Deploying and Configuring Object Storage (Swift)

Contents

Deploying and Configuring Object Storage (Swift)

This page provides detailed instructions on the deployment of HP Helion OpenStack Object Storage, and their configuration.

Preview of the Swift deployment

Perform the following steps to deploy Object Storage (Swift).

Prerequisite

1. HP Helion OpenStack deployer node must be installed.

Before starting the initial deployment you must update the input model by providing the following information:

Setting up the cloud model

- Hardware Requirement
- Allocate servers and disk drives
- Allocate networks
- Allocate a Ring -Builder Node
- Specify the rings (ring specifications)
- Making site-specific changes to configuration parameters (not aware about this. need input from dev team)

Hardware Requirement

It is recommended to have two disk drives on 3 controller nodes to run swift services. How do we add extra drive ? is it in baremetal file etc??

To run a swift service you require two disk drives on 3 controller server. do we edit this in baremetal file.???

single zone

Allocate servers and disk drives

The disk model describes the number of disk present on a particular server type and its usage. Disk are used by the operating system (for example: for the root, log crash filesystem) and swift storage.

- A disk model specifies a number of disk drives/devices and then allocte them for a specific purpose. The general syntax is as follows:

```
device-groups:

  - name: group_name
    devices:
      -  name: sdc
      -  name: sdd
      -  name: sde
    consumer:
        name: <subsystem>
        attrs:
            <attributes-sepecific-to-subsystem>
```

The following is a sample for object store. The **consumer** contains an item called **name**. Here the **subsystem (name)** is **swift**. The **attrs** must contain an item called **rings**. The **rings** item contains a list of ring names.

```
device-groups:

  - name: swiftdevices
    devices:
      -  name: sdc
      -  name: sdd
      -  name: sde
    consumer:
        name: swift
        attrs:
            rings:
              -  name: account
              -  name: container
              -  name: object-0
```

In the above example, a server using this model has at least three drives: `dev/sdc`, `/dev/sdd`, `/dev/sde`. Apart from the listed drives there may be other drives present in the server which are allocated to other subsystem or used as the boot device. If a device is not present in the disk mode, but exists on a server, it will not be used. Conversely, if a device is listed in a disk model and is not present in the server, the deployment fails. In this case you must either add the device or change the disk model so that the device is not mentioned.

Note: The device-group name is for information and has no effect on the drives usage of Object Storage system.

The `sdc` `sdd`,and `sde` drives are dedicated to the Object Storage system. And, the devices are used by the account, container, and object-0 rings. Refer ring specification for more information on rings.

You can modify or edit the disk models to suite your usage of the system. For example, given the same set of drives, you can allocate devices to rings in a different topology as follows:

```
device-groups:

  - name: metadata
    devices:
      -  name: sdc
    consumer:
        name: swift
        attrs:
            rings:
              -  name: account
              -  name: container
 - name: data
 devices:
      -  name: sdd
      -  name: sde
    consumer:
        name: swift
        attrs:
            rings:
              -  name: object-0
              -  name: object-1
```

In the above example `/dev/sdc` is allocated to account and container rings, whereas `/dev/sdd` and`dev.sde` are devoted to object storage using two different storage policies (object-0 and object-1). Can i remove object-1 from the storage policy as for beta) i think we only support one object storage.

Allocate networks

information required for beta 0

Allocate a Ring -Builder Node

information required fro Beta 0

Specify Ring

The ring maps the logical names of data to locations on a particular disks. There is a separate ring for account database, account database, and individual objects, but each rings works similarly. The rings are built and managed manually by a utility called the ring-builder. The ring-builder also keeps its own builder file with the ring information and additional data required to build future rings.

Using the ring -sepcification key you can specify the ring in the input model. Also, there is an entry for each distinct Swift System. Therefore, in the input model keystone region name in the specified in the **region-name**.

The example files, a ring-specification is located in `config/swift/rings.yml` file.The sample of `rings.yml` is as follows:

```
ring-specifications:
    - region-name: region1
      rings:
        - name: account
          display-name: Account Ring
          min-part-time: 24
          partition-power: 17
          replication-policy:
            replica-count: 3
        - name: container
          display-name: Container Ring
          min-part-time: 24
          partition-power: 17
          replication-policy:
            replica-count: 3
        - name: object-0
          display-name: General
          default: yes
          min-part-time: 24
          partition-power: 17
          replication-policy:
            replica-count: 3
```

The above example shows that the **region1** has three rings as follows:

- An account ring. You must always specify a ring called "account". The **display-name** is informational and not used. . The account ring is used by Swift to store metadata about the projects in your system. In Swift, a Keystone project maps to a Swift account.
  - **min-part-time**- This is the number of hours that the swift-ring-builder tool will enforce between ring rebuilds. On a small system, this can be as low as one hour. The value can be different for each ring.
  - **partition-power** - The appropriate value is related to the number of disk drives you allocate for the account and container storage. We recommend that you use the same drives for both the account and container rings. Hence, the **partition-power** value should be the same.
  - **replication-policy** - The **replication-policy** attribute is used to specify that a ring uses replicated storage. Here, the objects are duplicate copies of an object are made and stored on different disk drives. All replicas are identical. If one is lost or corrupted, the system will automatically copy one of the remaining replicas to restore the missing replica.
  - **replica-count** - Defines the number of copies of objects created. The recommended value for the **replica-count** attribute is 3. That is, three copies of data are kept to ensure redundancy and continued access in the even of component failures
- A container ring. You must always specify a ring called "container". The **display-name** is informational and not used.
- An object ring. You must always specify a ring called "object-0". It is possible to have multiple object rings – these are known are *storage policies*. The **display-name** is the name of the storage policy and may be used by users of the Swift system when they create containers. it allows them to specify the storage policy that the container uses. need to edit this section??????

Ring parameters

You can specify the ring parametes as follows:

1. You must have an account and container ring. The **name** attribute must be "account" and "container" respectively. The display-name attribute can be set to any value and is not used by the system.
2. Select a value for **partition-power**. The appropriate value is related to the number of disk drives you allocate for the account and container storage. We recommend that you use the same drives for both the account and container rings. Hence, the **partition-power** value should be the same. More information about selecting an appropriate value is below.
3. The **replication-policy** attribute must be present for account and container rings. The **replica-count** is normally 3 (i.e, The Object Store will keep three copies of accounts and containers).
4. (not applicable for beta)Determine the number of storage policies you need for object storage. Each storage policy will have a corresponding ring description. You must have at least one storage policy for object storage – hence you will have at least one ring description.
5. Storage policies allow you to differentiate the way objects are stored.. In this release we support only one storage policy, i.e. **object-0**The first storage ploicy must have a **name** of "object-0". The **display-name** must be a single word (character such as underscore or dash are allowed). This is the storage policy name and is visible to your end users.
6. (**Not applicable for beta??**)Subsequent object rings (if they exist) must have a name of "object-<number>" where number starts at "1" and is incremented by one for each storage policy (so "object-1", "object-2", and so on). The **display-name** attribute must be unique.
7. One (only) of the object rings must have the **default** attribute set to "yes". This means that when a user creates a container and does not specify a storage policy, the storage policy will be set to this object ring. (does this hold good with one object storage policy).
8. Pick a **partition-power** value for each storage policy. Each storage policy may have a different partition power. As explained below, the partition power is determined by the number of disk drives allocated to object storage. (does this hold good with one object storage policy).
9. Select *replicated* or *erasure-coded* storage using the **replication-policy** or **erasure-policy** attributes. Currently erasure-coded storage is not recommended for production storage. Do not specify both **replication-policy** and **erasure-policy** items – only one or other is allowed.
10. For replicated storage, set the **replica-count** attribute. It is recommended to set the replica count value to 3. This means that the Object Storage system will store three separate copies of each object. This ensures that even if a disk drive fails or a server is down, you still have other copies of the object,
11. For all rings, set a value for **min-part-time**. This is the number of hours that the swift-ring-builder tool will enforce between ring rebuilds. On a small system, this can be as low as one hour. See later for more information on selecting a value for **min-part-time**. The value can be different for each ring.
12. Optionally set the **swift-regions** and **swift-zones** attribute. The swift-regions attribute applies to all rings in a given Keystone region. The swift-zones can be applied to all rings in a given region or can be specified for a specific ring. See later for more information.

Selecting a Partition Power (for beta do we need to provide this much detailed information??)

When storing an object, the Object Storage system hashes the name. This hash results in a "hit" on a partition (so a number of different object names will result in the same partition number). The partition in turn is mapped to one the available disk drives. In fact, with a replica count of 3, each partition is mapped to three different disk drives. The hashing algorithm used hashes over a fixed number of partitions. The partition-power attribute determines how many such partitions you have. You must select a partition power for a given ring that is appropriate to the number of disk drives you allocate to the ring. This is because:

- If you use a high partition power and have few disk drives, each disk drives will have 1000s of partitions. With too many partitions, audit and other processes in the Object Storage system cannot "walk" the partitions in a reasonable time and updates will not occur in a timely manner.
- If you use a low partition power and have many disk drives, you will have 10s (or maybe only one) partition on a drive. The Object Storage system does not use size when hashing to a partition – it hashes the name. If sizes and names are randomly distributed, you would normally expect a random, even, distribution of sizes within a partition. However, this may not happen (i.e., two neighboring partitions may have different sizes). With many

partitions on a drive, a large partition is cancelled out by a smaller partition so the overall drive usage is similar. However, with very small numbers of partitions, the uneven distribution of sizes can be reflected in disk drive utilization (so one drive becomes full while a neighboring drive is empty).

An ideal number of partitions per drive is 100. If you know the number of drives, you can select a partition power that is close to 100 partitions per drive. However, this is fine if you never plan to add or remove drives from your system. Usually, you install a system with a specific number of drives and add drives as your storage needs grow. However, you cannot change the value of the partition power. Hence you must select a value that is a compromise between current and planned capacity. This is why the partition power selection table below shows a range of drives.

If you are installing a small capacity system, but know you need to grow to very large capacity, you may find that you cannot fit within any of the ranges in the table. if so, you should seek help from Professional Services to plan your system.

There are a few additional factors that can help or mitigate the fixed nature of the partition power, as follows:

- Account and container storage represents a small fraction (typically 1%) of your object storage needs. Hence, you can select a smaller partition power (relative to object ring partition power) for the account and container rings.
- For object storage, you can add additional storage policies (i.e., another object ring). When you have reached capacity in an existing storage policy, you can add a new storage policy with a higher partition power (because you now have more disk drives in your system). This means that you can install your system with a small partition power appropriate to a small number of initial disk drives. Later when you have many disk drives, the new storage policy can have a higher value appropriate to the larger number of drives.

  However, while his technique allows you to continue to add storage capacity you should note that existing containers continue to use their original storage policy. Hence, the additional objects must be added to new containers to take advantage of the new storage policy.

Swift Regions and Swift Zones

tbd