OpenStack Lab Exercise Answers - Storage (Cinder and Swift)

# Q1: Attach a Cinder volume to a running instance. After attaching, format the volume and mount it within the instance. What is the role of block storage in OpenStack?

Steps to Attach, Format, and Mount a Cinder Volume:  
  
1. \*\*Create a Cinder Volume:\*\*  
 - In Horizon, go to the \*\*"Volumes"\*\* section under \*\*"Project"\*\* and create a new volume by selecting \*\*"Create Volume"\*\*.   
 - Specify the size, name, and type of the volume.  
 - In CLI, use the following command to create a new volume:  
 ```  
 openstack volume create --size <size\_in\_gb> <volume\_name>  
 ```  
  
2. \*\*Attach the Volume to a Running Instance:\*\*  
 - In Horizon, under the \*\*"Volumes"\*\* section, select the newly created volume and click \*\*"Manage Attachments"\*\* to attach it to a running instance.  
 - In CLI, use:  
 ```  
 openstack server add volume <instance\_id> <volume\_id>  
 ```  
  
3. \*\*Connect to the Instance via SSH:\*\*  
 - SSH into the instance using the associated key pair and floating IP.  
 ```  
 ssh -i <key.pem> <user>@<floating\_ip>  
 ```  
  
4. \*\*List Attached Devices:\*\*  
 - Once logged in, verify that the volume is attached by running the following command to list block devices:  
 ```  
 lsblk  
 ```  
 - The new volume should appear as an unformatted block device (e.g., `/dev/vdb`).  
  
5. \*\*Format the Volume:\*\*  
 - Format the volume with a file system (e.g., ext4):  
 ```  
 sudo mkfs.ext4 /dev/vdb  
 ```  
  
6. \*\*Mount the Volume:\*\*  
 - Create a directory to mount the volume:  
 ```  
 sudo mkdir /mnt/myvolume  
 ```  
 - Mount the volume to the directory:  
 ```  
 sudo mount /dev/vdb /mnt/myvolume  
 ```  
  
7. \*\*Verify the Mount:\*\*  
 - Check if the volume is mounted correctly:  
 ```  
 df -h  
 ```  
  
Role of Block Storage in OpenStack:  
  
- \*\*Persistent Storage:\*\* Cinder provides block storage for instances, which is persistent and can survive reboots and instance termination.  
- \*\*Performance:\*\* Block storage is useful for databases, file systems, and applications requiring low-latency access to storage.  
- \*\*Flexibility:\*\* Volumes can be attached to or detached from instances as needed, allowing for scalable and dynamic storage management.

# Q2: Create an object storage container in Swift and upload a file. How does object storage differ from block storage in terms of use cases?

Steps to Create an Object Storage Container and Upload a File in Swift:  
  
1. \*\*Create a Swift Container:\*\*  
 - In Horizon, navigate to the \*\*"Object Store"\*\* section and select \*\*"Containers"\*\*.  
 - Click \*\*"Create Container"\*\*, provide a name for the container, and choose whether to make it public or private.  
 - In CLI, use:  
 ```  
 openstack container create <container\_name>  
 ```  
  
2. \*\*Upload a File to the Container:\*\*  
 - After the container is created, click \*\*"Upload File"\*\* in Horizon and select a file to upload.  
 - In CLI, use the following command to upload a file:  
 ```  
 openstack object create <container\_name> <file\_name>  
 ```  
  
3. \*\*Verify the Uploaded File:\*\*  
 - The file should now be visible in the container and can be accessed via URL if the container is public.  
  
Differences Between Object Storage and Block Storage:  
  
- \*\*Object Storage (Swift):\*\*  
 - \*\*Unstructured Data:\*\* Primarily used for storing unstructured data such as multimedia files, backups, logs, or large data archives.  
 - \*\*Scale and Durability:\*\* Designed for scalability, object storage can handle large volumes of data across distributed environments. Swift automatically replicates objects across multiple nodes for fault tolerance.  
 - \*\*Metadata and Direct Access:\*\* Objects are stored with metadata and accessed via RESTful APIs, making it ideal for cloud-native applications.  
 - \*\*Use Cases:\*\* Backup and archival, static content storage (images, videos), and data distribution.  
  
- \*\*Block Storage (Cinder):\*\*  
 - \*\*Structured Data:\*\* Provides persistent storage for structured workloads like databases or file systems that require direct access to storage devices.  
 - \*\*Low Latency:\*\* Block storage is suitable for applications with low-latency requirements and consistent performance.  
 - \*\*Volume Attachments:\*\* Volumes can be dynamically attached to instances as virtual disks for read/write operations.  
 - \*\*Use Cases:\*\* Running databases, bootable volumes for virtual machines, and high-performance applications.

# Q3: Enable object versioning in Swift. How can you access previous versions of objects stored in Swift?

Steps to Enable Object Versioning in Swift:  
  
1. \*\*Enable Versioning for a Container:\*\*  
 - In Horizon, create or edit a container. Under \*\*"More Actions"\*\*, select \*\*"Set Versioning"\*\* and specify the \*\*"Version Location"\*\*. This creates a versioned container to store object versions.  
 - In CLI, use the following command to enable versioning:  
 ```  
 openstack container set --version-location <version\_container> <container\_name>  
 ```  
  
2. \*\*Upload Objects to the Versioned Container:\*\*  
 - When you upload a new version of an existing object, Swift automatically moves the old version to the version container.  
  
Steps to Access Previous Versions:  
  
1. \*\*List Object Versions:\*\*  
 - In the version container, you can list older versions of objects. Horizon displays the previous versions, and you can restore or download them.  
 - In CLI, use:  
 ```  
 openstack object list <version\_container>  
 ```  
  
2. \*\*Restore or Download a Previous Version:\*\*  
 - To access a specific version, download it from the version container:  
 ```  
 openstack object save <version\_container> <object\_name>  
 ```  
  
How Object Versioning Works:  
  
- \*\*Automatic Versioning:\*\* When versioning is enabled, each time you upload a new version of an object, the previous version is moved to the version container. This allows you to maintain historical versions of objects.  
- \*\*Version Management:\*\* Previous versions can be restored or deleted as needed, providing flexibility for backup and recovery use cases.  
- \*\*Use Cases:\*\* Object versioning is particularly useful for maintaining historical records, managing backups, and recovering from accidental overwrites or deletions.