e35e350aded           "/bin/bash"              44 hours ago        Up 44 hours                                  c1
```

4. Deployment and Functional Architecture:

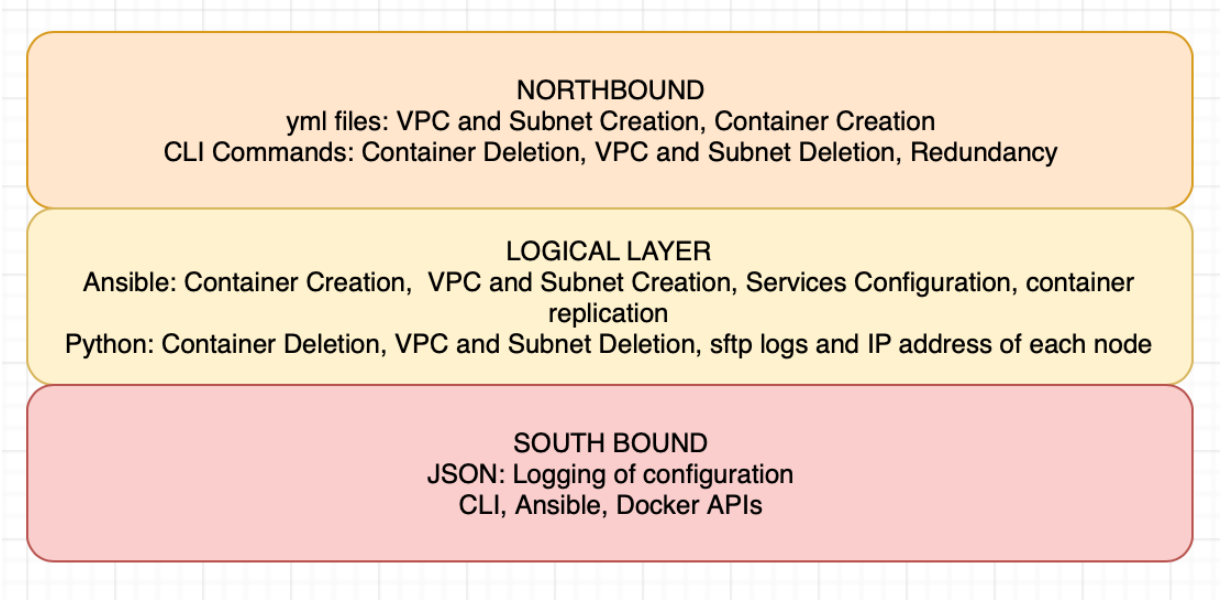
Northbound, Southbound and Logic Layer Implementation

- ***Functional Diagram of the VM Architecture:***



- ***Northbound:*** Input Collection Unit
  - ➢ Client uploads yml files with VPC, subnet, VM server instance creation files.
  - ➢ In case of deletion, user executes commands on CLI interface provided in the solution
- ***Logical Layer:*** Execution Unit. We have the following logical layers in the solution to take care of the tasks mentioned below:
  - ➢ ***Ansible:*** Playbooks are updated with variables with the help of user input and executed for the creation of VPC, containers, subnets and configuring services in the hypervisor.
  - ➢ ***Python:*** Scripts are being executed to provide container deletion and sftp logging, IP addresses of each node, deletion of subnets, deletion of VPCs and VM's.
- ***Southbound:*** Logging and CLI Output

➢ *CLI:* It shows the output of the scripts being run showing the status of the provisioning of the infrastructure and services.

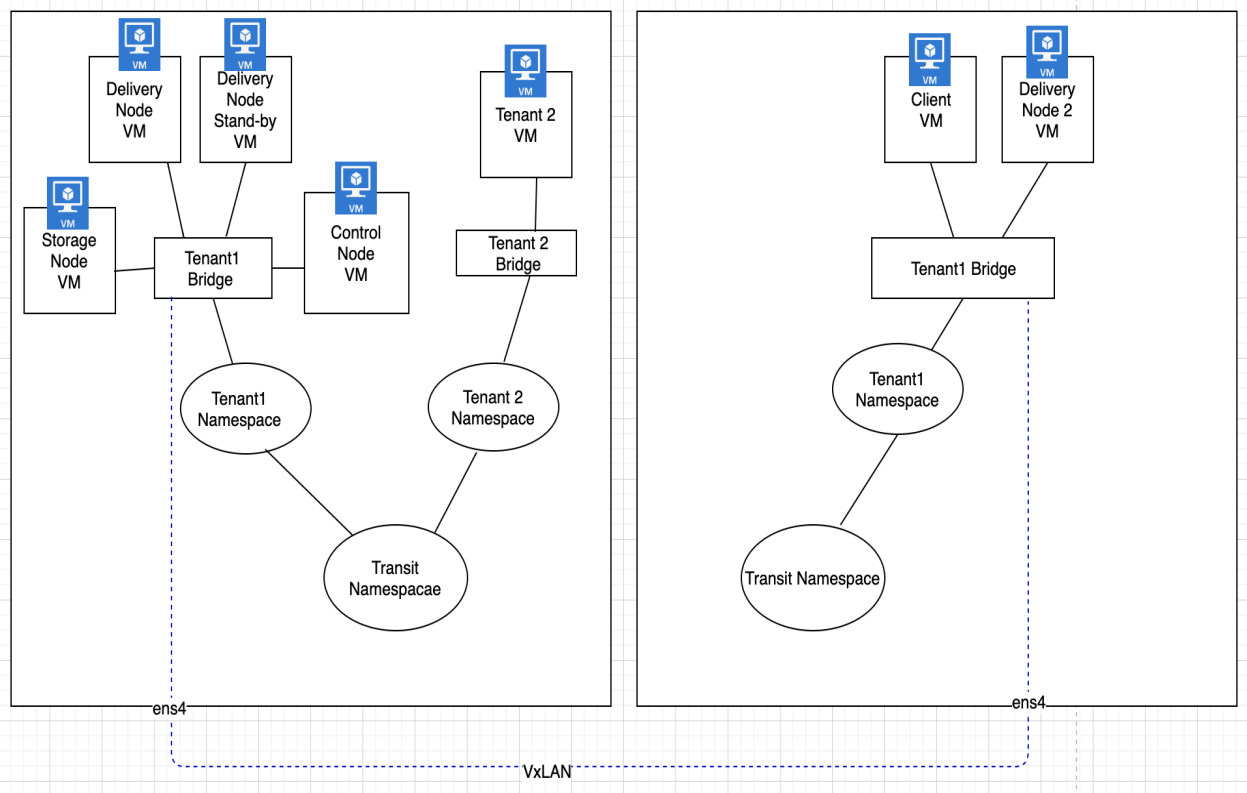


- *Functional Diagram of the Container Architecture:*

1. *Northbound:* Input Collection Unit
   - ➢ Client uploads yml files with VPC, subnet, Containerized server instance creation files.
   - ➢ In case of deletion, user executes commands on CLI interface provided in the solution
2. *Logical Layer:* Execution Unit. We have following logical layers in the solution to take care of tasks mentioned below:
   - ➢ *Ansible:* Playbooks are updated with variables with the help of user input and executed for the creation of VPC, containers, subnets and configuring services in the hypervisor.
   - ➢ *Python:* Scripts are being executed to provide container deletion and sftp logging, IP addresses of each node, deletion of subnets, deletion of VPCs and containers.
3. *Southbound:* Logging and CLI Output

➢ *CLI:* It shows the output of the scripts being run showing the status of the provisioning of the infrastructure and services.

5. CDN Architecture using Virtual Machines:



Components:

Storage Node VM: This VM is the node at which the customer can place all his contents that needs to be served among different clients. The customer who is using this CDN solution can place his data here either through wget command or using and cloud storage applications like Google Drive, S3, etc.

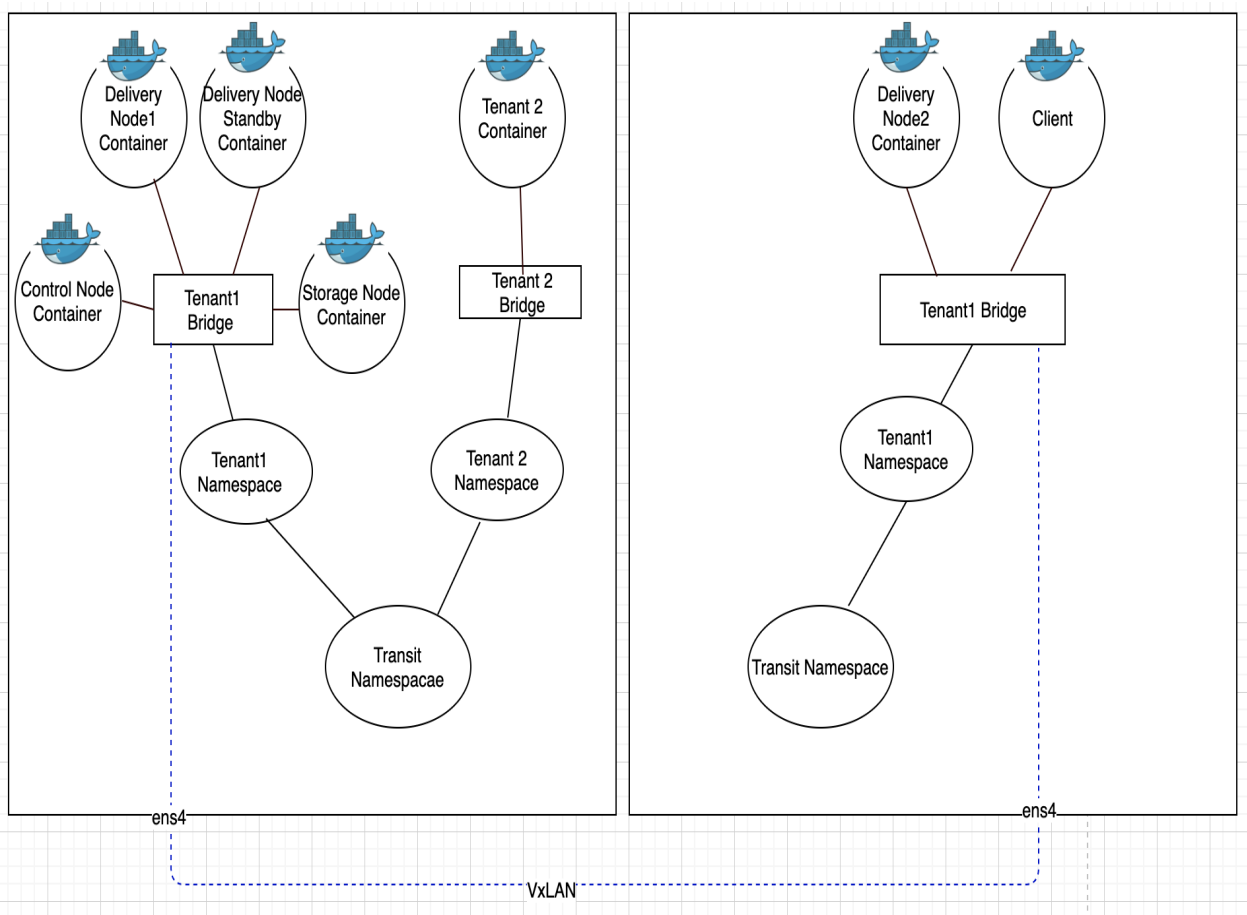
Delivery Node VM: This VM is the node that has the frequently accessed files(by the client). The customer has the functionality of placing the required files in the delivery node initially and then the contents of this node get changed dynamically based on the failed requests(which indirectly lets the customer know the files that are being frequently accessed - popular shows like Stranger things! (Netflix)).

Delivery Node Standby VM: This VM acts as an extra VM by which high availability is achieved which serves the content too to the customers.

Control Node VM: This VM acts as a master to this CDN architecture which decides about the files that need to be transferred from the storage to the delivery node(Data replication) and handle replication of VM's when the load on the delivery node that is serving the content is more.

Client VM: The VM that fetches the data by makes a request to the CDN architecture to binge-watch content. (only during semester-end holidays!)

6. CDN Architecture using Containers:



Components:

Storage Node Container: This Container is the node at which the customer can place all his contents that needs to be served among different clients. The customer who is using this CDN solution can place his data here either through wget command or using and cloud storage applications like Google Drive, S3, etc.

Delivery Node Container: This Container is the node that has the frequently accessed files(by the client). The customer has the functionality of placing the required files in the delivery node initially and then the contents of this node get changed dynamically based on the failed requests(which indirectly lets the customer know the files that are being frequently accessed - popular shows like Stranger things! (Netflix)).

Delivery Node Standby Container: This Container acts as an extra Container by which high availability is achieved which serves the content too to the customers.

Control Node Container: This Container acts as a master to this CDN architecture which decides about the files that need to be transferred from the storage to the delivery node(Data replication) and handle replication of VM's when the load on the delivery node that is serving the content is more.

Client Container: The Container that fetches the data by makes a request to the CDN architecture to binge-watch content. (only during semester-end holidays!)

7. Kubernetes inspired feature:

- ***Self-healing*** (High availability and automatic recovery of containers):

We provide automatic recovery from arbitrary exits in our design by running a python script for each tenant's containers. This script executes every T seconds (10 by default) and checks the status of the executing containers. If it finds any of them in the stopped state, i.e. the user has accidentally exited from them, it starts those containers again.

- Inspired by Kubernetes - is a service that helps restart Containers when they fail the health check.
- This service monitors all the containers on all hypervisors to identify failed containers and respawns them on a free host.
- This service will detect any container that does not have a valid performance metric value. This will happen usually when the container is shut down or deleted.
- This will also respawn all containers on an affected Host onto other Hosts. This is because when the Host is down, the Containers will not have a valid performance metric value and will be respawned.

Demo:

```
ece792@t11-vm5:~/Container$ cat CID.txt
2fd5a434e776
ece792@t11-vm5:~/Container$ sudo docker container ls
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
[sudo] password for ece792:
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS                    NAMES
8145d5380721        mycentos               "/entrypoint /usr/sb…"   30 hours ago        Up 30 hours                                  DNRep
50b90d6c5bb7        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 23 hours                                  DN1
b8eab9250383        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  SN
4519a0862a05        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DN2
5767d902c238        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DNSB
2fd5a434e776        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  CN
2b225eb18c69        5e35e350aded           "/bin/bash"              44 hours ago        Up 44 hours                                  c1
f4d2df1a9027        k8s.gcr.io/pause:3.1   "/pause"                 2 days ago          Up 2 days                                    k8s_POD_etcd-t11-vm5_ku
efaf2d62df7d_3
03d0f1a9052e        ubnare/centos-with-ssh "/bin/sh -c 'bash /d…"   2 days ago          Up 2 days           0.0.0.0:9090->22/tcp     Kashish
7359d3c8cedd        k8s.gcr.io/pause:3.1   "/pause"                 2 days ago          Up 2 days                                    k8s_POD_kube-scheduler-
1d60d131353157588ab020_2
6346c0f72cf4        k8s.gcr.io/pause:3.1   "/pause"                 2 days ago          Up 2 days                                    k8s_POD_kube-apiserver-
a6e1ce435d78825d10cf2c_1
bf43d9cc8e5f        k8s.gcr.io/pause:3.1   "/pause"                 2 days ago          Up 2 days                                    k8s_POD_kube-controller
71e8d4ae0c74918307381f166eaac16_1
ece792@t11-vm5:~/Container$ sudo docker stop ^C
ece792@t11-vm5:~/Container$ sudo docker stop 2fd5a434e776
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
2fd5a434e776
ece792@t11-vm5:~/Container$ sudo docker container ls
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS                    NAMES
8145d5380721        mycentos               "/entrypoint /usr/sb…"   30 hours ago        Up 30 hours                                  DNRep
50b90d6c5bb7        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 23 hours                                  DN1
b8eab9250383        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  SN
4519a0862a05        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DN2
5767d902c238        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DNSB
2b225eb18c69        5e35e350aded           "/bin/bash"              44 hours ago        Up 44 hours                                  c1
```

Self-healing of stopped container CN using SelfHealing.py script:

```
ece792@t11-vm5:~/Container$ sudo python SelfHealing.py CID.txt
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
The Container is back in action  with id  2fd5a434e776dd48f6a796da65a838d3fa6b5a789b6519fc1bdaadcfac34bfb5
^CTraceback (most recent call last):
  File "SelfHealing.py", line 17, in <module>
    sleep( 10 )
KeyboardInterrupt
ece792@t11-vm5:~/Container$ sudo docker container ls
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS                    NAMES
8145d5380721        mycentos               "/entrypoint /usr/sb…"   30 hours ago        Up 30 hours                                  DNRep
50b90d6c5bb7        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 23 hours                                  DN1
b8eab9250383        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  SN
4519a0862a05        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DN2
5767d902c238        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 43 hours                                  DNSB
2fd5a434e776        mycentos               "/entrypoint /usr/sb…"   43 hours ago        Up 31 seconds                                CN
2b225eb18c69        5e35e350aded           "/bin/bash"              44 hours ago        Up 44 hours                                  c1
```

8. Executional Setup:

8.1 In Virtual Machine Environment

- *VM REPLICATION:*

Based on the number of hits that each delivery nodes receives, The control node fetches the sftp logs that are continuously run after every 5 minutes using a defined cron job. Then if the number of hits crosses a specific limit of threshold a new VM gets generated with a static defined IP from that specifically defined IP subnet.
The working scenario is as explained below:

At TVM1(Delivery Node):

/var/log/Count.txt – 0 (populated by h.sh) running on a cron job after every 5 min DN1 hits count:

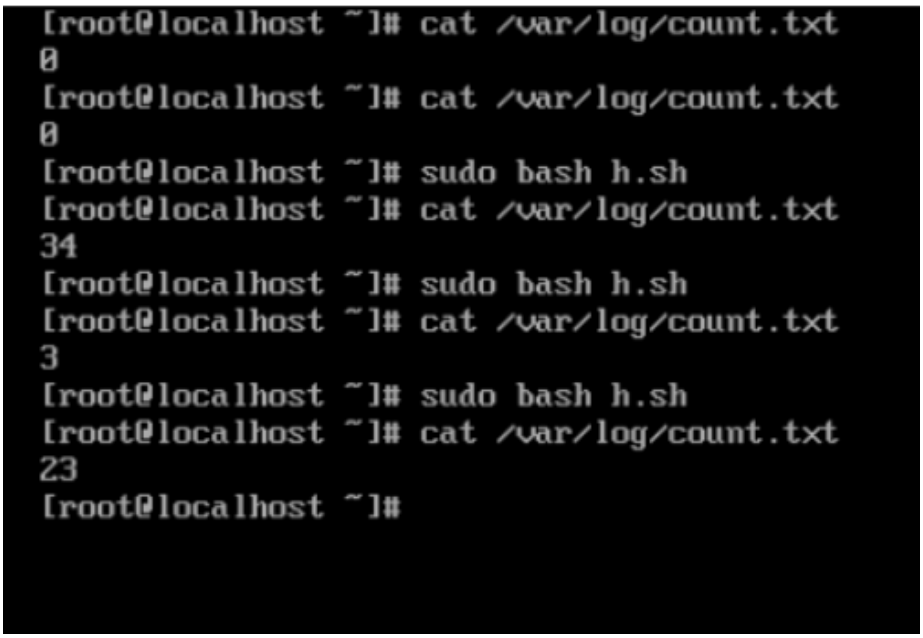


Fig: Snapshot at the storage node specifying the total hit counts for the received requests as above

- *Control Node Functionality:*

Run CN.sh (replicating to new VM based on number of count hits)

The CN.sh code that runs on the cron job after every 5 min collects the value of count.txt from the DN and then it does the VM replication-based in the specific threshold values as defined by the provider of the service.

- ➢ The VM gets replicated
- ➢ The VM receives a static IP defined by the provider from the same subnet.
- ➢ The sshpass file gets installed in the newly created VM
- ➢ The common files get transferred from the Storage node to the new Delivery node.
- ➢ The IP tables probabilistic IPtables gets configured at the NS connecting to both the VM's

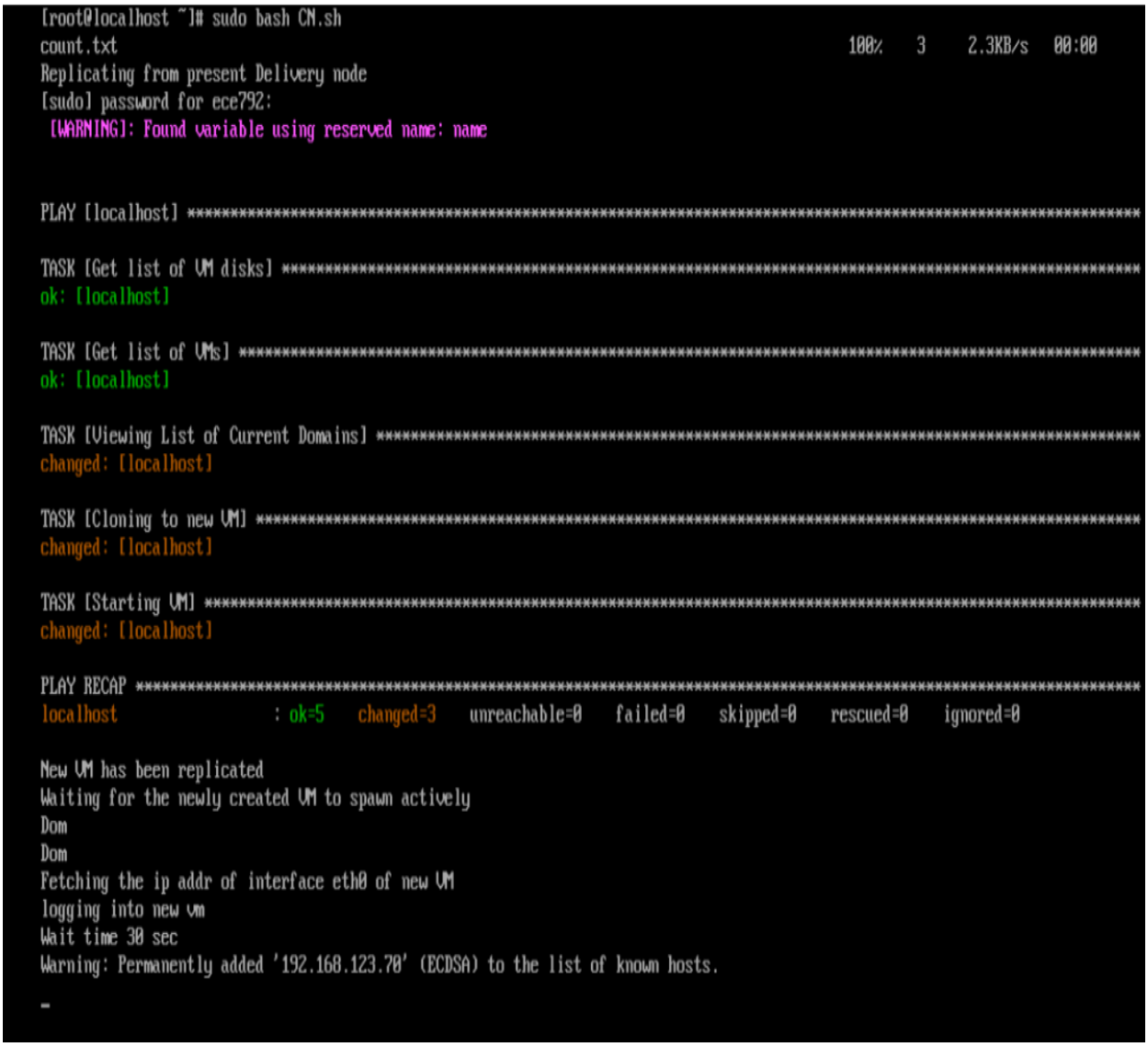


Fig: Snapshot at the CN that creates a new VM based on the number of requests thus providing load balancing

```
New VM has been replicated
Waiting for the newly created VM to spawn actively
Dom
Dom
Fetching the ip addr of interface eth0 of new VM
logging into new vm
Wait time 30 sec
Warning: Permanently added '192.168.123.70' (ECDSA) to the list of known hosts.
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 52:54:00:38:12:86 brd ff:ff:ff:ff:ff:ff
    inet 192.168.123.70/24 brd 192.168.123.255 scope global noprefixroute dynamic eth0
       valid_lft 3359sec preferred_lft 3359sec
    inet6 fe80::5e34:39b9:d2a7:f26c/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
    inet6 fe80::1820:5896:d4ef:db54/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 52:54:00:61:3c:bb brd ff:ff:ff:ff:ff:ff
    inet 120.0.0.22/24 scope global eth1
       valid_lft forever preferred_lft forever
Loaded plugins: fastestmirror
Determining fastest mirrors
 * base: packages.oit.ncsu.edu
 * extras: packages.oit.ncsu.edu
 * updates: packages.oit.ncsu.edu
```

Fig: Snapshot of the newly created VM IP address details

22 on QEMU/KVM@t11_vm5

```
[root@localhost ~]# ls
1.txt  2.mp4  3.txt  anaconda-ks.cfg
[root@localhost ~]#
```

Fig: Copy of the most requested files newly created VM:

```
[root@localhost ~]# sudo bash t.sh
Connected to 120.0.0.1.
Fetching /root/1.txt to ./1.txt
/root/1.txt                                                   100%   42    20.0KB/s   00:00
[root@localhost ~]# sudo bash t.sh
root@120.0.0.22's password:
Connected to 120.0.0.22.
Fetching /root/1.txt to ./1.txt
/root/1.txt                                                   100%   42    25.8KB/s   00:00
[root@localhost ~]#
```

Fig: Load balancing: (hitting old DN and newly created Delivery node separately to fetch data)

- ***Data hierarchy:***

File transfer from main Storage Node on the absence of the file from the local Delivery node. Since the availability of file is not there in the Storage node it gets fetched from the Delivery node: code bash t.sh(tvm3)

```
[root@localhost ~]# sudo bash t.sh
Connected to 120.0.0.1.
File "/root/1.txt" not found.
Connected to 120.0.0.5.
Fetching /root/1.txt to 1.txt
/root/1.txt                                                   100%   42    29.8KB/s   00:00
[root@localhost ~]# ls
1.txt  anaconda-ks.cfg  number.txt  py1.py  test.sh  t.sh  yoyo1.txt  yoyo.txt
[root@localhost ~]#
```

- *Data Replication:*

There are two main aspects that we will be considered during the development of Data Replication feature in our CDN:

Who should decide about the request redirection (location) and where should a request be directed for server selection:

➢ *Client-side redirection:* The approach that we develop has a set of physical replicas and chooses where to send the request, the below example shows the 2 Delivery nodes as the replicas but on the absence of it from both the DN it fetched from the storage nodes.

➢ *Server redirection:* The decision is taken based on the current workload exhibited by the Delivery nodes, which requires the Storage node to periodically sent their data to the Delivery node as shown in the 2nd snapshot.

Working – Control Node possess p.py that fetches sftplog file from the Delivery node and then based on the number of hits it received from the client it then sends that specific file from Storage node to Delivery node.
So until the file is not present in the Delivery node the client is basically transferred to download the data from the Storage node.

```
~ — root@localhost:/var/log — ssh -Y ece792@152.14.83.161   ~ — root@localhost:~ — ssh -Y ece792@152.14.83.161
[root@localhost ~]# vi t.sh
[root@localhost ~]# bash t.sh
Connected to 120.0.0.1.
File "/root/3.txt" not found.
Connected to 120.0.0.5.
Fetching /root/3.txt to 3.txt
/root/3.txt                                   100%   42    26.7KB/s   00:00
[root@localhost ~]# bash t.sh
Connected to 120.0.0.1.
File "/root/3.txt" not found.
Connected to 120.0.0.5.
Fetching /root/3.txt to 3.txt
/root/3.txt                                   100%   42    19.9KB/s   00:00
[root@localhost ~]# bash t.sh
Connected to 120.0.0.1.
File "/root/3.txt" not found.
Connected to 120.0.0.5.
Fetching /root/3.txt to 3.txt
/root/3.txt                                   100%   42    44.2KB/s   00:00
[root@localhost ~]# bash t.sh
Connected to 120.0.0.1.
File "/root/3.txt" not found.
Connected to 120.0.0.5.
Fetching /root/3.txt to 3.txt
/root/3.txt                                   100%   42    28.9KB/s   00:00
[root@localhost ~]# bash t.sh
Connected to 120.0.0.1.
File "/root/3.txt" not found.
Connected to 120.0.0.5.
Fetching /root/3.txt to 3.txt
/root/3.txt                                   100%   42    36.8KB/s   00:00
[root@localhost ~]# bash t.sh
Connected to 120.0.0.1.
File "/root/3.txt" not found.
Connected to 120.0.0.5.
Fetching /root/3.txt to 3.txt
/root/3.txt                                   100%   42    31.5KB/s   00:00
[root@localhost ~]# bash t.sh
Connected to 120.0.0.1.
File "/root/3.txt" not found.
Connected to 120.0.0.5.
```

Fig: Snapshot at the Delivery node while fetching files from the Storage node that is not present in the DN.

```
CN on QEMU/KVM@t11_vm5
[root@localhost ~]# sudo python p.py
sftp.log                                      100%   15KB   4.3MB/s   00:00
0
6
18
root@120.0.0.5's password:
2.mp4                                         100% 8832KB  40.9MB/s   00:00
root@120.0.0.5's password:
3.txt                                         100%   42    10.7KB/s   00:00
[root@localhost ~]#
```

Fig: Snapshot at the Control node that transfers the file from the storage node to the Delivery node.

The two files i.e 2.mp4 and 3.txt that was not present in the delivery node and specifically present in the storage node was fetched by the delivery node from the SN and then the Control node based on the number of hits for those file in the sftp log of the Delivery node sent the file from the storage node to the delivery node so that the main reason of implementing the content delivery network will be in the function.

Fig: Snapshot at the delivery node when the 2 files not present in the DN got added by the storage nodes.

8.2 In Container Environment

Creating VPC, subnet, and Container

```
ece792@t_vm16:/etc/ansible$ sudo bash test9.sh

PLAY [localhost] ***************************************************************************************************************

TASK [debug] *******************************************************************************************************************
ok: [localhost] => {
    "msg": {
        "t10": {
            "endpoint_name": "test",
            "os_type": "centos",
            "server_name": "Recv",
            "subnet_name": "T10sub",
            "vpc_name": "T10"
        }
    }
}

TASK [Install required packages for docker] ************************************************************************************
ok: [localhost]

TASK [Create docker image if not exists] ***************************************************************************************
ok: [localhost] => (item=t10)
[WARNING]: Please specify build.path instead of path. The path option has been renamed and will be removed in Ansible 2.12.

[WARNING]: The value of the "source" option was determined to be "build". Please set the "source" option explicitly.
Autodetection will be removed in Ansible 2.12.

[DEPRECATION WARNING]: Param 'path' is deprecated. See the module docs for more information. This feature will be removed in
version 2.12. Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.

TASK [Create and start container for client server if already not present] *****************************************************
ok: [localhost] => (item=t10)

TASK [Connect Container with bridge] *******************************************************************************************
included: /etc/ansible/cplayb/server_instance_config.yml for localhost

TASK [Veth Pair Interface prefix] **********************************************************************************************
ok: [localhost]

TASK [Create veth pair for bridge and container] *******************************************************************************
fatal: [localhost]: FAILED! => {"changed": true, "cmd": ["ip", "link", "add", "Recvvif1", "type", "veth", "peer", "name", "Recvvif
2"], "delta": "0:00:00.007351", "end": "2019-12-07 00:39:55.511932", "msg": "non-zero return code", "rc": 2, "start": "2019-12-07

ece792@t_vm16:/etc/ansible$ sudo bash test9.sh

PLAY [localhost] ***************************************************************************************************************

TASK [debug] *******************************************************************************************************************
ok: [localhost] => {
    "msg": {
        "t10": {
            "endpoint_name": "test",
            "os_type": "centos",
            "server_name": "Recv",
            "subnet_name": "T10sub",
            "vpc_name": "T10"
        }
    }
}

TASK [Install required packages for docker] ************************************************************************************
ok: [localhost]

TASK [Create docker image if not exists] ***************************************************************************************
ok: [localhost] => (item=t10)
[WARNING]: Please specify build.path instead of path. The path option has been renamed and will be removed in Ansible 2.12.

[WARNING]: The value of the "source" option was determined to be "build". Please set the "source" option explicitly.
Autodetection will be removed in Ansible 2.12.

[DEPRECATION WARNING]: Param 'path' is deprecated. See the module docs for more information. This feature will be removed in
version 2.12. Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.

TASK [Create and start container for client server if already not present] *****************************************************
ok: [localhost] => (item=t10)

TASK [Connect Container with bridge] *******************************************************************************************
included: /etc/ansible/cplayb/server_instance_config.yml for localhost

TASK [Veth Pair Interface prefix] **********************************************************************************************
ok: [localhost]
```

```
TASK [Make the interface running] ***********************************************************************************************
changed: [localhost]

TASK [Run dhclient] *************************************************************************************************************
changed: [localhost]

TASK [Get the ip address assigned to DNS container] *****************************************************************************
changed: [localhost]

TASK [debug] ********************************************************************************************************************
ok: [localhost] => {
    "msg": "Container IP: "
}

TASK [debug] ********************************************************************************************************************
ok: [localhost] => {
    "msg": "docker exec -i Recv /sbin/ifconfig Recvvif1 | grep 'inet' | cut -d: -f2 | awk '{print $2}'"
}

TASK [Get the ip address assigned to DNS container] *****************************************************************************
changed: [localhost]

TASK [debug] ********************************************************************************************************************
ok: [localhost] => {
    "msg": "Container Subnet IP: "
}

PLAY RECAP **********************************************************************************************************************
localhost                  : ok=18   changed=9    unreachable=0    failed=0    skipped=0    rescued=0    ignored=3
```

```
TASK [debug] *******************************************************************************************************************
ok: [localhost] => {
    "msg": {
        "t1": {
            "os_type": "centos",
            "server_name": "DN1",
            "subnet_name": "T10sub",
            "vpc_name": "T10"
        },
        "t2": {
            "os_type": "centos",
            "server_name": "DN2",
            "subnet_name": "T10sub",
            "vpc_name": "T10"
        },
        "t3": {
            "os_type": "centos",
            "server_name": "SN",
            "subnet_name": "T10sub",
            "vpc_name": "T10"
        },
        "t4": {
            "os_type": "centos",
            "server_name": "CN",
            "subnet_name": "T10sub",
            "vpc_name": "T10"
        },
        "t5": {
            "os_type": "centos",
            "server_name": "DNSB",
            "subnet_name": "T10sub",
            "vpc_name": "T10"
        }
    }
}

TASK [Install required packages for docker] ************************************************************************************
 [WARNING]: Could not find aptitude. Using apt-get instead

ok: [localhost]

TASK [Create docker image if not exists] ***************************************************************************************
ok: [localhost] => (item=t4)
ok: [localhost] => (item=t5)
ok: [localhost] => (item=t2)
ok: [localhost] => (item=t3)
ok: [localhost] => (item=t1)
 [WARNING]: Please specify build.path instead of path. The path option has been renamed and will be removed in Ansible 2.12.

 [WARNING]: The value of the "source" option was determined to be "build". Please set the "source" option explicitly.
Autodetection will be removed in Ansible 2.12.


TASK [Create and start container for client server if already not present] *****************************************************
ok: [localhost] => (item=t4)
ok: [localhost] => (item=t5)
ok: [localhost] => (item=t2)
ok: [localhost] => (item=t3)
ok: [localhost] => (item=t1)

TASK [Connect Container with bridge] *******************************************************************************************
 [WARNING]: While constructing a mapping from /etc/ansible/cplayb/server_instance_config.yml, line 29, column 5, found a
duplicate dict key (shell). Using last defined value only.

included: /etc/ansible/cplayb/server_instance_config.yml for localhost
included: /etc/ansible/cplayb/server_instance_config.yml for localhost
included: /etc/ansible/cplayb/server_instance_config.yml for localhost
included: /etc/ansible/cplayb/server_instance_config.yml for localhost
included: /etc/ansible/cplayb/server_instance_config.yml for localhost

TASK [Veth Pair Interface prefix] **********************************************************************************************
ok: [localhost]
```

```
PLAY RECAP *********************************************************************************************************************
localhost                  : ok=15   changed=5    unreachable=0    failed=1    skipped=0    rescued=0    ignored=3

DN1
files read
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
Changing password for user root.
New password: BAD PASSWORD: The password is shorter than 8 characters
Retype new password: passwd: all authentication tokens updated successfully.
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
DN1 pass changed
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
Changing password for user root.
New password: BAD PASSWORD: The password is shorter than 8 characters
Retype new password: passwd: all authentication tokens updated successfully.
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
DN2 pass changed
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
Changing password for user root.
New password: BAD PASSWORD: The password is shorter than 8 characters
Retype new password: passwd: all authentication tokens updated successfully.
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
SN pass changed
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
Changing password for user root.
New password: BAD PASSWORD: The password is shorter than 8 characters
Retype new password: passwd: all authentication tokens updated successfully.
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
CN pass changed
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
Changing password for user root.
New password: BAD PASSWORD: The password is shorter than 8 characters
Retype new password: passwd: all authentication tokens updated successfully.
sudo: unable to resolve host t11-vm5: Resource temporarily unavailable
all done
Connection to 192.168.122.23 closed.
Changing password for user root.
New password: BAD PASSWORD: The password is shorter than 8 characters
Retype new password: passwd: all authentication tokens updated successfully.
```

CDN automation for container:

```
import os
import json
with open("cdnauto.json") as f:
        js=f.read()
data=json.loads(js)
print("Please let us know if you are patient enough to give the inputs through CLI or else populate the json file. 1.CLI 2.json fi
le")
CLIorjson=input()
if(CLIorjson=="1"):
        print("Do you want to implement CDN architecture in this topology? 1.Yes 2.No")
        CDNresponse=input()
        if(CDNresponse=="1"):
                print("Thank you for choosing CDN service! We hope that you like it!!")
                os.system("sudo bash opposite.sh")
                print("Do you want to implement Data hierarchy functionality? 1.Yes 2.No")
                CDNdatahierarchy=input()
                if(CDNdatahierarchy=="1"):
                        os.system("sudo bash CDNsend.sh")
                print("Do you want to implement our new node replication feature in your setup??? 1.Yes 2.No")
                CDNdatarep=input()
                if(CDNdatarep=="1"):
                        print("Cool! The architecture would now scale as per the demands!! Sit back and relax!!")
                print("Do you need Accounting functionality? 1.Yes 2.No")
                Acc=input()
                if(Acc=="yes"):
                        print("You can find all the log statistics in CN node under log folder!")
        else:
                print("Alas! Hope to see you next time!")
else:
        if(data["cdnauto"]=="yes"):
                print("Thank you for choosing CDN service! We hope that you like it!!")
                os.system("sudo bash opposite.sh")
        if(data["data hierarchy"]=="yes"):
                os.system("sudo bash CDNsend.sh")
        if(data["delivery node replication"]=="yes"):
                print("Cool! The architecture would now scale as per the demands!! Sit back and relax!!")
        if(data["accounting"]=="yes"):
                print("You can find all the log statistics in CN node under log folder!")
~
~
```

Data_Hierarchy:

```
echo Downloading
file=/home/local/DN1.txt
ip=$(cat "$file")
file1=/home/local/DN2.txt
ip1=$(cat "$file1")
file2=/home/local/SN.txt
ip2=$(cat "$file2")
sshpass -p serv sftp local@$ip:1.txt .
if [ $? -eq 1 ]
then
    sshpass -p serv sftp local@$ip1:1.txt .
    if [ $? -eq 1 ]
    then
        sshpass -p serv sftp local@$ip2:1.txt .
    fi
fi
```

Data Replication:

```
#!/bin/bash
#ssh -tt ece792@192.168.122.23 " sudo ansible-playbook /etc/ansible/cplayb/create_replication_instance.yml"

scp ece792@192.168.122.23:/home/ece792/Container/DNRep.txt .

file1=/root/DNRep.txt
dnip=$(cat "$file1")

file2=/root/SN.txt
snip=$(cat "$file2")

scp root@$snip:/root/1.txt root@$dnip:/root/
scp root@$snip:/root/2.txt root@$dnip:/root/
```

Container Replication:

```
import re
import os
import time
timestr = time.strftime("%Y%m%d-%H%M%S")
with open("sftp.log") as f2:
    contents = f2.read()
    count1 = sum(1 for match in re.finditer(r"\blstat\b", contents))
   # count2 = sum(1 for match in re.finditer(r"\b4.txt\b", contents))

if (count1>3):
        os.system("bash containerrep.sh")
```

Based on the number of hits that each delivery node receives, The control node fetches the sftp logs that are continuously run after every 5 minutes using a defined cron job. Then if the number of hits crosses a specific limit of threshold a new container gets generated with a static defined IP from that specifically defined IP subnet.
The working scenario is as explained below:

At DN1(Delivery Node):

/var/log/Count.txt – 0 (populated by h.sh) running on a cron job after every 5 min DN1 hits count:

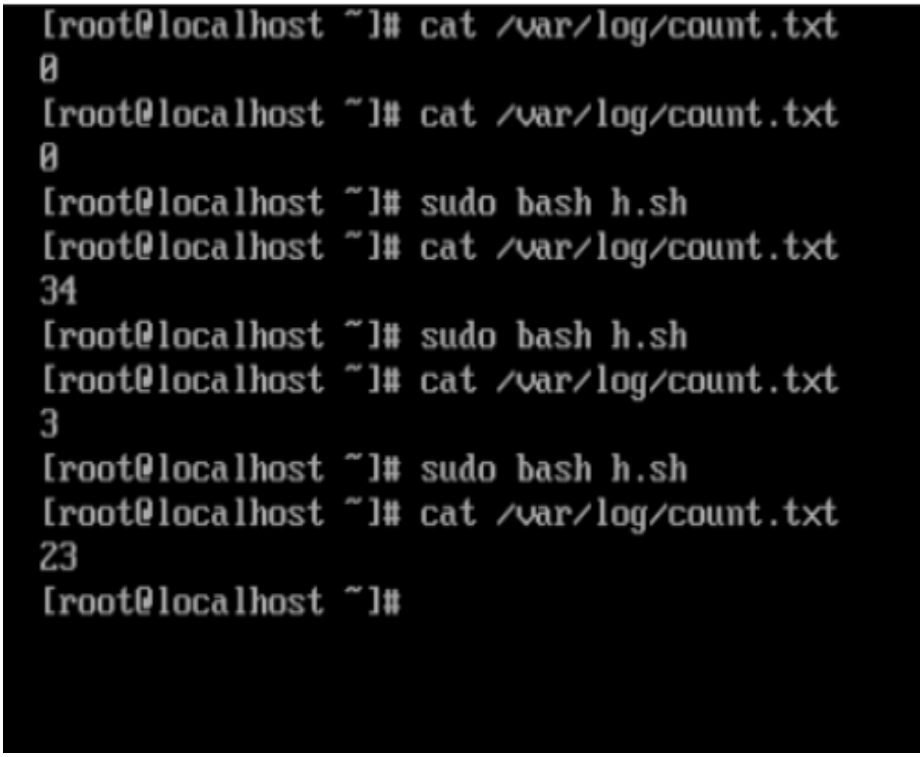


Fig: Snapshot at the storage node specifying the total hit counts for the received requests as above

- *Control Node Functionality:*

Run CNfetchDN1 (replicating to new Container based on number of count hits)

The CNfetchDN1 code that runs on the cron job after every 5 min collects the value of count.txt from the DN and then it does the Container replication-based in the specific threshold values as defined by the provider of the service.

- ➢ The Container gets replicated
- ➢ The Container receives a static IP defined by the provider from the same subnet.
- ➢ The sshpass file gets installed in the newly created VM
- ➢ The common files get transferred from the Storage node to the new Delivery node.

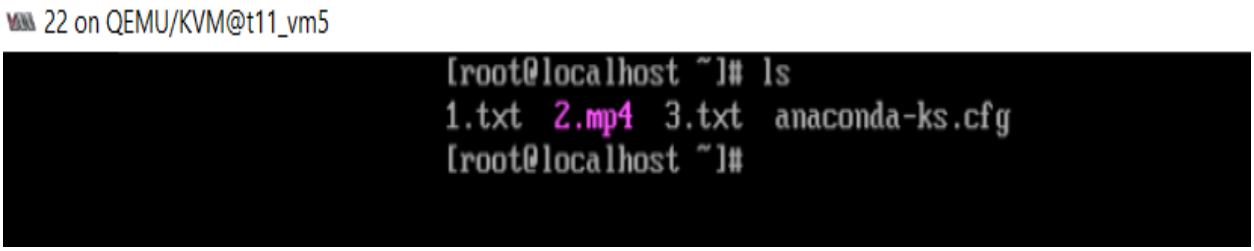
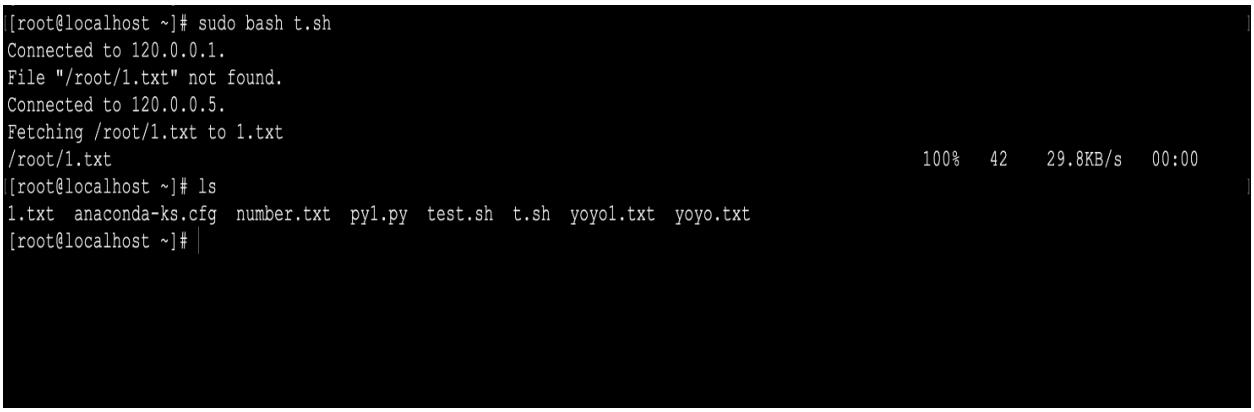


Fig: Copy of the most requested files newly created Container:

- *Data hierarchy:*

File transfer from main Storage Node on the absence of the file from the local Delivery node. Since the availability of file is not there in the Storage node it gets fetched from the delivery node.



- *Data Replication:*

There are two main aspects that we will be considered during the development of Data Replication feature in our CDN:

Who should decide about the request redirection (location) and where should a request be directed for server selection:

➢ *Client-side redirection:* The approach that we develop has a set of physical replicas and chooses where to send the request, the below example shows the 2 Delivery nodes as the replicas but on the absence of it from both the DN it fetched from the storage nodes.

➢ *Server redirection:* The decision is taken based on the current workload exhibited by the Delivery nodes, which requires the Storage node to periodically send their data to the Delivery node as shown in the 2nd snapshot.

Working – Control Node possess p.py that fetches sftplog file from the Delivery node and then based on the number of hits it received from the client it then sends that specific file from Storage node to Delivery node.
So until the file is not present in the Delivery node the client is basically transferred to download the data from the Storage node.

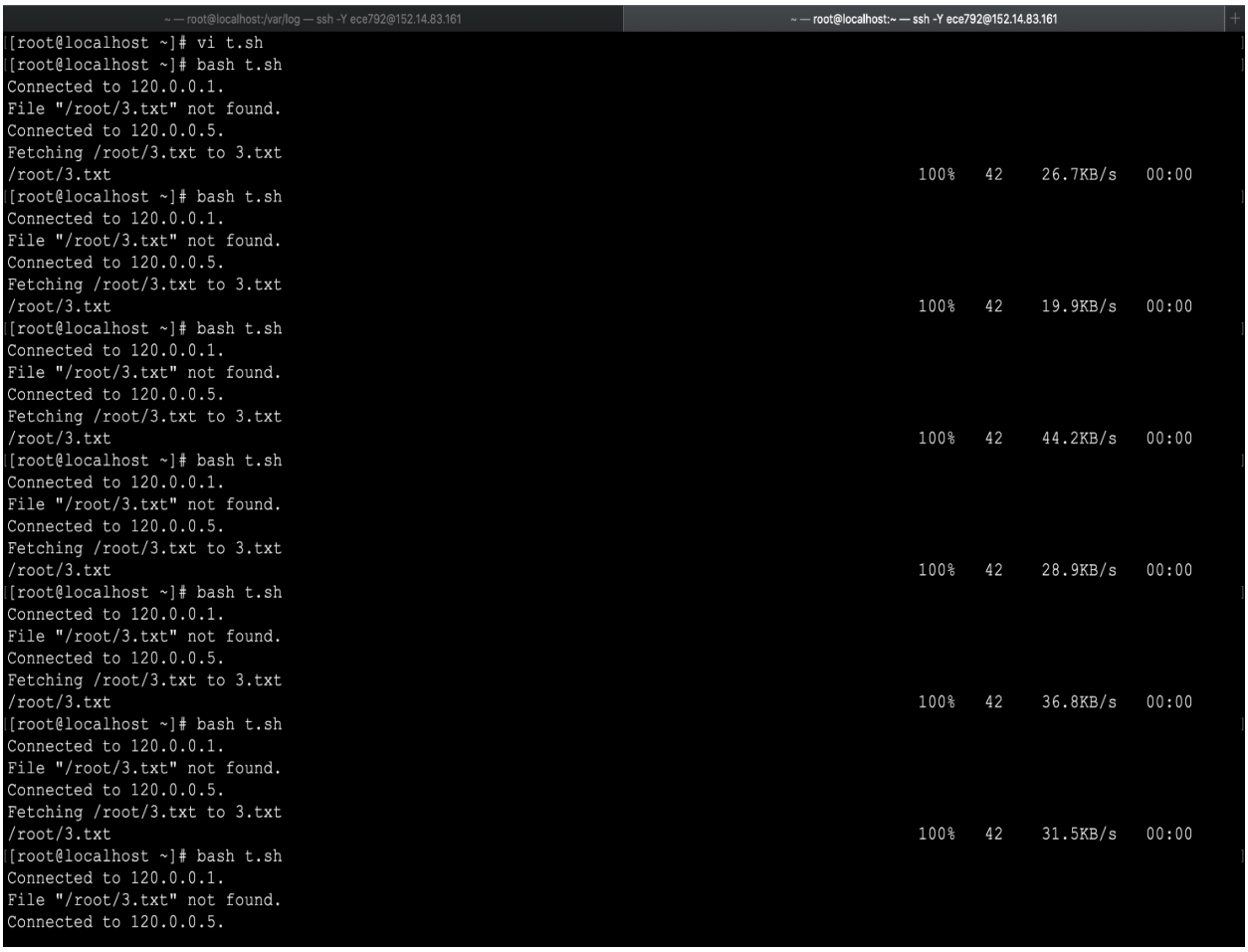


Fig: Snapshot at the Delivery node while fetching files from the Storage node that is not present in the DN.

Fig: Snapshot at the Control node that transfers the file from the storage node to the Delivery node.

The two files i.e 2.mp4 and 3.txt that was not present in the delivery node and specifically present in the storage node was fetched by the delivery node from the SN and then the Control node based on the number of hits for those file in the sftp log of the Delivery node sent the file from the storage node to the delivery node so that the main reason of implementing the content delivery network will be in the function.

Fig: Snapshot at the delivery node when the 2 files not present in the DN got added by the storage nodes.

10. Future Scope:

- ***DNS:***

  This project can be extended to provide DNS capability such that the user requests the files in the form of a domain name and this domain name is resolved to an IP address of the load balancer which does port forwarding to balance the traffic between the VM's having the same data.

- ***Auto-Scaling:***

  Auto-Scaling can be implemented in this project such that the architecture of the client is automatically scaled up and down based on the client threshold for better utilization of resources and hasslefree administration for the service provider.

- ***Media-Streaming:***

  This project can be extended to provide live media streaming to the users simultaneously handing the varying number of users requesting the service.

11. References:

1. https://neuvector.com/network-security/advanced-kubernetes-networking/
2. https://phoenixnap.com/kb/how-to-install-kubernetes-on-centos
3. https://www.techrepublic.com/article/how-to-quickly-install-kubernetes-on-ubuntu/
4. https://www.digitalocean.com/community/tutorials/how-to-create-a-kubernetes-cluster-using-kubeadm-on-ubuntu-18-04
5. https://www.nokia.com/blog/new-approach-publishing-and-caching-video/
6. https://stackoverflow.com/questions/48784098/issue-with-kvm-libvirt-and-linux-namespaces
7. https://ops.tips/blog/using-network-namespaces-and-bridge-to-isolate-servers/
8. https://www.unixmen.com/setting-dns-server-centos-7/
9. https://www.cloudflare.com/learning/cdn/what-is-a-cdn/
10. https://www.webopedia.com/TERM/C/CDN.html
11. https://www.akamai.com/us/en/cdn/what-is-a-cdn.jsp