Robinhood with Managed Lustre

# Prerequistes:

Virtual Network

Azure BLOB storage account

Azure Managed Lustre

Service request to open change logs for robinhood

Bicep deployment template

Install.sh bash script

SSH key

# Robinhood deployment process:

1. From Azure CLI upload both lfazsync.bicep and install.sh files
2. Gather parameters for bicep deployment:
   1. Create storage SAS key via Azure CLI

account\_name=

container\_name=

start\_date=$(date -u +"%Y-%m-%dT%H:%M:%SZ")

expiry\_date=$(date -u +"%Y-%m-%dT%H:%M:%SZ" --date "next month")

az storage container generate-sas \

--account-name $account\_name \

--name $container\_name \

--permissions rwld \

--start $start\_date \

--expiry $expiry\_date \

-o tsv

* 1. Gather subnet id

resource\_group=

vnet\_name=

subnet\_name=

az network vnet subnet show --resource-group $resource\_group --vnet-name $vnet\_name --name $subnet\_name --query id --output tsv

1. Set variables for bicep deployment in CLI. Note: RHEL will use os=almalinux in order to pull the correct packages from github.

resource\_group=

subnet\_id= ‘from step 2

vmsku=Standard\_D32ds\_v4 ‘change as required. Recommend SSD

admin\_user=

ssh\_key=

lustre\_mgs= ‘IP of managed lustre MGS

storage\_account\_name=

storage\_container\_name=

storage\_sas\_key= ‘from step 2

ssh\_port=22

github\_release=v1.0.0 ‘[GitHub - edwardsp/lfsazsync: Lustre to Azure BLOB synchronisation](https://github.com/edwardsp/lfsazsync)

os=almalinux87

1. Execute bicep template

az deployment group create \

--resource-group $resource\_group \

--template-file lfsazsync.bicep \

--parameters \

subnet\_id="$subnet\_id" \

vmsku=$vmsku \

admin\_user="admin\_user" \

ssh\_key="$ssh\_key" \

lustre\_mgs=$lustre\_mgs \

storage\_account\_name=$storage\_account\_name \

storage\_container\_name=$storage\_container\_name \

storage\_sas\_key="$storage\_sas\_key" \

ssh\_port=$ssh\_port \

github\_release=$github\_release \

os=$os

1. Review the installation script status on the newly created VM at /var/log/cloud-init-output.log

# Robinhood policy:

The Robinhood policy can be updated at /opt/robinhood/etc/robinhood.d/lustre.conf

Full robinhood documentation: [robinhood\_v3\_admin\_doc · cea-hpc/robinhood Wiki · GitHub](https://github.com/cea-hpc/robinhood/wiki/robinhood_v3_admin_doc)

Common robinhood commands:

Run all policy triggers – robinhood --run=all

Run archive trigger – robinhood --run=lhsm\_archive

Run release trigger – robinhood --run=lhsm\_release

# Troubleshooting:

1. Review logs:
   1. 'lhsmd': /var/log/lhsmd.log
   2. 'robinhood': /var/log/robinhood\*.log
   3. 'lustremetasync': /var/log/lustremetasync.log
2. Verify /amlfs is mounted to lustre file system
3. Verify robinhood service is running for archive and releasing files
4. Verify lustremetasync service is running for metadata for both files and directories

# Appendix A: Deployment files



# Appendix B: Manual archive/release script



# Robinhood Configuration for data migration

If you're copying data at a rate of 11GB per minute and you want to maintain a 20% buffer on a 48TB Lustre filesystem, you'll first determine how much data can be ingested before hitting the buffer threshold.

**Calculate 20% of the 48TB filesystem:**

20% of 48TB = 0.20 × 48TB = 9.6TB

**Calculate the safety margin before hitting the 20% buffer:**

If you want to initiate the check (and subsequent archival by Robinhood) when you're still above the 20% buffer (to ensure that you never go below it), you might set a safety margin. For instance, you might decide that you want to start checking when you have used up to 75% of the total space, leaving a 25% total free space (which includes your 20% buffer plus an additional 5% safety margin).

75% of 48TB = 0.75 × 48TB = 36TB

**Determine how long it takes to ingest data up to the safety margin:**

You currently have 9.6TB (20% buffer) + 2.4TB (5% safety margin) = 12TB of space before hitting the safety margin.

At a rate of 11GB per minute: Time to ingest 12TB = 12,000GB /11GB/min = 1,091.91 minutes

However, you wouldn't wait for the full duration to start your checks. You'd want to check more frequently to ensure you're never close to hitting the limit. The frequency of checks would depend on how conservative you want to be.

**Determine check interval:**

If you decide you want to check when every 1TB of data is ingested: Check interval = 1,000GB / 11GB/min = 90.91 minutes

For a more aggressive check to be on the safer side, you might decide to check every 500GB: Check interval = 500GB / 11GB/min = 45.45 minutes

Given the rate of data ingestion and the need to ensure you always maintain the buffer, setting a check interval between 45 to 90 minutes would be a reasonable approach. The exact frequency would depend on how conservative or aggressive you wish to be in ensuring the buffer isn't breached.

Example:

30 Minute Archival Rate

5 Minute Release Rate

11GB/Min Copy Rate

If you're copying data at a rate of 11GB per minute and you plan to check for archival every 30 minutes and release every 5 minutes, let's assess the sufficiency of this approach.

**Data Accumulated in 30 minutes**:

At 11GB per minute for 30 minutes: 11GB/min×30min=330GB

So, in the 30 minutes between each archival check, you would accumulate 330GB of data.

**Data Released in 5 minutes**:

If we assume that the archival and release rates are roughly equivalent (though in practice, they might differ), in the 5 minutes between each release check, you'd accumulate: 11GB/min×5min=55gb

Given that you're releasing every 5 minutes, during the 30-minute window between archival checks, you'd have 6 release checks, potentially releasing 6 times the 5-minute accumulation: 6×55gb=330gb

In theory, this matches the data accumulation rate perfectly, suggesting that over the 30-minute interval, you'd release as much data as you ingest.

**Considerations**:

**Archival Rate**: If the rate at which data is archived (and subsequently released) is slower than the ingestion rate, then even with the checks, the filesystem might fill up. Ensure the archival process can handle 55GB every 5 minutes.

**min\_release\_age**: The Robinhood **min\_release\_age** parameter specifies how old a file must be before it's eligible for release. If it's set too high, files might not be released during the 5-minute checks.

**Overhead and Other Activities**: Robinhood and Lustre will have some overhead, and other activities might be happening on the filesystem. This can affect the effectiveness of the checks.

**Safety Margin**: It's always good to have a safety margin. Even if calculations suggest you're breaking even, unforeseen spikes in data transfer or delays in archival could result in space issues.

Given the numbers and your strategy, if the archival and release processes can keep up with the data ingestion rate and there are no major delays or hiccups, this approach should work. However, it's essential to monitor the system closely, especially initially, to ensure the expected behavior is observed and to adjust as necessary.

**Desired "Last Access" Setting**:

If you want Robinhood to consider files for release more rapidly to keep up with the data ingestion rate, you'll need to adjust the "last access" time requirement.

Given that you're checking for archival every 30 minutes, if you'd like files to be eligible for release shortly after they're archived, you might consider setting the "last access" time to be a bit more than 30 minutes, say 40 minutes to 1 hour. This provides a bit of a buffer but ensures that files are rapidly considered for release after archival checks.