Name: Amjad Alqahtani **Cloud Computing 5573**

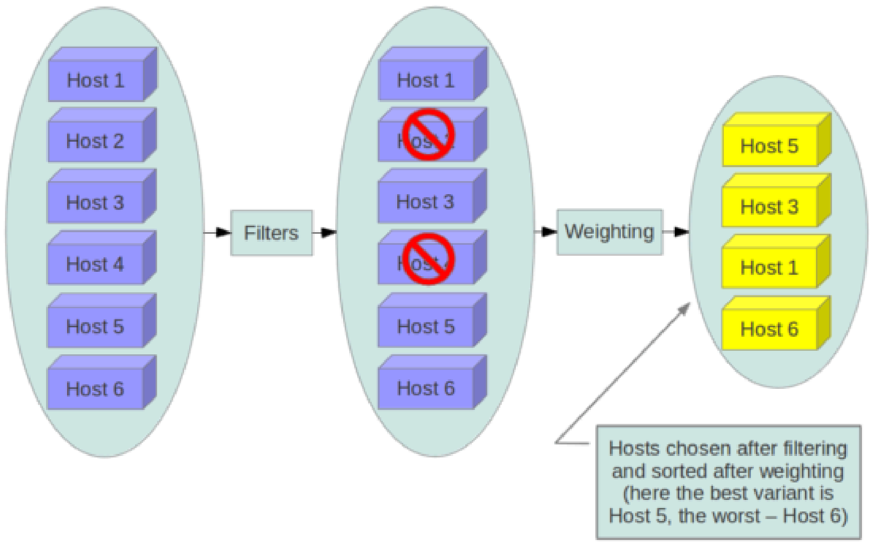
Date: 10/27/2024 **Assignment 3**

1) OpenStack Filter Scheduler:

The filter scheduler decides where to place virtual machines (VMs) on available compute nodes using filters and weighting algorithms to match VM requests with hosts that best meet the specified criteria.

Internally, we have three components called filtering, weighing, and resource allocation. The scheduler uses filters to narrow down the list of potential compute nodes and evaluates each node's characteristics and capabilities against the requirements. Once filtering narrows down the eligible hosts, the scheduler applies weights to the remaining hosts. These weights are calculated based on certain metrics like memory or CPU usage.

Overall architecture is dictated in the figure below. The scheduler generally chooses the node with the highest weight. Then, Filter Scheduler allocates resources on this node and initiates the VM provisioning process.


Host 1
Host 2
Host 3
Host 4
Host 5
Host 6
Filters
Host 1
Host 3
Host 5
Host 6
Weighting
Host 5
Host 3
Host 1
Host 6
Hosts chosen after filtering and sorted after weighting (here the best variant is Host 5, the worst – Host 6)

Examples of filters used by the OpenStack Filter Scheduler.

RAMFilter: Excludes hosts without enough free memory for the requested VM.
DiskFilter: Ensures each host has the required disk space.
ComputeFilter: Verifies that the compute service is running on the host.
AvailabilityZoneFilter: Limits eligible hosts to the specified availability zone.

2) **OpenStack Swift decides where to store an object. Provide a full description of the steps, components and data structures involved.**

OpenStack Swift uses a consistent hashing to map object names to storage nodes. Consistent hashing is doing partitions and ensuring a balanced distribution and redundancy. The consistent hashing has a ring form that helps maintain high availability by directing objects to specific partitions and replicas across nodes.

<u>Key Components:</u>

1. Account Server: tracks metadata for each account
2. Container Server: manages metadata for containers, storing mappings of objects within each container
3. Object Server: handles the actual storage of object data on disk

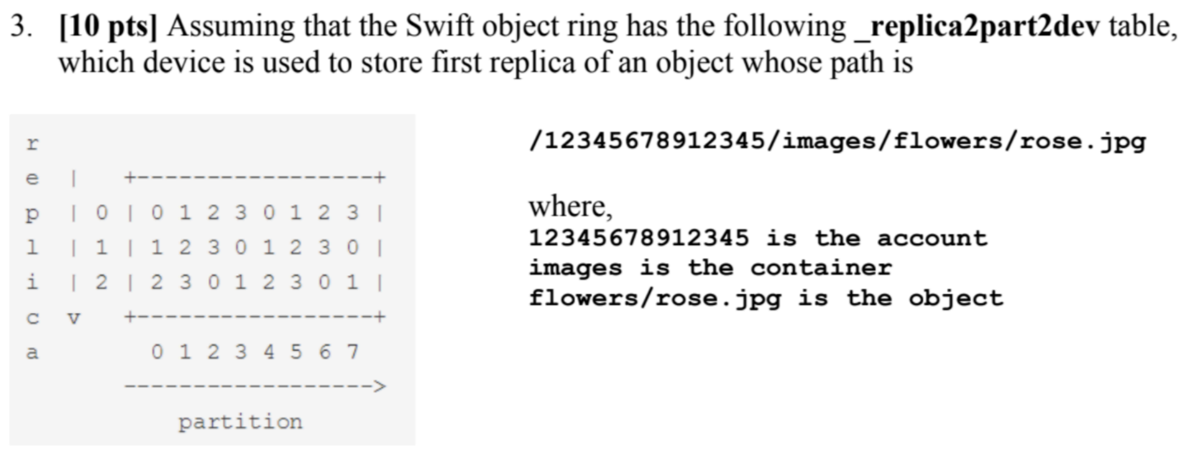
Steps and Data Structures:

Swift uses a consistent hashing ring to determine where to place an object. The ring maps object names to physical storage locations across multiple storage nodes. Each ring is unique for accounts, containers, and objects. The ring divides the storage cluster into partitions, each representing a portion of the entire storage space. The partition count helps balance load and distribute objects evenly across devices. Swift ensures data redundancy by storing multiple replicas of each object across different zones.

When an object is stored, Swift hashes the object's name and its path within the container. This hash value is used with the ring to identify a specific partition and, subsequently, the storage nodes designated for that partition.

Swift then places object replicas in distinct zones, following placement policies to optimize fault tolerance. Failure Handling and Rebalancing: If a node or device fails, the ring adjusts data placement without downtime, reassigning partitions as needed. When new hardware is added, Swift's ring rebalancing redistributes data evenly.

3) **Which device is used to store the first replica of an object.**



3. **[10 pts]** Assuming that the Swift object ring has the following **_replica2part2dev** table, which device is used to store first replica of an object whose path is

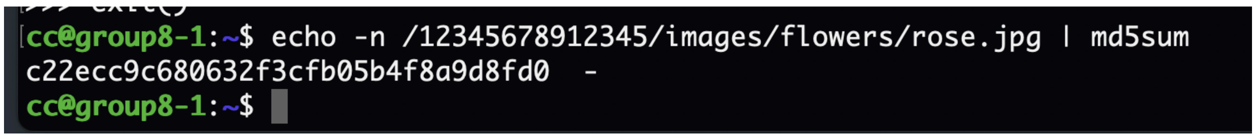
```
r
e  |   +-----------------+
p  | 0 | 0 1 2 3 0 1 2 3 |
l  | 1 | 1 2 3 0 1 2 3 0 |
i  | 2 | 2 3 0 1 2 3 0 1 |
c  v   +-----------------+
a        0 1 2 3 4 5 6 7
       ------------------>
           partition
```

`/12345678912345/images/flowers/rose.jpg`

where,
`12345678912345 is the account`
`images is the container`
`flowers/rose.jpg is the object`

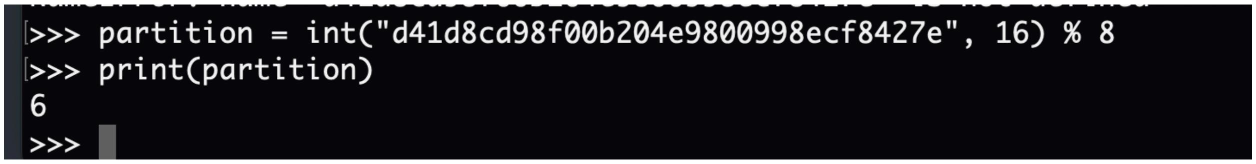
3 replicas, 8 partitions and 4 devices

The string is: c22ecc9c680632f3cfb05b4f8a9d8fd0



```
cc@group8-1:~$ echo -n /12345678912345/images/flowers/rose.jpg | md5sum
c22ecc9c680632f3cfb05b4f8a9d8fd0  -
cc@group8-1:~$
```

Partition 6



```
>>> partition = int("d41d8cd98f00b204e9800998ecf8427e", 16) % 8
>>> print(partition)
6
>>> 
```

Summary:

partition 6:

First Replica: stored on **device2**.

Note: If we have replications for fault tolerance, we need to have three copies, or replications.

Second Replica: stored on **device3**.

Third Replica: stored on **device0**