EMC²

EMC UNITY: OPENSTACK BEST PRACTICES FOR OCATA RELEASE

A Detailed Review

ABSTRACT

This white paper discusses the OpenStack architecture and the integration available with Unity systems. Best practices are outlined for the OpenStack services supported by Unity.

April, 2017

TABLE OF CONTENTS

EXECUTIVE SUMMARY

OpenStack is an open source cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter. It can all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface.

Dell EMC supports various components of the OpenStack project, including Cinder, Nova, Manila, Swift, and Glance. This document will review these componenets in relation to Unity integration. This guide introduces specific configuration recommendations that enable good performance with specified Unity OpenStack drivers.

AUDIENCE

This white paper is intended for storage architects, administrators, partners, Dell EMC employees and any others involved in evaluating, acquiring, managing, operating, or designing an OpenStack environment using Unity. This paper assumes the reader has basic knowledge of OpenStack technology, terminology, and usage as well as Linux knowledge.

UNITY OPENSTACK CINDER

Cinder, the OpenStack block storage driver, provides the persistent block storage resources that OpenStack Compute instances can consume. This includes the secondary attached storage similar to the Amazon Elastic Block Store (EBS) offering. In addition, you can write images to a block storage device for the OpenStack Compute driver to use as a bootable persistent instance.

The block storage service provides:

- *cinder-api* – A Web Server Gateway Interface (WSGI) app that authenticates and routes requests throughout the block storage service. It supports the OpenStack APIs only, although there is a translation that can be done through OpenStack Compute's EC2 interface which calls in to the block storage client.
- *cinder-scheduler* – Schedules and routes requests to the appropriate volume service. Depending upon your configuration, this may be simple round-robin scheduling to the running volume services, or it can be more sophisticated through the use of the Filter Scheduler. The Filter Scheduler is the default and enables filters on things like Capacity, Availability Zone, Volume Types, and Capabilities as well as custom filters.
- *cinder-volume* – Manages block storage devices, specifically the back-end devices themselves.
- *cinder-backup* – Provides a means to back up a block storage volume to OpenStack Object Storage (Swift).

Unity Cinder driver is integrated in OpenStack Cinder project since *Ocata* release. The driver is built on the top of Cinder framework and a Dell EMC distributed Python package storops. No other dependency is required.

CINDER PREREQUISITES

Software	Version
Unity OE	4.1.x or later
OpenStack (Cinder)	Ocata
storops	0.4.7 or newer

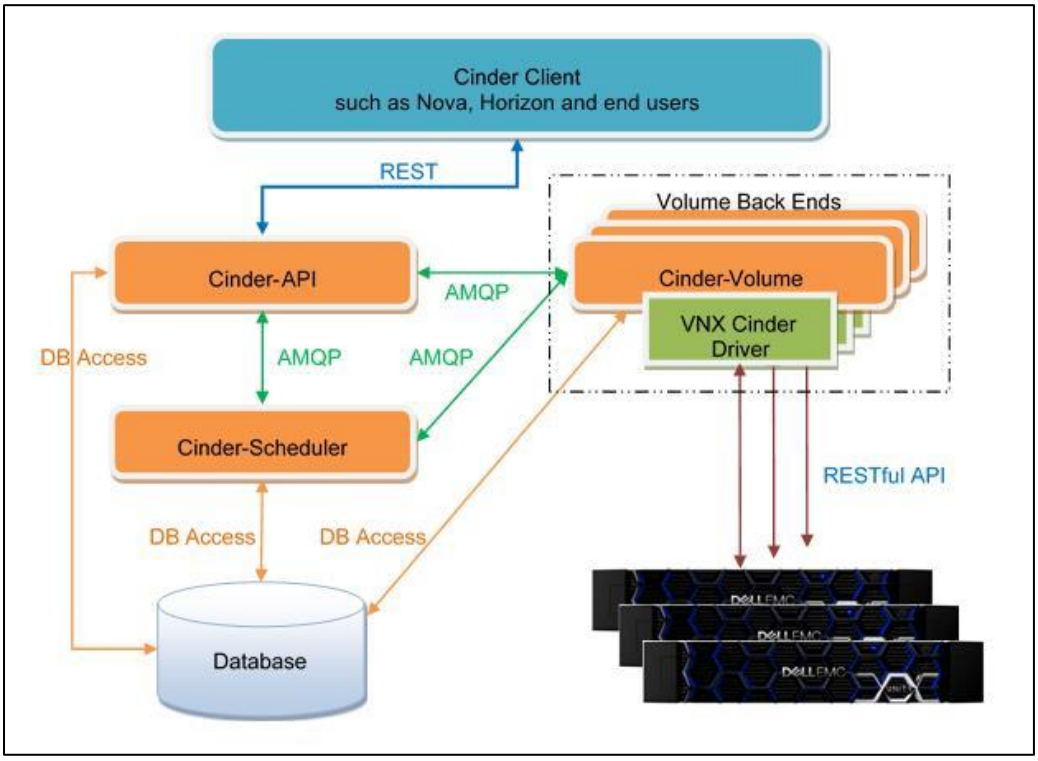

Figure 1 - OpenStack (Cinder) Architecture

CINDER DRIVER CONFIGURATION

The Unity Cinder driver is loaded by the *cinder-volume* service. Please refer to the *cinder-volume* installation steps for your particular deployment platform, available on the official OpenStack website (http://www.openstack.org).

Perform the below steps on your Cinder nodes.

1. Install *storops* from Python Package Index (PyPI):

```
sudo pip install storops
```

2. Configure Cinder to use the Unity driver. Add the following content into */etc/cinder/cinder.conf*:

```
[DEFAULT]
enabled_backends = unity
[unity]
# Storage protocol
storage_protocol = iSCSI
# Unisphere IP
san_ip = <SAN IP>
# Unisphere username and password
san_login = <SAN LOGIN>
san_password = <SAN PASSWORD>
# Volume driver name
volume_driver = cinder.volume.drivers.dell_emc.unity.Driver
# backend's name
volume_backend_name = Storage_ISCSI_01
sudo pip install storops
```

Note: These are the minimal set of options for the Unity driver. For more options, see the *Cinder Driver Options* section.

The following are optional steps to be performed on both Cinder and Nova nodes if multipath based data access is required.

1. Install *sysfsutils*, *sg3-utils* and *multipath-tools*:

```
sudo apt-get install multipath-tools sg3-utils sysfsutils
```

2. Zone Nova FC ports with the Unity FC target port (required for FC driver in case Auto-zoning Support is disabled; see the *Auto-Zoning Support* section for more information).

3. Enable Unity storage optimized multipath configuration by appending the following content into */etc/multipath.conf*:

```
blacklist {
    # Skip the files uner /dev that are definitely not FC/iSCSI devices
    # Different system may need different customization
    devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]*"
    devnode "^hd[a-z][0-9]*"
    devnode "^cciss!c[0-9]d[0-9]*[p[0-9]*]"

    # Skip LUNZ device from VNX/Unity
    device {
        vendor "DGC"
        product "LUNZ"
    }
}

defaults {
    user_friendly_names no
    flush_on_last_del yes
}

devices {
    # Device attributed for EMC CLARiiON and VNX/Unity series ALUA
    device {
        vendor "DGC"
        product ".*"
        product_blacklist "LUNZ"
        path_grouping_policy group_by_prio
        path_selector "round-robin 0"
        path_checker emc_clariion
        features "0"
        no_path_retry 12
        hardware_handler "1 alua"
        prio alua
        failback immediate
    }
}
```

4. Restart the multipath service:

```
sudo service multipath-tools restart
```

5. Enable multipath for image transfer in */etc/cinder/cinder.conf*:

```
use_multipath_for_image_xfer = True
```

6. Restart the *cinder-volume* service to load the change.

7. Enable multipath for volume attach/detach in */etc/nova/nova.conf*:

```
[libvirt]
...

volume_use_multipath = True

...
```

8. Restart the *nova-compute* service.

Cinder Driver Options

Option Name	Type	Description
storage_protocol	Optional	Storage protocol to provide data access. Can be *iscsi* (Default) or *fc.*
unity_storage_pool_names	Optional	Pool names to be managed by Cinder, if not specified, all pools will be managed.
san_ip	Required	Unisphere IP to manage Unity storage.
san_login	Required	Username to access Unity.
san_password	Required	Password to access Unity.
volume_driver	Required	cinder.volume.drivers.dell_emc.unity.Driver.
volume_backend_name	Optional	Any descriptive string to identify this cinder back-end. The default value is UnityiSCSI else UnityFCDriver.
use_multipath_for_image_xfer	Optional	Use *True* to enable multipath for volume to image and image to volume. The default value is *False.*
unity_io_ports	Optional	Specify the list of FC or iSCSI ports to be used to perform the I/O. Asterisk wildcard character is supported.

NOVA LIVE MIGRATION INTEGRATION

It is suggested to have *multipath* configured on Nova nodes for robust data access during live migration of VM instances. Once *user_friendly_names no* is set in *defaults* section of */etc/multipath.conf*, nova nodes will use the WWID as the alias for the multipath devices.

To enable multipath in Nova migration, perform the steps below. Make sure the *Cinder Driver Configuration* steps are performed before doing the following steps:

1. Set multipath in */etc/nova/nova.conf*:

```
[libvirt]
...

volume_use_multipath = True

...
```

2. Restart the *nova-compute* service.

3. Set *user_friendly_names no* in */etc/multipath.conf*:

```
...

defaults {
    user_friendly_names no
}

...
```

4. Restart *multipath-tools* service.

THIN/THICK PROVISIONING

Only *Thin* volume provisioning is supported by the Unity Cinder driver.

AUTO-ZONING SUPPORT

Unity Cinder driver supports **Auto-Zoning**, and shares the same configuration guide as other vendors. Please refer to the Fibre Channel Zone Manager webpage at the OpenStack website (http://openstack.org) for detailed configuration steps.

SOLUTION FOR LUNZ DEVICE

Dell EMC's recommended best practice is to present a LUN with HLU 0 to clear any LUNZ devices as they can cause issues on the host. See KB article 463402 for more information. To workaround this issue, Unity Cinder driver automatically creates a *Dummy LUN* (if not present), and adds it to each host to occupy the *HLU 0* during volume attachment. This *Dummy LUN* is shared between all host connected to the Unity system.

DEPLOYMENT TIPS

Timeout Settings

rpc_response_timeout in the *[DEFAULT]* section of *cinder.conf* defines the seconds to wait for a response from the *cinder-api* service. The default value is 60 seconds if rpc_response_timeout is not specified explicitly. When the *cinder-volume* service or Unity system has high load, 60 seconds will not be enough and different deployments may need different values. If no empirical value is available for a specific deployment, the following value is recommended in *cinder.conf*:

```
[DEFAULT]
...
rpc_response_timeout = 600
...
```

Since some functionality of Nova will need to send a request to the *cinder-api* service, the response time of Cinder Remote Procedure Calls (RPC) will be transferred to Nova. For this reason, *rpc_response_timeout* in the *[DEFAULT]* section of *nova.conf* should be set to a value larger than the one in *cinder.conf*.

HAProxy is a very fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications. It can be a proxy for the *cinder-api* service. The OpenStack High Availability Guide, available on the official OpenStack website, introduces the solution to have HAProxy integrated with OpenStack services. The "*timeout server*" in HAProxy is the time that HAProxy will wait for a response before HAProxy considers the server-side timed out (HTTP Gateway Timeout 504 will be returned to the client side when the timeout happens). It follows that "*timeout server*" should be set to a value larger than *rpc_response_timeout* in the *[DEFAULT]* section of *cinder.conf*.

Storage backends, including Unity, will periodically send statistic update to the *cinder-scheduler* service for check-in. *service_down_time* in the *[DEFAULT]* section of *cinder.conf* defines the maximum number of seconds since the last check-in for a service to be considered alive. The default value is 60 seconds if *service_down_time* is not specified explicitly. When a storage backend is busy, the update may not occur within the defined window. Thus, the *cinder-scheduler* service may consider the storage backend unresponsive in error. It is suggested to extend the value according to the hardware condition and the load of the environment. If no empirical value is available for the specific deployment, a value of 600 is recommended.

OpenStack components rely on the system clock for timeout calculations. If different nodes have desynced on their clocks, the components' connections may become unpredictable. It is suggested that all nodes have consistent NTP configuration to synchronize system time.

TROUBLESHOOTING

To troubleshoot a failure in an OpenStack deployment, the best way is to enable verbose and debug logging. At the same time, leverage the built-in Return requested ID to caller feature to track specific Cinder command logs. More information on this feature can be found at the official OpenStack website.

Here are the general steps for troubleshooting use cases:

1. Enable verbose and debug logging. Set the following in */etc/cinder/cinder.conf* and restart all Cinder services:

```
[DEFALUT]

...

debug = True
verbose = True

...
```

Note: If other projects/drivers are also involved, it is recommended to also set *debug* and *verbose* to *True* for those projects/drivers as well.

2. Use *–debug* to trigger any problematic cinder operation like below:

```
cinder --debug create --name unity_vol1 100
```

You will see the request ID from the console like below:

```
DEBUG:keystoneauth:REQ: curl -g -i -X POST
http://192.168.1.9:8776/v2/e50d22bdb5a34078a8bfe7be89324078/volumes -H
"User-Agent: python-cinderclient" -H "Content-Type: application/json" -H
"Accept: application/json" -H "X-Auth-Token:
{SHA1}bf4a85ad64302b67a39ad7c6f695a9630f39ab0e" -d '{"volume": {"status":
"creating", "user_id": null, "name": "unity_vol1", "imageRef": null,
"availability_zone": null, "description": null, "multiattach": false,
"attach_status": "detached", "volume_type": null, "metadata": {},
"consistencygroup_id": null, "source_volid": null, "snapshot_id": null,
"project_id": null, "source_replica": null, "size": 10}}'
DEBUG:keystoneauth:RESP: [202] X-Compute-Request-Id:
req-3a459e0e-871a-49f9-9796-b63cc48b5015 Content-Type: application/json
Content-Length: 804 X-Openstack-Request-Id:
req-3a459e0e-871a-49f9-9796-b63cc48b5015 Date: Mon, 12 Dec 2016 09:31:44 GMT
Connection: keep-alive
```

3. Use commands like *grep* and *awk* to find the error related to the cinder operation:

```
grep "req-3a459e0e-871a-49f9-9796-b63cc48b5015" cinder-volume.log
```

UNITY OPENSTACK MANILA

MANILA OVERVIEW

The Shared File Systems service (Manila) provides shared file systems that compute instances can consume.

The Shared File Systems service contains following four components:

- *manila-api* – A WSGI app that authenticates and routes requests throughout the Shared File Systems service. It supports the OpenStack APIs.
- *manila-data* – A standalone service whose purpose is to receive requests and process data operations with potentially long running time such as copying, share migration or backup.
- *manila-scheduler* – Schedules and routes requests to the appropriate share service. The scheduler uses configurable filters and weighers to route requests. The Filter Scheduler is the default and enables filters on things like Capacity, Availability Zone, Share Types, and Capabilities as well as custom filters.
- *manila-share* – Manages back-end devices that provide shared file systems. A manila-share service can run in one of two modes, with or without handling of share servers. Share servers export file shares via share networks. When shared servers are not used, the networking requirements are handled outside of Manila.

UNITY AND MANILA INTEGRATION

The Unity Manila driver has been part of the official OpenStack project since the *Newton* release. It supports both NFS and CIFS (SMB) protocol.

The Unity Manila driver provides IP-based authentication method support for NFS shares and user based authentication method support for CIFS (SMB) shares.

For CIFS (SMB) shares, Microsoft Active Directory is the only supported security service.

The Unity Manila driver supports the following operations:

- Create a share.
- Delete a share.
- Allow share access.
- Deny share access.
- Create a snapshot.
- Delete a snapshot.
- Create a share from a snapshot.
- Extend a share.

MANILA PREREQUISITES

Software	Version
Unity OE	4.1.x or later
OpenStack	Ocata
storops	0.4.7 or newer

SUPPORTED NETWORK TYPES

The Unity Manila driver supports the "driver handles share servers" (DHSS) mode, where the Unity driver can dynamically allocate one share server ("NAS Server" on Unity) for each Manila share network. Unity driver suppors *Flat* and *VLAN* network types with its built-in functionality, and can also co-exist with VXLAN networks on some specific hardware configurations.

FLAT NETWORK

This type is fully supported by the Unity Manila driver; however, flat networks are restricted to the limited number of tenant networks that can be created from them.

VLAN NETWORK

This type is the recommended Network topology in Manila.

In most use cases, VLANs are used to isolate different tenants and provide an isolated network for each tenant. To support this functionality, administrators needs to set the slot connected with the Unity Ethernet port to Trunk mode or allow multiple VLANs on the associated ethernet switch.

VXLAN NETWORK

Unity native VXLAN is unavailable, but by leveraging Hierarchical Port Bind (HPB) in Neutron and Manila, it is possible to have Unity co-exist with a VXLAN enabled network environment.

Here is a typical reference setup for VXLAN with HPB architecture:

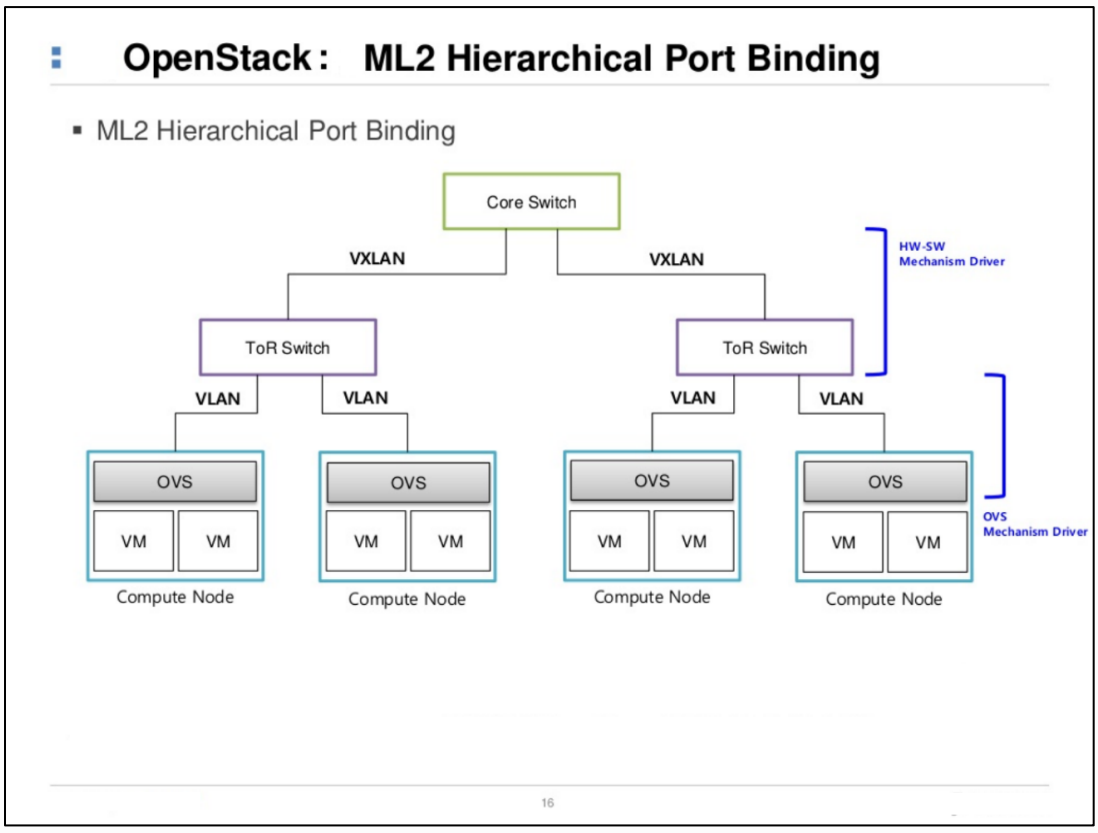

Figure 2 - Hierarchical Port Binding Architecture

In the above architecture, OpenStack nodes on the same rack communicate with each other using VLANs. For nodes on different racks, the VXLAN protocol is used instead. Currently not all switch vendors support HPB architecture, so please consult your switch vendor for more specific information. In the above example, the Unity storage is connected with a ToR switch.

MANILA DRIVER CONFIGURATION

Perform the following steps on the Unity system prior to configuring the Manila driver:

Note: The time on the Unity system and the Active Directory domains used in security services should be in sync. It is recommended to use the same NTP server on both the Unity system and Active Directory domains.

1. Configure NTP on the Unity system.

   Navigate to **System Time and NTP** page in Unisphere:

   - **Unisphere > Settings > Management > System Time and NTP**
   - Select **Enable NTP synchronization** and add your NTP server(s)

2. Configure the DNS on Unity system.

   Navigate to **DNS Server** page in Unisphere:

   - **Unisphere > Settings > Management > DNS Server**
   - Select **Configure DNS server address manually** and add your DNS server(s)

Perform the following steps on Manila service nodes:

1. Install *storops*:

```
sudo pip install storops
```

2. Place the options below in */etc/manila/manila.conf* and adjust according to your environment:

```
[DEFAULT]
enabled_backends = unity
[unity]
# Share driver name
share_driver = manila.share.drivers.emc.driver.EMCShareDriver
# Share backend to load, You must specify the back end name as unity
emc_share_backend = unity
# Management IP of Unity system
emc_nas_server = <EMC NAS SERVER>
# Username to login
emc_nas_login = <EMC NAS LOGIN>
# Password to login
emc_nas_password = <EMC NAS PASSWORD>
# Pool to store metadata of NAS server
unity_server_meta_pool = <NAS SERVER POOL>
# comma separated pool name
unity_share_data_pools = <EMC NAS POOL NAMES>
# Ethernet ports to interface allocation of NAS server
unity_ethernet_ports = <EMC PORTS>
# Must be True for Unity driver
driver_handles_share_servers = True
```

3. Restart the *manila-share* service to enable the changes.

Manila Driver Options

Option Name	Type	Description
emc_share_backend	Required	Must specify *unity.*
emc_nas_server	Required	Unity management IP address.
unity_server_meta_pool	Required	The name of the pool to persist the meta-data of NAS server.
unity_share_data_pools	Optional	Comma separated list specifying the name of the pools to be used by the backend. Do not set this option if all storage pools on the system can be used. Wild card character is supported. Examples: pool_*.
unity_ethernet_ports	Optional	Comma separated list specifying the ethernet ports of Unity system that can be used for share. Do not set this option if all ethernet ports can be used. Asterisk wildcard character is supported. Exmples *spa_eth1*, *spa_*, *.*
driver_handles_share_servers	Required	Unity driver requires this option to be as True.

DEPLOYMENT TIPS

Mount the NFS Share on VM Instances

On the instance, execute the following commands before mounting the NFS share:

```
sudo apt-get update
sudo apt-get install nfs-common
```

Maximum Supported NAS Server

The maximum number of Manila share networks is equal to the Unity NAS Server system limit, as the Unity Manila driver creates one NAS Server for each Manila share network. This restriction is subject to the limits of the Unity system and the number may vary in different models and software releases.

MTU Size Setting

Unity supports 2 MTU sizes: *1500* and *9000*. In most use cases, *1500* is generally suitable. In a performance sensitive deployment environment, it is suggested to set MTU to *9000* on physical interfaces for both Unity systems and Compute nodes. Note that inconsistent MTU size settings in the same L2 network may cause performance degredation and network lag.

Link Aggregation

Unity supports **Link Aggregation**, where multiple physical Ethernet ports can be combined together to make a single high-bandwidth data path for hosts/clients.

To enable this feature in Unity Manila driver, follow the steps below:

1. Configure link aggregation on the Unity system.

   Open Unisphere and access the **Ethernet** settings:

   - **Unisphere > Settings > Access > Ethernet**
   - Click the **Link Aggregation** dropdown button, choose **Create Link Aggregation** and configure accordingly
   - Note down port IDs of link aggregation port(s)

2. On your Manila node, add the link aggegation port(s) in */etc/manila/manila.conf* using the internal port IDs:

```
...

emc_interface_ports = spa_la_1

...
```

3. Restart the *manila-share* service to load the change.

I/O Load Balance

The Unity Manila driver automatically distributes the file interfaces per storage processor based on the option *unity_ethernet_ports*. This balances I/O traffic. The configuration for *unity_ethernet_ports* should specify balanced ports per storage processor. See below for example:

```
# Use eth2 from both SPs
unity_ethernet_ports = spa_eth2, spb_eth2
```

UNITY OPENSTACK SWIFT

SWIFT OVERVIEW

Swift, the OpenStack Object Storage service, is a multi-tenant object storage system. It is highly scalable and can manage large amounts of unstructured data at low cost through a RESTful HTTP API.

It includes the following components:

- Proxy services
- Account servicess
- Container services
- Object services

Here is the Swift brief architecture overview:

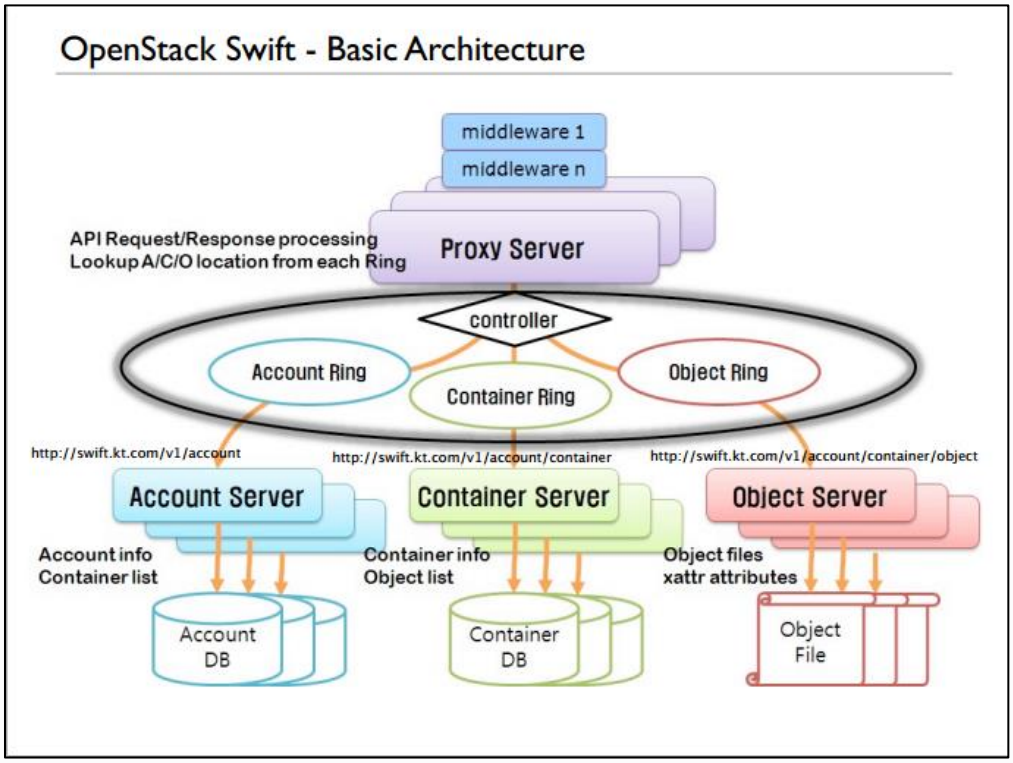

Figure 3 – Swift architecture overview

UNITY AND SWIFT INTEGRATION

This section describes how to configure the Unity system to provide storage for Swift clusters. For simplicity, this guide installs and configures the proxy services on the proxy nodes, the object services on the object nodes.

SWIFT ZONES AND UNITY STORAGE

From the OpenStack Configuration Reference Guide on OpenStack's website:

"In OpenStack Object Storage, data is placed across different tiers of failure domains. First, data is spread across regions, then zones, then servers, and finally across drives. Data is placed to get the highest failure domain isolation. If you deploy multiple regions, the Object Storage service places the data across the regions. Within a region, each replica of the data should be stored in unique zones, if possible. If there is only one zone, data should be placed on different servers. And if there is only one server, data should be placed on different drives."

To accommodate the reasons above, this guide will use the reference topology setup pictured below. Each zone contains one Unity system and all servers in one zone use storage from that Unity system.

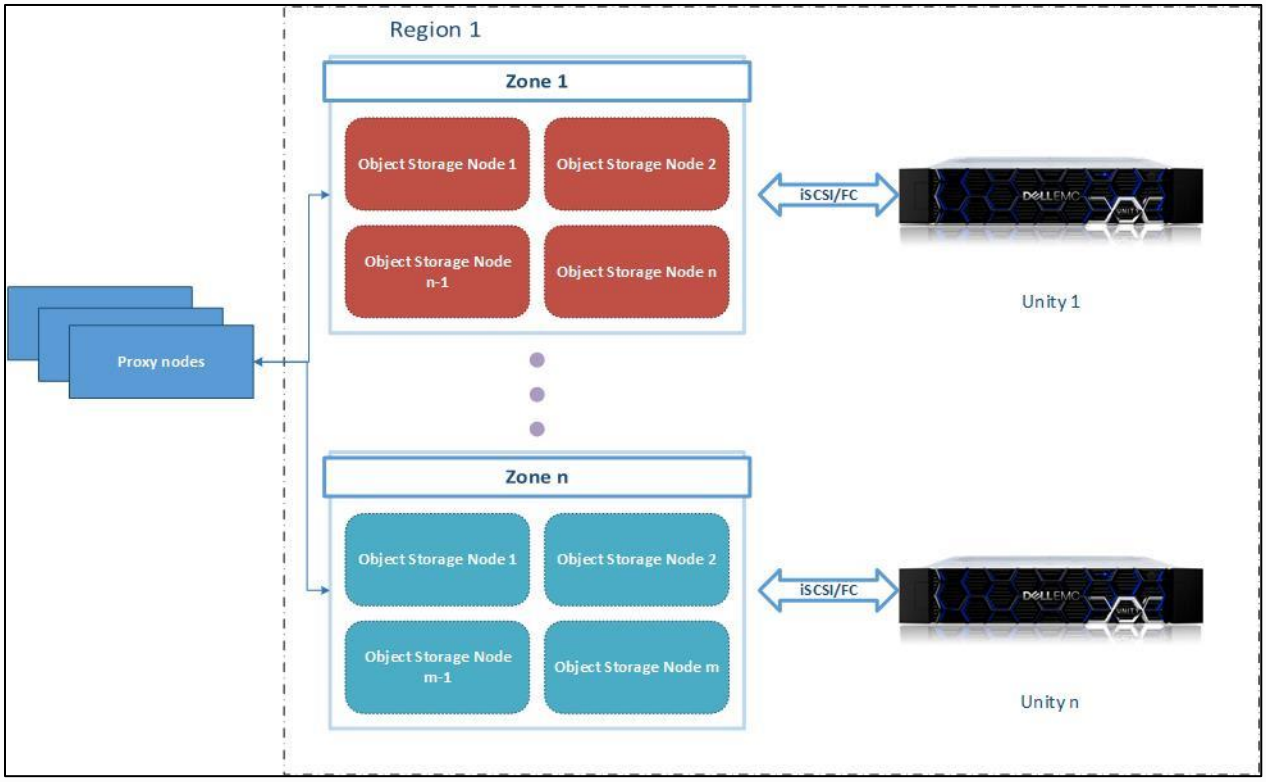

Figure 4 - Swift Reference Technology

SWIFT PREREQUISITES

- Swift required components are configured
- Network connectivity is configured between Swift object servers and Unity arrays

CONFIGURATION STEPS

Perform these steps on each storage/object node:

1. Create LUNs from Unity, then map them to the object servers as needed.
    - Create LUNs on Unity
    - Create hosts for each object server
    - Add LUNs to hosts
2. Discover LUNs on each object server.

    2.1. If iSCSI connection is used:

    2.1.1. Install the *open-iscsi* package on object server.

```
sudo apt-get install open-iscsi
```

    2.1.2. Connect to Unity ISCSI targets.

```
sudo iscsiadm -m discovery -p 192.168.1.59:3260  -t sendtargets
sudo iscsiadm -m node --login
sudo iscsiadm -m session -R
```

2.2. If FibreChannel is used:

2.2.1. Zone the object servers and the Unity FC ports.

2.3. (Optional) Install and configure Linux *multipath-tools*.

```
sudo apt-get install multipath-tools
```

Put the following Dell EMC recommended contents in */etc/multipath.conf*:

```
blacklist {
    # Skip the files uner /dev that are definitely not FC/iSCSI devices
    # Different system may need different customization
    devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]*"
    devnode "^hd[a-z][0-9]*"
    devnode "^cciss!c[0-9]d[0-9]*[p[0-9]*]"

    # Skip LUNZ device from VNX
    device {
        vendor "DGC"
        product "LUNZ"
    }
}

defaults {
    user_friendly_names no
    flush_on_last_del yes
}

devices {
    # Device attributed for EMC CLARiiON and VNX series ALUA
    device {
        vendor "DGC"
        product ".*"
        product_blacklist "LUNZ"
        path_grouping_policy group_by_prio
        path_selector "round-robin 0"
        path_checker emc_clariion
        features "0"
        no_path_retry 12
        hardware_handler "1 alua"
        prio alua
        failback immediate
    }
}
```

And restart the multipath service:

```
sudo service multipath-tools restart
```

2.4. Now, the LUNs should be mapped by multipath tools. Here we find the newly created LUN of 100GB.

```
sudo multipath -r
```

```
reload: 36006016074e03a00a2a04658cd4e046e undef DGC     ,VRAID
size=100G features='0' hwhandler='1 alua' wp=undef
|-+- policy='round-robin 0' prio=130 status=undef
| `- 10:0:0:1 sdg 8:96  active ready  running
`-+- policy='round-robin 0' prio=10 status=undef
  `- 11:0:0:1 sdh 8:112 active ready  running
```

2.5. Format the LUN with *xfs* formatting.

```
sudo mkfs.xfs /dev/mapper/36006016074e03a00a2a04658cd4e046e
```

2.6. Make partition auto mounted on boot.

```
echo "/dev/mapper/36006016074e03a00a2a04658cd4e046e /srv/node/unity_drive1 xfs
noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab
```

Note: Following */srv/node* must match option devices in */etc/swift/object-server.conf*. *unity_driver1* is the device name to be added to the Swift object ring.

2.7. Mount the partition under */srv/node* as *unity_driver1*.

```
mount -t xfs /dev/mapper/36006016074e03a00a2a04658cd4e046e
```

Perform the following steps on each proxy node:

1. Add devices on storage node to the object ring.

   - Change directory to */etc/swift* (specified by option *swift_dir* in */etc/swift/proxy-server.conf*)

```
cd /etc/swift
```

```
swift-ring-builder object.builder \
add --region 1 --zone 1 --ip STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS
--port 6002 \
--device DEVICE_NAME --weight DEVICE_WEIGHT
```

2. Verify the object ring and rebalance.

```
swift-ring-builder object.builder
swift-ring-builder object.builder rebalance
```

3. Copy *account.ring.gz*, *container.ring.gz*, and *object.ring.gz* files to the */etc/swift/* directory on each storage node and any additional nodes running the proxy service.

UNITY OPENSTACK GLANCE

GLANCE OVERVIEW

Glance, the OpenStack image service, is the supported image service for OpenStack Compute. Compute relies on it to store virtual machine images and maintain a catalog of availabe images.

The image service has two user-facing APIs, and the registry API, which is for internal requests that require access to the database.

Glance uses a local file system to store images by default, it also supports several backends for storing virtual machine images:

- Block storage service (Cinder)
- A directory on a local file system
- HTTP
- Ceph RBD
- Sheepdog
- Object Storage service (Swift)
- VMware ESX

UNITY AND GLANCE INTERGRATION

Glance can be configured to use storage provided by Cinder, following documentation will cover how to enable Cinder as a Glance backend as well as leverage the *volume-cloning* functionality provided by Unity Cinder driver.

GLANCE CONFIGURATIONS

User can enable Cinder-backed image management by make following changes in Glance.

1. Add *cinder* in default section:

```
[DEFAULT]
...
stores = file, cinder
default_store = cinder
...
```

2. Restart *glance-api* and *glance-registry* services:

```
$ sudo service glance-api restart

$ sudo service glance-registry restart
```

3. Create a new image in Glance:

```
$ openstack image create --file <image file> <image name>
```

4. Use command to test:

```
$ openstack volume list

+--------------------------------------+--------------------------------------------+-----------
+------+-------------+

| ID                                   | Display Name                               | Status    |
Size | Attached to |

+--------------------------------------+--------------------------------------------+-----------
+------+-------------+

| 201ffc2b-2114-491f-94ca-81f5694da55b | image-6e0fc8a2-760e-41f4-b2d3-64cc6ceef5cd | available |
1 |             |

+--------------------------------------+--------------------------------------------+-----------
+------+-------------+
```

Note: The *Display Name* follows the pattern *image-<image-id>* where *<image-id>* is the unique ID in glance.

UNITY VOLUME CLONING FOR GLANCE CONFIGURATIONS

Before doing the following steps, ensure *Unity OpenStack Cinder* is configured first.

If a user needs to leverage the volume-cloning functionality for both the *create volume from image* and *create image from volume* operations, the following steps need to be performed accordingly.

1. Configure in */etc/glance/glance-api.conf*

```
[DEFAULT]

..

show_multiple_locations = True

..
```

2. Configure in */etc/cinder/cinder.conf*

```
[DEFAULT]

..

glance_api_version = 2

allowed_direct_url_schemes = cinder

..
```

3. Append the following in each back-end section of */etc/cinder/cinder.conf*:

```
[back-end section]

...

image_upload_use_cinder_backend = True

..
```

4. Restart Glance and Cinder services to let the changes take effect.

```
$ sudo service glance-api restart

$ sudo service glance-registry restart

$ sudo service cinder-volume restart
```