NC STATE UNIVERSITY

STORAGE AS A SERVICE CLOUD SOLUTION

ECE- 792 LINUX NETWORKING PROJECT REPORT MILESTONE 2

by

Moulik P. Naredi

Student ID: 200317312

Atul Sharma

Student ID: 200284660

Ayushi Rajendra Kumar Chawla

Student ID: 200313243

Priyanka

Student ID: 200323431

Guided by

Prof. Anand Singh

Computer Network Department-NC State University

Overview

Over the last few years, we have seen a new trend of storing data from personal computers to cloud storage or network shared storage devices. This provides users flexibility to access data from anywhere (over the internet or intranet) anytime.

Here we are focusing on building private storage systems which can be used in a Small Enterprise or Small Network.

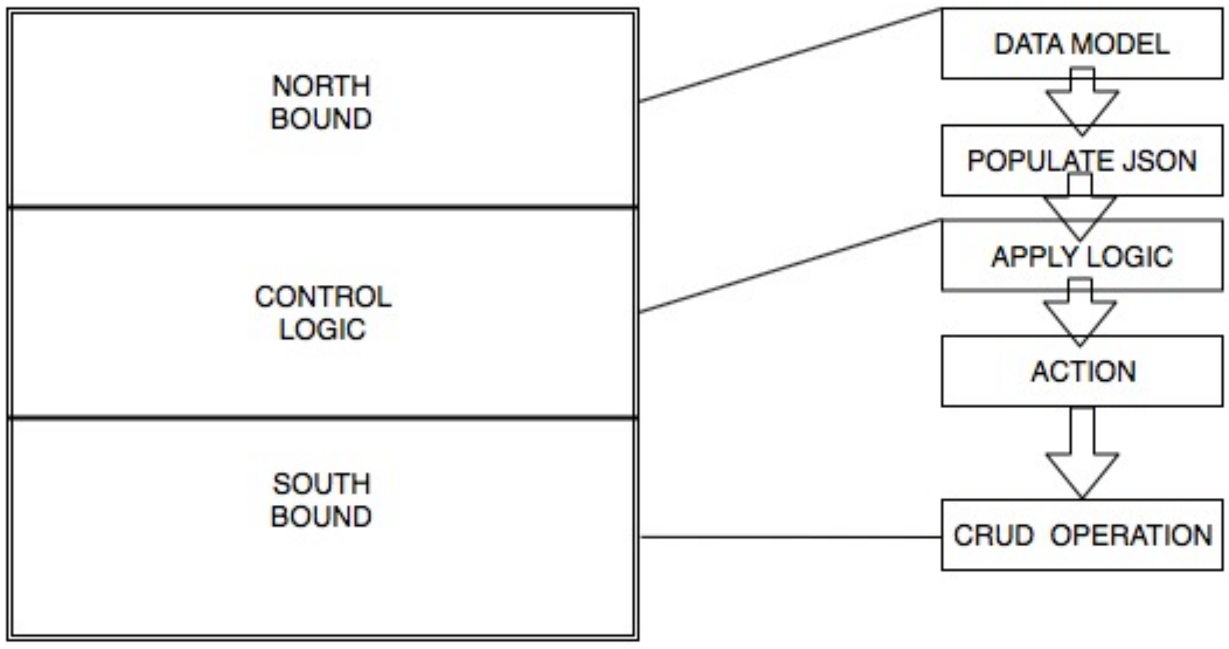
Private Cloud Storage

Private Cloud Storage is basically used by only one Enterprise. Enterprise with critical and sensitive data uses private cloud to store information. Private Cloud Storage reside either On-Premise or at Third Party Vendor. Private Cloud Storage offers advantages such as

- Scalability
- Highly efficient
- Security
- Customization

Virtual Private Cloud (VPC) is an on-demand customized pool of shared computing resources allocated within a public cloud environment. VPC provides isolation between different tenants, organizations using resources. This isolation between users can be achieved by using different private IP subnets, VLANs or encryption channels.

PROJECT APPROACH DIAGRAM:

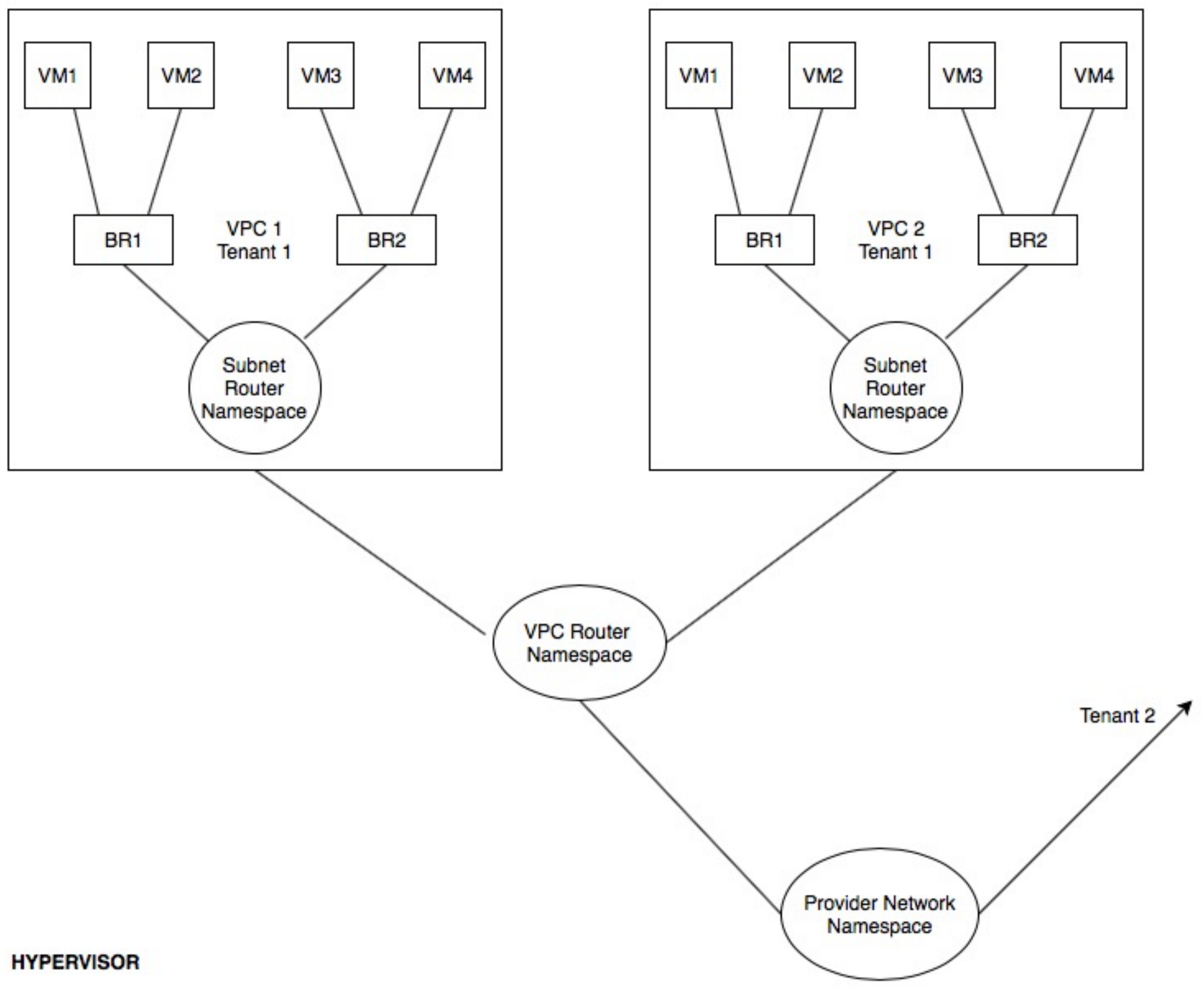


The data model creation involved listing down all the inputs that would be taken from the user and administrator. Then subsequent task involved was organising this raw data into a form that gives ease in extraction of the required data in the controller action process. Hence, this raw data was put in a jason format. This was how the North Bound part of the project was accomplished.

For the controller logic, first the structure was created on paper. A rigorous discussion and a foresight of a myriad number of end cases led to various versions of the structure. The final structure that was decided is depicted below. Once we gained structural clarity, we tested the structure by manually creating the topology in one hypervisor. After this test, we started working upon the logic to automate the entire process. In the process we used the data model that was created in the north bound.

Parallelly we were working on the South Bound which included establishment of CRUD operation. The idea here was that South Bound should be as independent as possible from North Bound. Furthermore, we have made sure that the datapath is as independent as possible from the control path.

Basic Topology



Tenant is an isolation space within the single Hypervisor where a single Tenant can have multiple VPC clouds. Complete isolation within VPCs is provided, meaning each VPC of each tenant can have Overlapping L3 IPs and L2 MAC address without breaking any functionality. Our system provides complete isolation of data between each tenant. This structure can be extended to multiple hypervisors.

COMPONENTS:

PRIVATE NETWORK

The Provider Network block acts as a staging server for the tenants.It controls the traffic between multiple Tenants. The connections originating here are controlled by the provider and the tenant has no information about them.

VPC ROUTER

The VPC router is unique to each tenant. As the name suggests, this acts as a router between various VPCs of each tenant.

SUBNET ROUTER

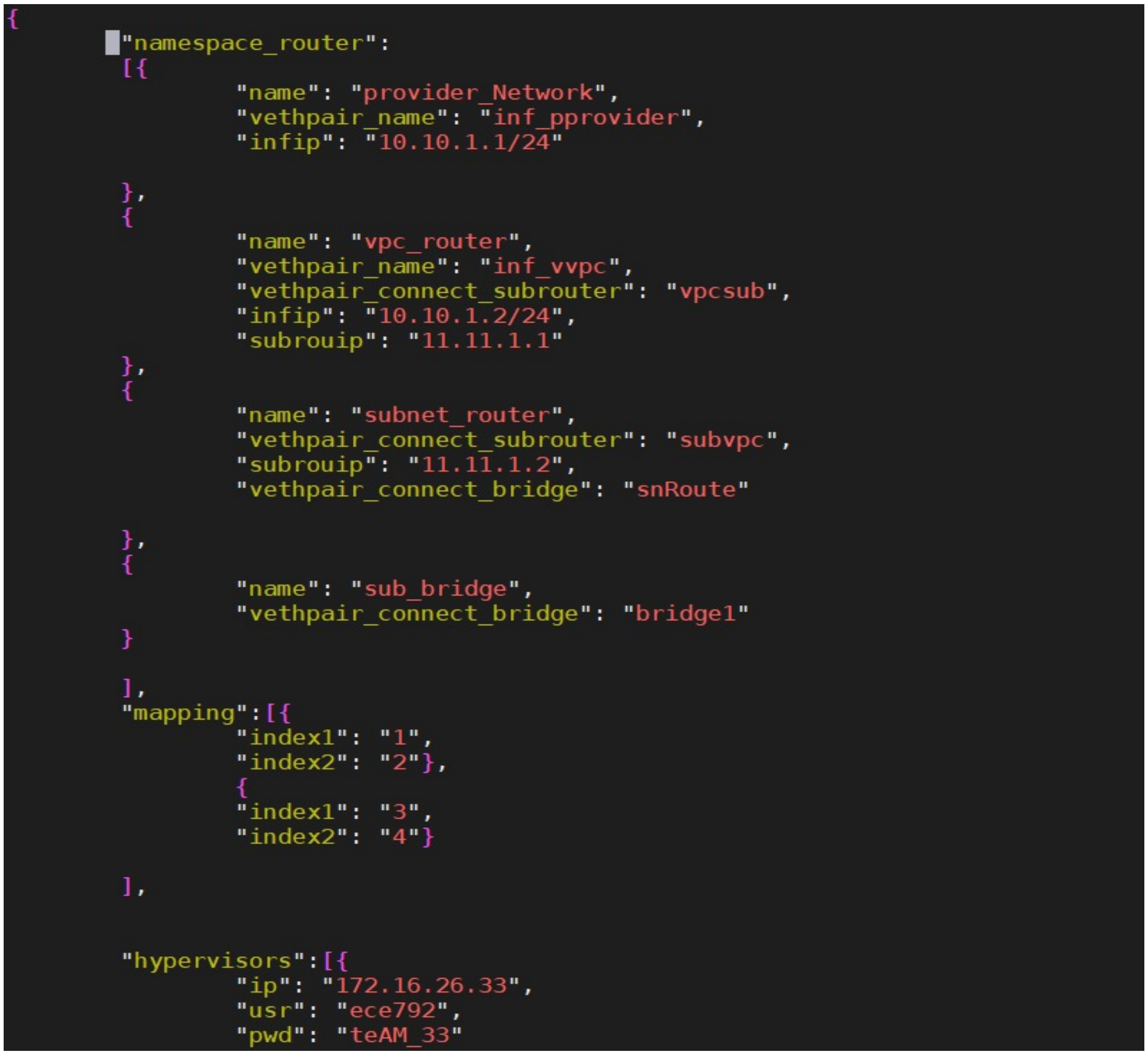
The Subnet router is unique to each VPC. This means that a tenant can have multiple Subnet Routers. Each Subnet Router interconnects all the subnets in that VPC. Each subnet can have multiple VMs connected to a layer 2 bridge. Each subnet.

This structure repeats for every tenant. The IP tables are configured at each stage to ensure that VM has internet connectivity. The tenants are provided with a public IP and ports through which these tenants can access their machines.

A tenant can have VPC extended over more than one hypervisor. In a very similar way, a tenant can have a subnet extended over multiple hypervisors.

THE JASON FILE:

It is a specific format file containing the data that is accessed by the scripts in the controller to perform necessary tasks. For this stage, these files are manually filled by the user. For further stages we aim to fill this through user UI.



```
{
        "namespace_router":
         [{
                "name": "provider_Network",
                "vethpair_name": "inf_pprovider",
                "infip": "10.10.1.1/24"

         },
         {
                "name": "vpc_router",
                "vethpair_name": "inf_vvpc",
                "vethpair_connect_subrouter": "vpcsub",
                "infip": "10.10.1.2/24",
                "subrouip": "11.11.1.1"
         },
         {
                "name": "subnet_router",
                "vethpair_connect_subrouter": "subvpc",
                "subrouip": "11.11.1.2",
                "vethpair_connect_bridge": "snRoute"

         },
         {
                "name": "sub_bridge",
                "vethpair_connect_bridge": "bridge1"
         }

         ],
         "mapping":[{
                "index1": "1",
                "index2": "2"},
                {
                "index1": "3",
                "index2": "4"}

         ],


         "hypervisors":[{
                "ip": "172.16.26.33",
                "usr": "ece792",
                "pwd": "teAM_33"
```

THE CONTROLLER:

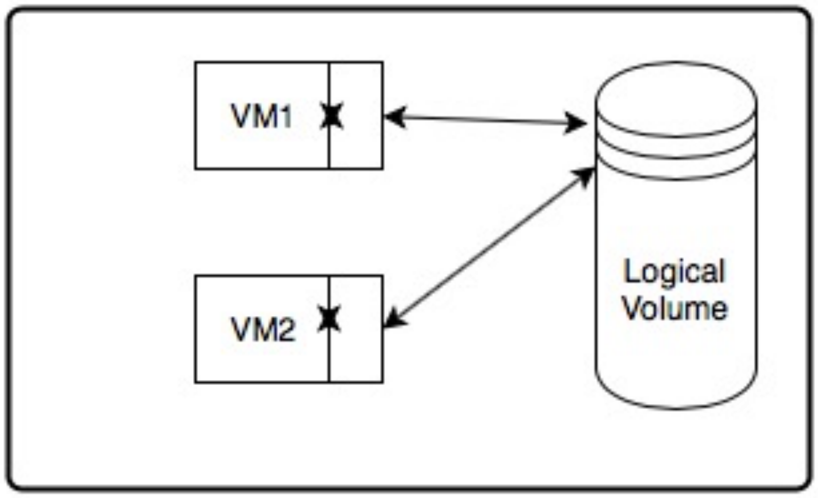
One of the hypervisors currently is configured to act as a controller. All the scripts run in this controller. The decision to add any VM to hypervisor is based on which hypervisor has more space available. The controller takes decision and action of creation of entire topology and attaching storage space. GRE tunnels are implemented to connect VPC routers for the tenants whose VPCs are distributed in multiple hypervisors. Also, Vxlan tunnel would be used for extending a subnet of a tenant in multiple hypervisor.

USER INTERFACE:

The further implementation would include Creation of UI through which the jason file will be automatically populated and the scripts would act upon the jason file to expand topology dynamically.

STORAGE VIRTUALIZATION

On the back end, The first part is to create virtual memory space from given memory space. This is called storage virtualization.This includes presenting a logical view of physical storage resources to the user. There are virtual volumes that are accessed by user application. These volumes are matched to actual storage in arrays. When any I/O is passed to these virtual volumes, it is redirected to the storage network layer.



Allocating Logical Storage Volume to VM

Providing storage along with the Virtual Machine to a Tenant is the simple task but providing additional storage on demand and removing the same is tedious task. This mechanism of attaching, expanding or removing storage from the assigned Virtual Machine should be flexible enough not to interrupt the user’s environment.

We have images directory where we are creating the required space image file in .img format. Required size is provided by the user on the basis of which we are creating chunks and attaching to the VM. We are using the same storage from which VMs are getting their default storage.

File-based storage is a collection of files that are stored on the host physical machines file system that act as virtualized hard drives for guests. To add file-based storage, perform the following steps:

Storage Creation, Attaching to VM and Initialization:

ADMINISTRATOR TASKS:

sudo dd if=/dev/zero of=/var/lib/libvirt/images/<storage name>.img bs=1G count=4

Storage name.img – new partition image. For every VM, this file should be new.

Dom Name – VM/Guest Name
vdb – storage device attached to Guest. This name can be changed.

virsh attach-disk <Dom Name> /var/lib/libvirt/images/<storage name>.img vdb --cache none

(--persistent option can also be selected at the end)

USER TASKS:

The guest now has a hard disk device called /dev/vdb

Now users or guests have an external storage with the name vdb of 4Gb. Guests can format or partition the drive.

fdisk /dev/vdb
Enter – n (for new partition)
Press enter for default values
Enter w and quit from the process
<<< Now storage pool has been created and allocated>>>
Format the new partition by running - 'mke2fs -j /dev/vdb'

REMOVAL OF STORAGE ON DEMAND:

USER TASKS:

On user demand, whenever this is a need to remove the storage,

Remove Storage steps:

1. Delete vdb partition from VM
2. fdisk /dev/vdb
3. Enter ‘d’ to delete the partition
4. Enter ‘w’ for writing the configuration
5. Exit

ADMINISTRATOR TASKS:

Now detach vdb storage from VM.

```
virsh detach-disk <Dom Name> vdb (virtual storage name)
ece792@t13_vm7:/var/lib/libvirt/images$rm <storage name>.img
```

Features of the Solution:

1. Scalable
2. Performance Monitoring
3. Data Replication

Scalability
The VPC design we created is scalable and flexible. Users can add VPCs and subnets at any point of time.
Also there will be situations when a user needs additional space allocation or wants to remove extra space and that time system should accept the requirement without impacting the architecture, performance and outage. So, storage systems should be scalable enough to cater such ad-hoc requirements.

Performance Monitoring
It is very important to keep monitoring all the statistics of the system like CPU Usage, Memory Usage, Available Storage Space and Running Processes etc. It's very useful for the Administrator to avoid any malfunctioning and also to the User/Tenant to manage and avoid service failure. The monitoring comprises of:

1. Load Average
2. CPU Idle Percentage
3. Kernel Utilization
4. Memory Utilization
5. Swapping activity

Data Replication
Users will never want to lose any data. So in order to give such a feature, data should be replicated somewhere in the backend from where any loss can be restored. Data backup will be done on the other Hypervisor which is on other physical hardware so that data loss can be prevented with any hardware level failure. We will explore and implement this feature in the next stage.

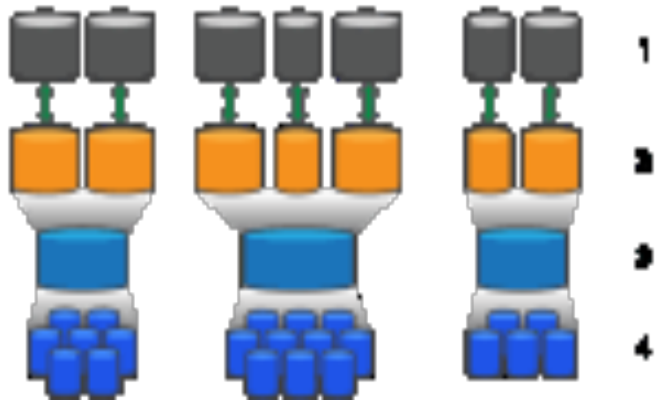
Further Exploration in NFS:

Physical hardware is provisioned into logical components so that data can be organized and easily retrieved. There are two types of groupings supported:

●Pools

●RAID volume groups

The pools and volume groups are the top-level units of storage in a storage array: they divide the capacity of drives into manageable divisions. Within these logical divisions are the individual volumes or LUNs where data is stored. The following figure illustrates this concept.



In the above figure:

1 represents the Host LUN’s , 2 represents Volumes, 3 represents the Volume groups or pools and 4 represents the HDD or SSD Drives.

When a storage system is deployed, the first step is to present the available drive capacity to the various hosts by:

●Creating pools or volume groups with sufficient capacity

●Adding the number of drives required to meet performance requirements to the pool or volume group

●Selecting the desired level of RAID protection (if using volume groups) to meet specific business requirements

You can have pools or volume groups on the same storage system, but a drive cannot be part of more than one pool or volume group. Volumes that are presented to hosts for I/O are then created, using the space on the pool or volume group.

Storage Pools & Volumes:

A storage pool is a quantity of storage set aside for use by guest virtual machines. Storage pools are divided into storage volumes. Each storage volume is assigned to a guest virtual machine as a block device on a guest bus.

Storage pools and volumes are managed using libvirt. With libvirt's remote protocol, it is possible to manage all aspects of a guest virtual machine's life cycle, as well as the configuration of the resources required by the guest virtual machine. These operations can be performed on a remote host.

NFS pools are storage resources provided by OVP hosts, used by virtual machines for storage purposes.
This exercise creates a virtual disk on an NFS pool provided by an OVP host. It uses virsh to configure and operate NFS storage resources.

Procedure

1. Set up the NFS pool directory.
    a. Create the pool directory.
       # mkdir -p /export/x86-64-kvm-guest-pool
    b. Edit /etc/exports to add the corresponding export line.
       # cat /etc/exports/export/x86-64-kvm-guest-pool
       *(rw,no_subtree_check,insecure,no_root_squash)
    c. Tell the NFS server to reload the exports configuration file.
       # exportfs -a
2. Connect to the QEMU hypervisor.
   # virsh connect qemu:///system
3. Load the configuration file for the NFS pool. Before loading the configuration file, it is recommended that you verify that the POOL_HOST variable contains a fully qualified name and not a local name such as localhost. Using fully qualified names allows virtual machines to find the required storage resources, even when they migrate across different OVP hosts.
   # virsh pool-define xmlDir/x86-64-kvm-guest-pool.xml
4. Start the storage pool.
   # virsh pool-start x86-64-kvm-guest-pool
5. Create the storage volume on the x86-64-kvm-guest-pool storage pool.
   # virsh vol-create-as x86-64-kvm-guest-pool x86-64-kvm-guest-vda.raw 10G --format raw .

We can also use the qcow2 format in this case so we can do x86-64-kvm-guest-vda.qcow2 10G --format qcow2 in the above command.

This storage volume henceforth created needs to be added to the Virtual machines created using the virsh attach-disk command.

The NFS storage pool is now available in the directory **/export/x86-64-kvm-guest-pool** of the OVP host. You can verify that a virtual disk file of the requested size has been created.

Rehosting NFS volume

Volume rehost enables you to reassign NAS or SAN volumes from one storage virtual machine (SVM, formerly known as Vserver) to another SVM without requiring a SnapMirror copy.

We need to rehost volumes that serve data over NFS protocol. After rehosting the NFS volumes, to continue accessing data over NFS protocol, you must associate the volume with the export policy of the hosting VM and manually configure policies and associated rules.

Prerequisites for Rehosting:

- The volume must be online.
- Volume management operations, such as volume moves or LUN moves, must not be running.
- Data access to the volume that is being rehosted must be stopped.
- The ns-switch and name services configuration of the target SVM must be configured to support data access of the rehosting volume.

The user ID and group ID of the volume must be either available in the target SVM or changed on the hosting volume

About this task

- Rehosting is a disruptive operation.
- If the rehosting operation fails, you might need to reconfigure the volume policies and the associated rules on the source volume.
- After the rehost operation, the following volume policies, policy rules, and configurations are lost from the source volume, and must be manually reconfigured on the rehosted volume:
  - Volume and qtree export policies
  - Antivirus policies
  - Volume efficiency policy
  - Quality of service (QoS) policies
  - Snapshot policies
  - Quota rules
  - ns-switch and name services configuration export policy and rules
  - User and group IDs

Steps

1. Record information about the NFS export policies to avoid losing information on NFS policies in case volume rehost operation fails.

2. Unmount the volume from the parent volume:
   volume unmount
3. Switch to the advanced privilege level:
   set -privilege advanced
4. Rehost the volume on the destination SVM:
   volume rehost -vserver source_svm -volume vol_name -destination-vserver destination_svm
   The default export policy of the destination SVM is applied to the rehosted volume.
5. Create the export policy:
   vserver export-policy create
6. Update the export policy of the rehosted volume to a user-defined export policy:
   volume modify
7. Mount the volume under the appropriate junction path in the destination SVM:
   volume mount

8. Verify that the NFS service is running on the destination SVM.

9. Resume NFS access to the rehosted volume.

10. Update the NFS client credentials and LIF configurations to reflect the destination SVM LIFs.

    This is because the volume access path (LIFs and junction path) has undergone changes.

Reference:

https://docs.netapp.com/ontap-9/index.jsp?topic=%2Fcom.netapp.doc.dot-cm-vsmg%2FGUID-84308166-6872-47C2-AEC0-D6346AD1D761.html