Lightbits and Veeam: Backup and Restore Integration

Implementation Guide for Data Protection with Veeam and Lightbits Software-Defined Storage

November 2025

Abstract

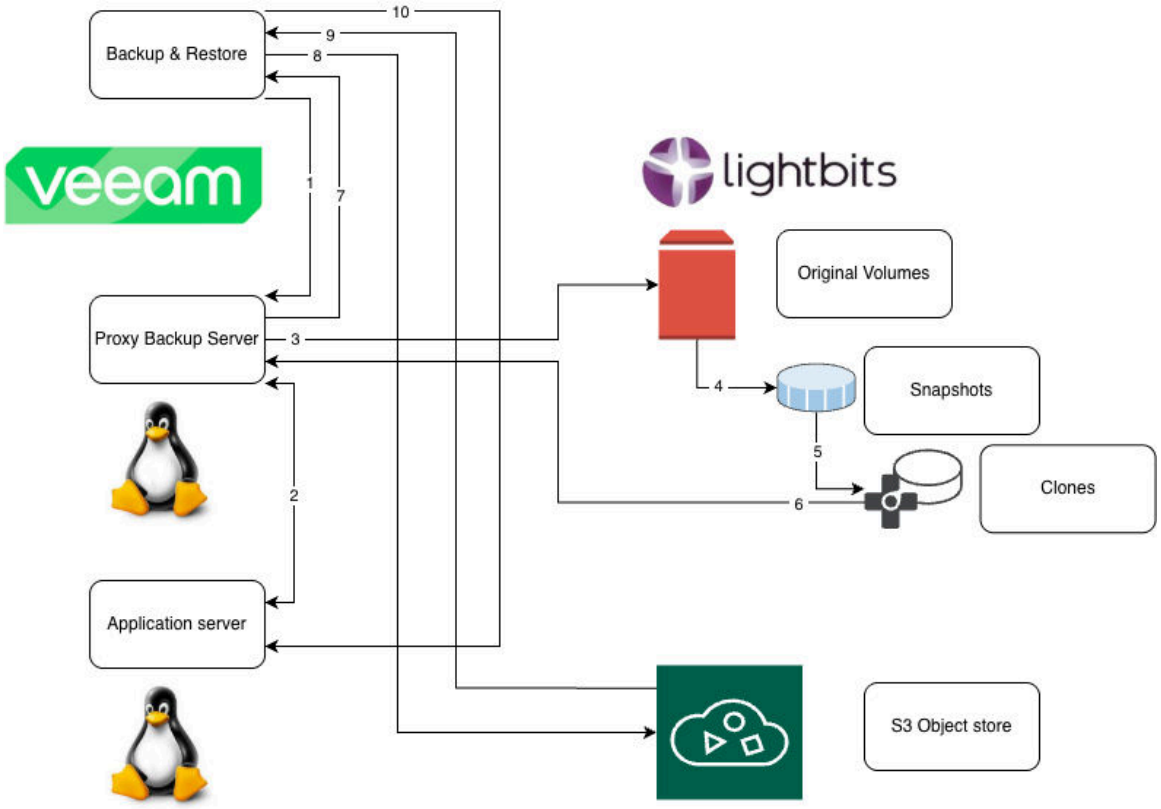
This white paper goes beyond simple installation, guiding you through the essential steps to seamlessly integrate the Veeam Data Platform with your existing server and virtual infrastructure running on Lightbits software-defined storage. The paper illustrates how to execute highly reliable backups and restores with confidence, ensuring that all components work together seamlessly. By following this documented approach, you can protect your critical production workloads and simplify disaster recovery—all while leveraging the extreme performance and resource efficiency of Lightbits block storage. Transform your data protection strategy from a complex challenge into a streamlined, automated process.

Table of Contents

1. Introduction

This white paper details a method for achieving integrated backup and restore functionality using Veeam Backup & Replication. This approach relies on scripting to manage the creation of snapshots and clones from a proxy server connected to the source server. All backup processing and data handling are executed on the proxy server, offloading the application server, while data restoration is performed directly onto the source server. For this implementation guide, Veeam version 12.3 has been used.

The diagram below illustrates the architecture for this integration:



The steps are shown in the diagram above:

1. Veeam reaches out to the proxy server
2. The proxy backup server requests the volume uuid from the application server
3. The proxy backup server connects to Lightbits
4. The proxy backup server creates the snapshots
5. The proxy backup server creates the clones
6. The proxy backup server mounts the clones
7. The proxy backup provides the information to the Veeam server
8. The Veeam server creates the backup on the S3 object store
9. The Veeam server retrieves the data from the S3 object store
10. The Veeam server restores the data directly on the application server

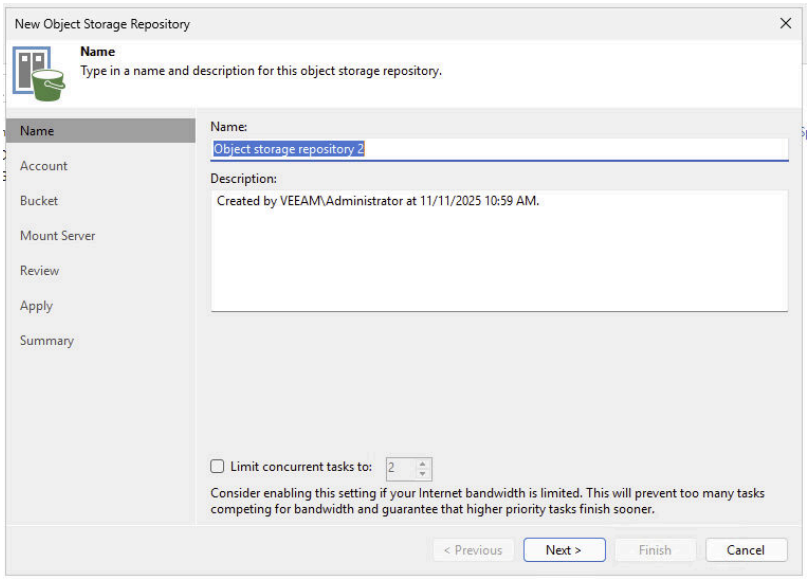
2. Veeam Backup Repository

The initial configuration requires establishing a Backup Repository within Veeam. Since Garage is already deployed within the environment (refer to the "Backup for Kubernetes" white paper for configuration details), it will serve as the S3-compatible target for the Veeam repository.

2.1. Adding an S3-Compatible Backup Repository in Veeam

To configure the Backup Repository using the S3-compatible storage (Garage), perform the following steps within the Veeam Backup & Replication console:

1. Launch the Veeam Backup & Replication application.
2. Navigate to the Backup Infrastructure section by clicking the corresponding option in the bottom-left navigation pane.
3. Initiate the repository creation process by clicking the "Add Repository" button, typically located in the top-left corner.
4. In the ensuing Add Backup Repository dialog box, select Object Storage.
5. On the next screen, choose the S3 Compatible option.
6. This action will open the New Object Storage Repository wizard, where you will continue configuring the connection details for the Garage storage.



2.1.1 S3-Compatible Repository Configuration

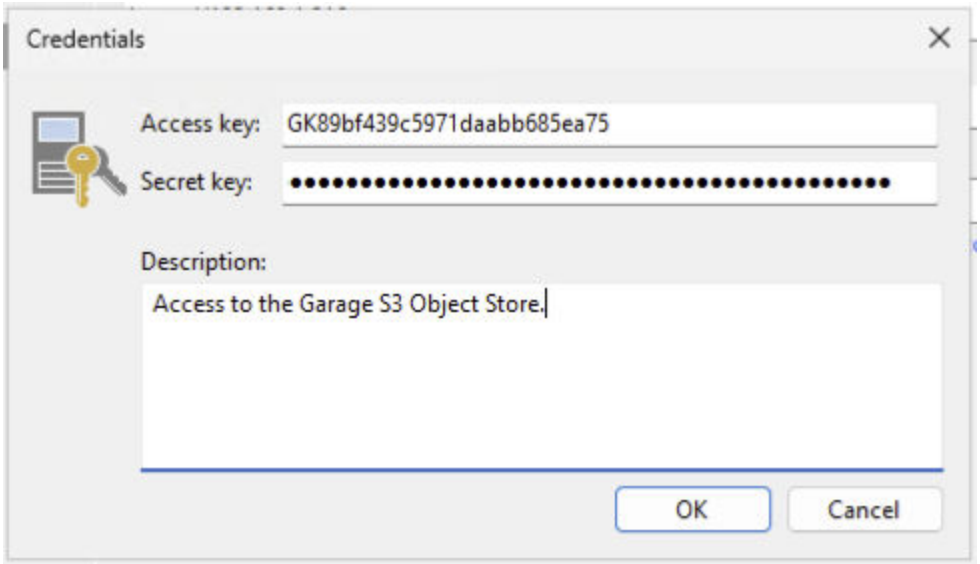
Naming and Service Endpoint

1. In the wizard screen, provide a descriptive name for the new Object Storage Repository and click Next.
2. The subsequent Account screen requires input for the S3 object store connection details:
    - Service Point: Enter the Service Point URL of your S3 object store. In the provided example, this is https://192.168.1.216.
    - Region: Specify the Region configured for your S3 object store. For this example, the value used is Garage.

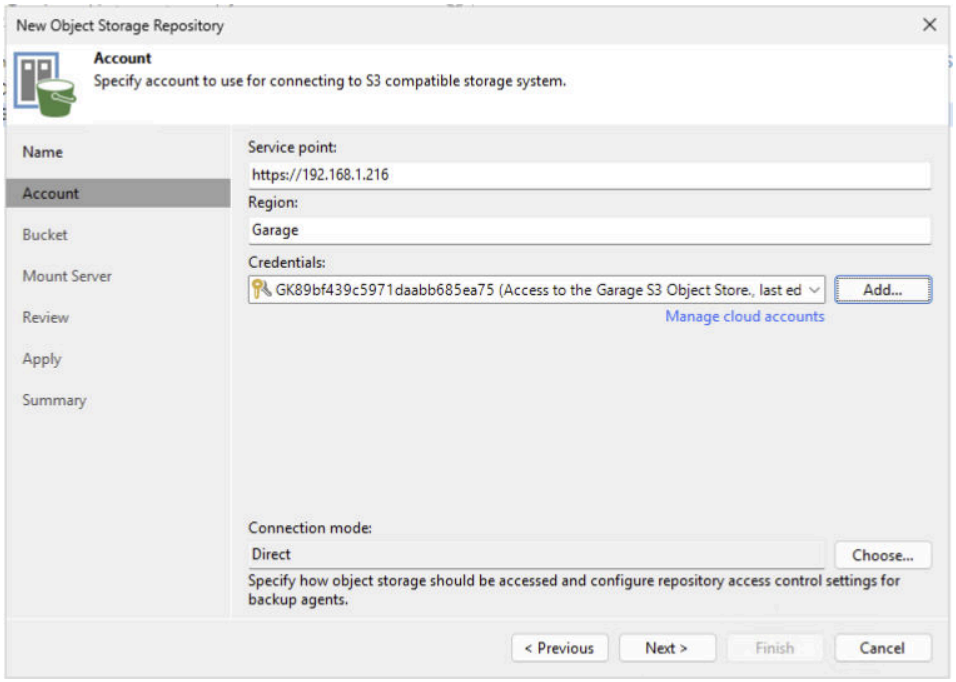
2.1.2 Credential Configuration

Next, you must supply the appropriate Credentials obtained from your S3 object store (Garage):

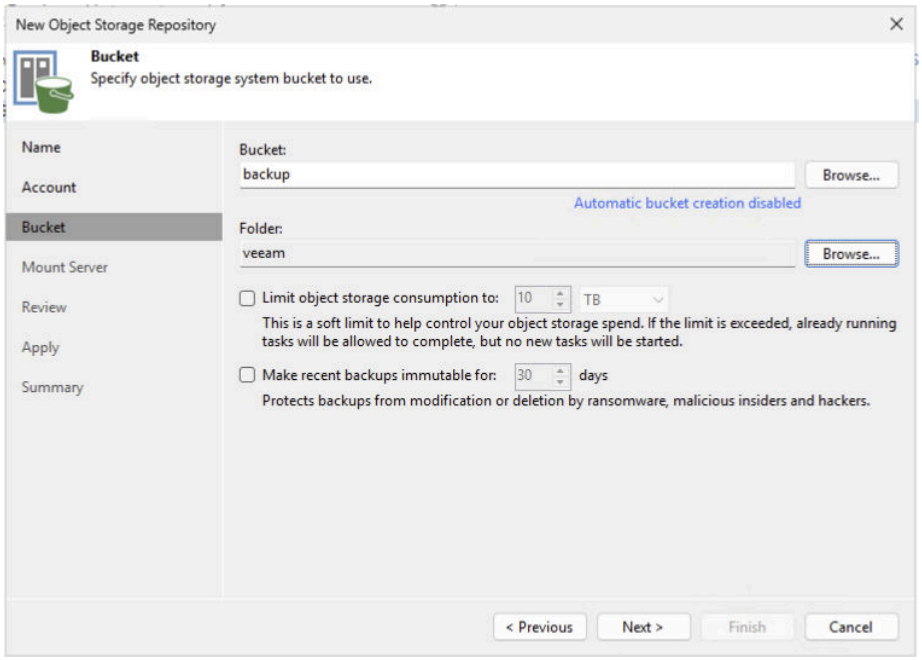
- The required credentials are the Access Key ID and the Secret Access Key.
- In the case of Garage, these keys are typically sourced from a configuration file, such as .awsrc, which contains the defined AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.
- Enter these keys into the respective fields in the Veeam wizard.



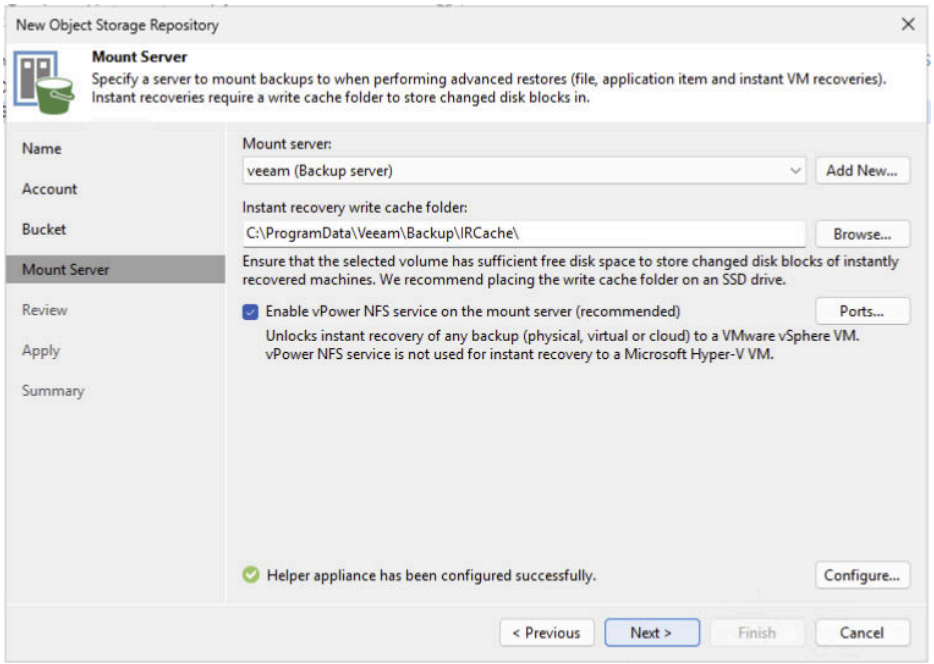
Click on Ok and click on the Next button in the screen below:



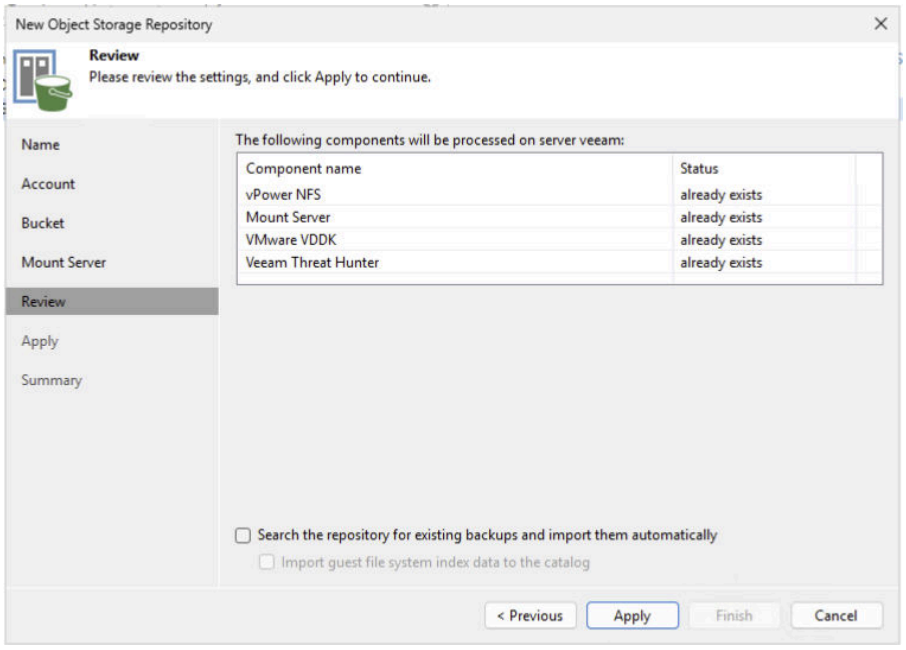
If a certificate needs to be installed, simply follow the installation instructions. The next step is to create a bucket; for this example, we choose 'backup' and 'Veeam' as the folder Name. Click on Next.



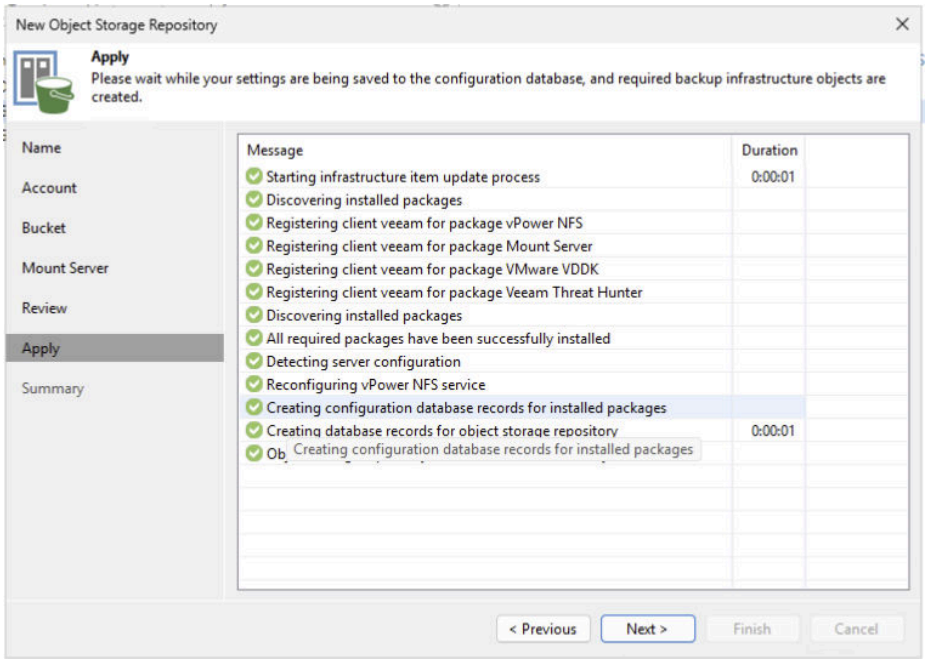
On the next screen, keep the default settings and click on Next.



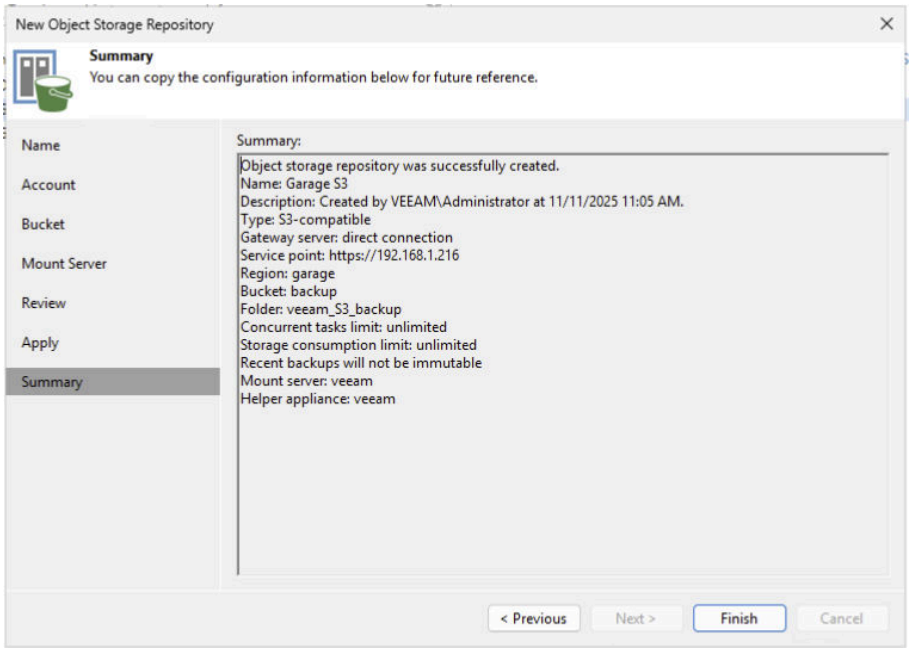
In the review screen, please double-check the components and click 'Apply' to proceed.



Veeam is now constructing the S3 object store, and the output should look similar to this:



Click on Next. The next screen will display a summary of the added Object storage repository. Click on Finish, and object storage has been added to your backup infrastructure.



3. Scripting

For the scope of this implementation guide, the integration between Veeam Backup & Replication and Lightbits block storage is implemented via scripting.

Scripting Focus

1. The initial focus will be on developing the script responsible for creating snapshots and volume clones on the Lightbits block storage system.
2. Following this, a second script will be created to delete these snapshots and clones after the backup operation is completed.

Veeam Server Automation

The Veeam Backup server must be configured with a third and a fourth script within the backup job. The purpose of these commands is to remotely invoke the scripts (snapshot/clone creation and cleanup scripts) on the designated server, facilitating the proxy-based backup of the client-1 application server through the VeeamProxy server.

3.1. Script to Create the Snapshots, Clones and Mountpoints

To enhance readability and comprehension, the script is logically partitioned into multiple sections. The script created is called: **GetReadyForBackup.sh**

The first section of the script is dedicated to identifying the necessary volumes on the Lightbits storage system. This is crucial for creating snapshots and clones.

- **Required Data:** The script must obtain the Volume UUIDs (Universally Unique Identifiers) of the volumes that need protection.
- **Data Source:** These Volume UUIDs must be retrieved directly from the application server, designated as Client-1.
- **Execution Location:** Please note that this initial volume identification script will be executed on the Veeam proxy server, which is the machine where all interaction with the Lightbits volumes and Veeam ultimately occurs.

The second section involves creating clones from the snapshots, and the third section consists in mounting the clones on the Veeam proxy server.

3.1.1. First Section

The script to identify the volume UUIDs for Client-1 and create the snapshots on Lightbits is as follows:

```shell
#!/bin/bash
```

```
# --- Full Pre-freeze Script for Lightbits Snapshot and Volume Creation ---
# REVISED: This script performs the workflow in three phases: Snapshot, Clone,
Mount.
# NOTE: Assumes local host starts with 0 Lightbits NVMe devices.
# Path to the log file for troubleshooting

LOG_FILE="./GetReadyForBackup.log"
exec > $LOG_FILE 2>&1 > "$LOG_FILE"
echo "$(date) - Starting Lightbits snapshot and volume creation workflow."
echo "--------------------------------------------------------------------------"

# --- Configuration for Lightbits API ---
LIGHTBITS_IP="192.168.1.42"
PROJECT_NAME="default"
LB_API_URL="https://$LIGHTBITS_IP:443/api/v2/projects/$PROJECT_NAME"

# Note: You must manually provide your Bearer Token here

LIGHTBITS_JWT="eyJhbGciOiJSUzI1NiIsImtpZCI6InN5c3RlbTpyb290IiwidHlwIjoiSldUIn0.
eyJpc3MiOiIvaG9tZS9kZW1vL2xpZ2h0b3MtY2VydGlmaWNhdGVzL2NlcnQtbGItYWRtaW4ta2V5LnB
lbSIsInJvbGVzIjpbInN5c3RlbTpjbHVzdGVyLWFkbWluIl0sImF1ZCI6IkxpZ2h0T1MiLCJzdWIiOi
JsaWdodG9zLWNsaWVudCIsImlhdCI6MTc1NTE2NDQ0NywiZXhwIjoxNzg2NzAwNDQ3fQ.l2Ak3cjAUP
Gkt_cKkaaQF-RP_Ear6bpARyz9U-ngDr2p57TqWgTLYZrF6coU5XK4q92R5cWt1udILGZL88dASim5m
MB59Qe5OPl48t7kQwJKJf-dcr_CInf4vJtSW1OeohSimKY5QBQD8XieFMbliuZ5K2uPAhLW1rQNmsR4
WGX_ClrlJwksl5CYXTGS9L63aurJKTika2je995XyyNlu__Hgalh0wZCGHUwDzFHBIAJvIL-XRk5sb8
agIjMwlHH1hTCYGCt0NeRImZUa10MHVZj2ZbtN5xXXQYPsDXKjhUmnA6m-5DHmjYgN5FyUOnB1qLoil
YIDlsDsChZeD66iQ"

# --- Configuration for the remote host (Source of NGUIDs) ---
REMOTE_USER="backup"
REMOTE_HOST="client-1"
# --- File paths for UUIDs ---
SNAPSHOT_FILE="./Snapshots"
VOLUMES_FILE="./Volumes"
MOUNTPOINTS_FILE="./Mountpoints"
NGUIDS_FILE="./NGuids"

# --- Script Logic ---
# Clear old output files to ensure fresh data for each run

> "$SNAPSHOT_FILE"
> "$VOLUMES_FILE"
```

```
> "$MOUNTPOINTS_FILE"
> "$NGUIDS_FILE"

# Check for required tools on the local Veeam client machine

if ! command -v /usr/bin/curl &> /dev/null || ! command -v /usr/bin/jq &>
/dev/null; then
  echo "$(date) -  ERROR: Required tools are missing. Please install curl and
jq on the Veeam client machine."
  exit 1
fi
echo "$(date) - Connecting to '$REMOTE_HOST' to get device info (NGUIDs)..."

# Find all device paths for Lightbits devices on the remote host (source
devices)

LIGHTBITS_DEVICES=$(ssh "$REMOTE_USER@$REMOTE_HOST" "sudo nvme list | grep
'Lightbits' | awk '{print \$1}'" 2>/dev/null)

if [ -z "$LIGHTBITS_DEVICES" ]; then
  echo "$(date) -  ERROR: No Lightbits NVMe devices found on '$REMOTE_HOST'."
  echo "$(date) - Please ensure 'nvme-cli' is installed and the SSH user has
passwordless sudo for 'nvme'."
  exit 1
fi

echo "$(date) - Found the following source devices on '$REMOTE_HOST':"
echo "$LIGHTBITS_DEVICES"
echo "------------------------------------------------------------------------"

## Phase 1: Create all the snapshots

echo "$(date) - --- PHASE 1: Starting snapshot creation for all devices --- "
counter=1

# Loop through each device and perform the full workflow

for DEVICE_PATH in $LIGHTBITS_DEVICES; do
  echo "$(date) - --- Processing source device: $DEVICE_PATH ---"

  # Get the NGUID for the current device and format it with sed
```

```
  NGUID=$(ssh "$REMOTE_USER@$REMOTE_HOST" "sudo nvme id-ns $DEVICE_PATH | grep
'nguid' | awk '{print \$NF}' | sed
's/\(.\{8\}\)\(.\{4\}\)\(.\{4\}\)\(.\{4\}\)\(.\{12\}\)/\1-\2-\3-\4-\5/'"
2>/dev/null)

  if [ -z "$NGUID" ]; then
    echo "$(date) -  ERROR: Failed to retrieve NGUID for $DEVICE_PATH."
    continue
  fi

  echo "$NGUID" >> "$NGUIDS_FILE" # Save NGUID of source volume
  echo "$(date) -  Found NGUID for $DEVICE_PATH: $NGUID"

  # --- Create Snapshot via API ---

  SNAPSHOT_NAME="snapshot-$(date +%Y%m%d%H%M%S)-$counter"
  echo "$(date) -  Creating snapshot '$SNAPSHOT_NAME' for NGUID '$NGUID' via
REST API..."

  RESPONSE=$(/usr/bin/curl -s -X POST --insecure -H "Accept: application/json"
-H "Content-Type: application/json" -H "Authorization: Bearer $LIGHTBITS_JWT"
-d "{\"name\": \"$SNAPSHOT_NAME\", \"sourceVolumeUUID\": \"$NGUID\"}"
"$LB_API_URL/snapshots")

  if [ $? -ne 0 ] || [ "$(echo "$RESPONSE" | /usr/bin/jq -r '.state // empty')"
!= "Creating" ]; then
    echo "$(date) -  ERROR: Failed to create snapshot."
    echo "$(date) - API Response: $RESPONSE"
    exit 1
  fi

  CREATED_SNAPSHOT_UUID=$(echo "$RESPONSE" | /usr/bin/jq -r '.UUID // empty')
  echo "$CREATED_SNAPSHOT_UUID" >> "$SNAPSHOT_FILE"
  echo "$(date) -  Success! Snapshot '$SNAPSHOT_NAME' (UUID:
$CREATED_SNAPSHOT_UUID) has been created."
  ((counter++))
done

echo "$(date) -  All snapshots initiated."
echo
"------------------------------------------------------------------------------
-------------------------"
```

3.1.2 Second Section

Now that the snapshots are created, the second session will use the snapshot UUIDs as input to make the clones from. This script describes the second phase of the process.

Shell

```
## Phase 2: Create all cloned volumes
echo "$(date) - --- PHASE 2: Starting cloned volume creation ---"
SNAPSHOT_UUIDS=($(cat "$SNAPSHOT_FILE"))
NGUIDS=($(cat "$NGUIDS_FILE"))

if [ ${#SNAPSHOT_UUIDS[@]} -eq 0 ]; then
  echo "$(date) -  ERROR: No snapshots were successfully created to clone
from."
  exit 1
fi

HOST_NQN=$(sudo cat /etc/nvme/hostnqn 2>/dev/null)

if [ -z "$HOST_NQN" ]; then
  echo "$(date) -  ERROR: Could not retrieve the host NQN."
  exit 1
fi

clone_counter=1

# Loop through snapshot UUIDs and corresponding original NGUIDs

for i in "${!SNAPSHOT_UUIDS[@]}"; do
  CREATED_SNAPSHOT_UUID=${SNAPSHOT_UUIDS[$i]}
  CURRENT_NGUID=${NGUIDS[$i]}
  echo "$(date) -  Waiting for snapshot '$CREATED_SNAPSHOT_UUID' to become
Available..."
  STATE="Creating"
  TIMEOUT=60
  ELAPSED_TIME=0
  while [ "$STATE" != "Available" ] && [ "$ELAPSED_TIME" -le "$TIMEOUT" ]; do
```

```
    SNAPSHOT_STATUS=$(/usr/bin/curl -s -X GET --insecure -H "Accept:
application/json" -H "Authorization: Bearer $LIGHTBITS_JWT"
"$LB_API_URL/snapshots/$CREATED_SNAPSHOT_UUID")
    STATE=$(echo "$SNAPSHOT_STATUS" | /usr/bin/jq -r '.state // empty')
    sleep 2
    ELAPSED_TIME=$((ELAPSED_TIME + 2))
  done

  if [ "$STATE" != "Available" ]; then
    echo "$(date) -  ERROR: Timeout reached. Snapshot did not become Available
within $TIMEOUT seconds."
    exit 1
  fi
  echo "$(date) -  Snapshot is now Available."

  # --- Retrieve Configuration for Clone Volume ---
  echo "$(date) -  Getting original volume size and replica count (NGUID:
$CURRENT_NGUID)..."
  VOLUME_INFO=$(/usr/bin/curl -s -X GET --insecure -H "Accept:
application/json" -H "Authorization: Bearer $LIGHTBITS_JWT"
"$LB_API_URL/volumes/$CURRENT_NGUID")
  ORIGINAL_SIZE=$(echo "$VOLUME_INFO" | /usr/bin/jq -r '.size // empty')
  ORIGINAL_REPLICAS=$(echo "$VOLUME_INFO" | jq -r '.replicaCount // empty')
  ORIGINAL_PROJECT_NAME=$(echo "$VOLUME_INFO" | jq -r '.projectName // empty')
  ORIGINAL_COMPRESSION=$(echo "$VOLUME_INFO" | jq -r '.compression // empty')

    # Check if necessary info was retrieved

    if [ -z "$ORIGINAL_SIZE" ] || [ -z "$ORIGINAL_REPLICAS" ]; then
        echo "$(date) -  ERROR: Could not retrieve original volume size or
replica count for NGUID $CURRENT_NGUID."
        exit 1
    fi

  NEW_VOLUME_NAME="volume-from-snapshot-$(date
+%Y-%m-%d-%H%M%S)-$clone_counter"
  echo "$(date) -  Creating new volume '$NEW_VOLUME_NAME' from the snapshot..."

  # API call uses fetched properties (since you included them)

  VOLUME_RESPONSE=$(/usr/bin/curl -s -X POST --insecure -H "Accept:
application/json" -H "Content-Type: application/json" -H "Authorization: Bearer
```

```
$LIGHTBITS_JWT" -d "{\"name\": \"$NEW_VOLUME_NAME\", \"sourceSnapshotUUID\":
\"$CREATED_SNAPSHOT_UUID\", \"size\": \"$ORIGINAL_SIZE\", \"replicaCount\":
$ORIGINAL_REPLICAS, \"projectName\": \"$ORIGINAL_PROJECT_NAME\",
\"compression\": \"$ORIGINAL_COMPRESSION\", \"acl\": {\"values\":
[\"$HOST_NQN\"]}}" "$LB_API_URL/volumes")

  if [ $? -ne 0 ] || [ "$(echo "$VOLUME_RESPONSE" | /usr/bin/jq -r '.state //
empty')" != "Creating" ]; then
    echo "$(date) -  ERROR: Failed to create a new volume."
    echo "$(date) - API Response: $VOLUME_RESPONSE"
    exit 1
  fi

  NEW_VOLUME_UUID=$(echo "$VOLUME_RESPONSE" | /usr/bin/jq -r '.UUID // empty')
  echo "$NEW_VOLUME_UUID" >> "$VOLUMES_FILE"
  echo "$(date) -  Success! New volume '$NEW_VOLUME_NAME' (UUID:
$NEW_VOLUME_UUID) has been created."
  ((clone_counter++))
done

# Run the discovery client to make sure the volumes are mapped to the client

sudo /usr/bin/discovery-client connect-all -t tcp -a 192.168.1.42 -q
nqn.2014-08.org.nvmexpress:uuid:9062bbb0-9b6a-47b6-be09-a9bd037dbe83
echo "$(date) - Waiting 20 seconds for the new volumes to appear on the local
host..."
sleep 5
echo "$(date) -  All cloned volumes created and connecting."
echo "------------------------------------------------------------------------"
```

3.1.3. Third Section

Now that the clones are created and directly attached to the Veeam proxy server, the third step is to mount the clones in the same order as they were on the Client-1 server.

Shell

```
## Phase 3: Create All Mountpoints (Corrected Logic)

echo "$(date) - --- PHASE 3: Starting Mountpoint Creation ---"
VOLUME_UUIDS=($(cat "$VOLUMES_FILE"))

if [ ${#VOLUME_UUIDS[@]} -eq 0 ]; then
  echo "$(date) -  ERROR: No volumes were successfully created to mount."
  exit 1
fi

NEW_DEVICE_COUNT=${#VOLUME_UUIDS[@]}
mount_counter=1

# --- Robust Waiting Loop ---

TARGET_DEVICE_COUNT="$NEW_DEVICE_COUNT"
ELAPSED_WAIT=0

# Use a filter that reliably counts NVMe namespaces

NVME_FILTER="grep '/dev/nvme[0-9]n[0-9]'"
while [ $(sudo nvme list | eval "$NVME_FILTER" | wc -l) -lt
"$TARGET_DEVICE_COUNT" ] && [ "$ELAPSED_WAIT" -le 60 ]; do
    echo "$(date) - Status: Only $(sudo nvme list | eval "$NVME_FILTER" | wc
-l) of $TARGET_DEVICE_COUNT devices found. Waiting..."
    sleep 5
    ELAPSED_WAIT=$((ELAPSED_WAIT + 5))
done

if [ $(sudo nvme list | eval "$NVME_FILTER" | wc -l) -ne "$TARGET_DEVICE_COUNT"
]; then
    echo "$(date) -  ERROR: Timeout reached. Expected $TARGET_DEVICE_COUNT
devices, but only $(sudo nvme list | eval "$NVME_FILTER" | wc -l) appeared."
    exit 1
fi

echo "$(date) -  All $TARGET_DEVICE_COUNT cloned devices are now visible."

# --- End Robust Waiting Loop ---


# Get the list of the new NVMe devices (which should be ALL devices on this
empty host)
```

```
ALL_LOCAL_NVME_DEVICES=$(sudo nvme list | eval "$NVME_FILTER" | awk '{print
$1}' 2>/dev/null | sort -V)

# --- CRITICAL FIX: Loop through ALL devices found, as they are all new clones
---
# Since ORIGINAL_DEVICE_COUNT = 0, the first device found is the first clone.

for NEW_DEVICE_PATH in $ALL_LOCAL_NVME_DEVICES; do
  MOUNT_DIR="/mnt/Client-1-Vol-$mount_counter"
  echo "$(date) -  Attempting to mount '$NEW_DEVICE_PATH' to '$MOUNT_DIR'
(Clone ${mount_counter})..."
  sudo /usr/sbin/mkdir -p "$MOUNT_DIR"

  # Use 'ro,nouuid' for read-only mounts that ignore duplicate UUIDs
  sudo mount -t auto -o ro,nouuid "$NEW_DEVICE_PATH" "$MOUNT_DIR"

  if [ $? -ne 0 ]; then
    echo "$(date) -  ERROR: Failed to mount '$NEW_DEVICE_PATH'. Check dmesg for
filesystem errors."
    exit 1
  fi

  echo "$MOUNT_DIR" >> "$MOUNTPOINTS_FILE"
  echo "$(date) -  Successfully mounted '$NEW_DEVICE_PATH' to '$MOUNT_DIR'."
  ((mount_counter++))
done

echo "$(date) - All workflows complete. Exiting with success."
exit 0
```

At this point, the snapshots and clones have been created. The clones are connected to the Veeam proxy server and mounted in the same structure as on the Client-1 server.

The original state on Client-1 server is:

/mnt/vol-1
/mnt/vol-2

On the Veeam proxy server, after the script has been executed, it will have the following:

/mnt/Client-1-vol-1
/mnt/Client-1-vol-2

3.2. Script to Unmount the Mountpoints to Delete the Clones and the Snapshots

To enhance readability and comprehension, the script is logically partitioned into multiple sections. The script created is called: **DeleteVolumesSnapshots.sh**

The first section involves unmounting the volumes. The second section involves deleting the snapshots. The third section consists of deleting the volumes (clones). The fourth section is about reporting on the previous three sections.

3.2.1. First Section

The script to unmount the volumes is as follows:

Shell

```
#!/bin/bash
# --- CONFIGURATION ---
# IMPORTANT: Replace these with your actual Lightbits environment details

PROJECT_NAME="default"
LIGHTBITS_MANAGEMENT_IP="192.168.1.42"
LIGHTBITS_API_URL="https://${LIGHTBITS_MANAGEMENT_IP}:443/api/v2/projects/$PROJ
ECT_NAME"

# You MUST define your Authorization token here.
# Replace this placeholder with your actual Bearer Token or API Key.

AUTH_TOKEN="eyJhbGciOiJSUzI1NiIsImtpZCI6InN5c3RlbTpyb290IiwidHlwIjoiSldUIn0.eyJ
pc3MiOiIvaG9tZS9kZW1vL2xpZ2h0b3MtY2VydGlmaWNhdGVzL2NlcnQtbGItYWRtaW4ta2V5LnBlbS
IsInJvbGVzIjpbInN5c3RlbTpjbHVzdGVyLWFkbWluIl0sImF1ZCI6IkxpZ2h0T1MiLCJzdWIiOiJsa
WdodG9zLWNsaWVudCIsImlhdCI6MTc1NTE2NDQ0NywiZXhwIjoxNzg2NzAwNDQ3fQ.l2Ak3cjAUPGkt
_cKkaaQF-RP_Ear6bpARyz9U-ngDr2p57TqWgTLYZrF6coU5XK4q92R5cWt1udILGZL88dASim5mMB5
9Qe5OPl48t7kQwJKJf-dcr_CInf4vJtSW1OeohSimKY5QBQD8XieFMbliuZ5K2uPAhLW1rQNmsR4WGX
_ClrlJwksl5CYXTGS9L63aurJKTika2je995XyyNlu__Hgalh0wZCGHUwDzFHBIAJvIL-XRk5sb8agI
```

```
jMwlHH1hTCYGCt0NeRImZUa10MHVZj2ZbtN5xXXQYPsDXKjhUmnA6m-5DHmjYgN5FyUOnB1qLoilYID
lsDsChZeD66iQ"



# File paths
UUID_FILE_VOLUMES="./Volumes"
UUID_FILE_SNAPSHOTS="./Snapshots"
UUID_FILE_MOUNTPOINTS="./Mountpoints"
UUID_FILE_NGUIDS="./NGuids"
LOG_FILE="./DeleteVolumesSnapshots.log"

# API Endpoints

VOLUME_ENDPOINT="/volumes"
SNAPSHOT_ENDPOINT="/snapshots"

# ================================================================================
# 0. LOG TRUNCATION AND REDIRECTION
# ================================================================================

# The 'exec' command redirects all future stdout (1) and stderr (2) to the log
file.
# The '>' operator ensures the log file is CREATED or TRUNCATED (deleted)
before writing.

exec > "$LOG_FILE" 2>&1
echo "============================================================"
echo " Lightbits Deletion Script Started: $(date)"
echo " All subsequent output is redirected to: $LOG_FILE"
echo "============================================================"

# --- SCRIPT START ---
# Basic check for the token placeholder

if [ "$AUTH_TOKEN" == "<YOUR_ACTUAL_LIGHTBITS_BEARER_TOKEN>" ]; then
    echo "Error: Please update the 'AUTH_TOKEN' variable in the script with
your actual token." >&2
    exit 1
fi

echo "Starting Lightbits REST API deletion process..."
```

```
# Initialize global counters

TOTAL_SUCCESS=0
TOTAL_FAILURE=0

# ================================================================================
# 1. UMOUNT LOOP
# ================================================================================
echo ""
echo "============================================================"
echo "                🚀 Starting umount                          "
echo "============================================================"

# --- Function to display script usage ---
usage() {
    echo "Usage: $0"
    echo "Reads mount points from $UUID_FILE_MOUNTPOINTS and attempts to
unmount them."
    echo "Each mount point should be on a new line in the file."
}

# --- Main script execution ---
# Check if the mount points file exists and is readable

if [[ ! -r "$UUID_FILE_MOUNTPOINTS" ]]; then
    echo "Error: Mount points file '$UUID_FILE_MOUNTPOINTS' not found or is not
readable." >&2
    usage
    exit 1
fi

echo "Attempting to unmount devices listed in '$UUID_FILE_MOUNTPOINTS'..."
echo "------------------------------------------------------------"

# Read the file line by line

while IFS= read -r MOUNT_POINT; do
    # Skip empty lines and lines starting with '#' (comments)
    if [[ -z "$MOUNT_POINT" || "$MOUNT_POINT" =~ ^[[:space:]]*# ]]; then
        continue
    fi
```

```
    # Remove leading/trailing whitespace
    MOUNT_POINT=$(echo "$MOUNT_POINT" | xargs)

    # Check if the mount point is not empty after cleanup
    if [[ -z "$MOUNT_POINT" ]]; then
        continue
    fi

    echo "Unmounting: $MOUNT_POINT"

    # Execute the umount command
    if umount "$MOUNT_POINT"; then
        echo "✅ Successfully unmounted $MOUNT_POINT"
    else
        # Log the error and continue to the next mount point
        echo "❌ ERROR: Failed to unmount $MOUNT_POINT (return code: $?)." >&2
        # Possible reasons include: Device not mounted, resource busy, or
permission denied.
    fi
done < "$UUID_FILE_MOUNTPOINTS"

echo "------------------------------------------------------"
echo "Umount process complete."
#disconnect all the drives
sudo discovery-client disconnect-all
```

3.2.2. Second Section

The second section involves deleting the snapshots.

Shell

```
# ======================================================================
# 2. SNAPSHOT DELETION LOOP
# ======================================================================
echo ""
echo "============================================================"
```

```
echo "               🚀 Starting Snapshot Deletion               "
echo "============================================================"

if [ ! -f "$UUID_FILE_SNAPSHOTS" ]; then
    echo "Warning: Snapshot list file '$UUID_FILE_SNAPSHOTS' not found.
Skipping snapshots."
else
    SNAPSHOT_SUCCESS=0
    SNAPSHOT_FAILURE=0
    SNAPSHOT_TARGET="${LIGHTBITS_API_URL}${SNAPSHOT_ENDPOINT}"
    echo "API Target: $SNAPSHOT_TARGET"
    while IFS= read -r UUID; do
        UUID=$(echo "$UUID" | xargs)
        if [ -z "$UUID" ]; then
            continue
        fi
        echo "--- Attempting DELETE for snapshot UUID: $UUID ---"
        DELETE_URL="${SNAPSHOT_TARGET}/${UUID}"

        # Execute the DELETE request using curl
        # -s: Silent mode, -X DELETE: Method, --insecure: for self-signed certs
(remove if using proper TLS)
        # -o /dev/null -w "%{http_code}": Ensures ONLY the HTTP code is
captured.
        HTTP_CODE=$(curl -s -X DELETE \
            --insecure \
            -H "Authorization: Bearer $AUTH_TOKEN" \
            -H "Content-Type: application/json" \
            -o /dev/null -w "%{http_code}" \
            "$DELETE_URL" )

        # Check the HTTP status code for success (200-204 range)
        if [[ "$HTTP_CODE" -ge 200 && "$HTTP_CODE" -le 204 ]]; then
            echo "✅ Successfully deleted snapshot (HTTP $HTTP_CODE): $UUID"
            SNAPSHOT_SUCCESS=$((SNAPSHOT_SUCCESS + 1))
        else
            echo "❌ Failed to delete snapshot (HTTP $HTTP_CODE): $UUID" >&2
            SNAPSHOT_FAILURE=$((SNAPSHOT_FAILURE + 1))
        fi
    done < "$UUID_FILE_SNAPSHOTS"
```

```
    TOTAL_SUCCESS=$((TOTAL_SUCCESS + SNAPSHOT_SUCCESS))
    TOTAL_FAILURE=$((TOTAL_FAILURE + SNAPSHOT_FAILURE))
    echo "Snapshot Deletion Summary: Success: $SNAPSHOT_SUCCESS, Failed:
$SNAPSHOT_FAILURE"
fi
```

3.2.3. Third Section

The third section is to delete the clones:

Shell

```
# ================================================================================
# 3. VOLUME DELETION LOOP
# ================================================================================
echo ""
echo "============================================================"
echo "               🗄️  Starting Volume Deletion               "
echo "============================================================"

if [ ! -f "$UUID_FILE_VOLUMES" ]; then
    echo "Warning: Volume list file '$UUID_FILE_VOLUMES' not found. Skipping
volumes."
else
    VOLUME_SUCCESS=0
    VOLUME_FAILURE=0
    VOLUME_TARGET="${LIGHTBITS_API_URL}${VOLUME_ENDPOINT}"
    echo "API Target: $VOLUME_TARGET"
    while IFS= read -r UUID; do
        UUID=$(echo "$UUID" | xargs)
        if [ -z "$UUID" ]; then continue; fi
        echo "--- Attempting DELETE for volume UUID: $UUID ---"
        DELETE_URL="${VOLUME_TARGET}/${UUID}"
```

```
        # Execute the DELETE request using curl (same logic as snapshots)
        HTTP_CODE=$(curl -s -X DELETE \
            --insecure \
            -H "Authorization: Bearer $AUTH_TOKEN" \
            -H "Content-Type: application/json" \
            -o /dev/null -w "%{http_code}" \
            "$DELETE_URL" )

        # Check the HTTP status code for success (200-204 range)
        if [[ "$HTTP_CODE" -ge 200 && "$HTTP_CODE" -le 204 ]]; then
            echo "✅ Successfully deleted volume (HTTP $HTTP_CODE): $UUID"
            VOLUME_SUCCESS=$((VOLUME_SUCCESS + 1))
        else
            echo "❌ Failed to delete volume (HTTP $HTTP_CODE): $UUID" >&2
            VOLUME_FAILURE=$((VOLUME_FAILURE + 1))
        fi
    done < "$UUID_FILE_VOLUMES"

    TOTAL_SUCCESS=$((TOTAL_SUCCESS + VOLUME_SUCCESS))
    TOTAL_FAILURE=$((TOTAL_FAILURE + VOLUME_FAILURE))
    echo "Volume Deletion Summary: Success: $VOLUME_SUCCESS, Failed:
$VOLUME_FAILURE"
fi
```

3.2.4. Fourth Section

The fourth section is the summary of the three previous sections:

Shell

```
# ================================================================================
# 3. FINAL SUMMARY
# ================================================================================
echo ""
echo "--- TOTAL DELETION SUMMARY ---"
```

```
echo "Total successful deletions (Snapshots + Volumes): $TOTAL_SUCCESS"
echo "Total failed deletions (Snapshots + Volumes): $TOTAL_FAILURE"

if [ "$TOTAL_FAILURE" -gt 0 ]; then
    echo "⚠️  Some items failed to delete. Check the output above for errors."
    exit 2
else
    echo "🎉 All specified items processed successfully."
fi

# ================================================================================
# 4. CLEANUP SECTION
# ================================================================================
echo ""
echo "============================================================"
echo "                   🧹 Starting Cleanup                      "
echo "============================================================"

# Clean up ./Mountpoints if it was processed
    if rm -f "$UUID_FILE_MOUNTPOINTS"; then
        echo "✅ Cleaned up and deleted mountpoint list file:
$UUID_FILE_MOUNTPOINTS"
    else
        echo "❌ Warning: Failed to delete mountpoint list file:
$UUID_FILE_MOUNTPOINTS" >&2
    fi

# Clean up ./Snapshots if it was processed
    if rm -f "$UUID_FILE_SNAPSHOTS"; then
        echo "   Cleaned up and deleted snapshot list file:
$UUID_FILE_SNAPSHOTS"
    else
        echo "❌ Warning: Failed to delete snapshot list file:
$UUID_FILE_SNAPSHOTS" >&2
    fi

# Clean up ./Volumes if it was processed

    if rm -f "$UUID_FILE_VOLUMES"; then
        echo "   Cleaned up and deleted volume list file: $UUID_FILE_VOLUMES"
    else
        echo "❌ Warning: Failed to delete volume list file:
```

```
$UUID_FILE_VOLUMES" >&2
    fi

# Clean up ./NGUIDS if it was processed
    if rm -f "$UUID_FILE_NGUIDS"; then
        echo "✅ Cleaned up and deleted volume list file: $UUID_FILE_NGUIDS"
    else
        echo "❌ Warning: Failed to delete volume list file: $UUID_FILE_NGUIDS"
>&2
    fi

# Set final exit code

if [ "$TOTAL_FAILURE" -gt 0 ]; then
    echo "⚠️  Script completed with some failures."
    exit 2
else
    echo "🎉 Script completed successfully."
    exit 0
fi
```

3.3. Script on the Veeam Server to Start the Script on Veeam Proxy

The script on the Veeam server that calls the first script (**GetReadyForBackup.sh**) on the Veeam proxy server is as follows:

None

```
# Backup-Client-1.ps1
$SSHCommand = "ssh root@veeamproxy /root/GetReadyForBackup.sh"

# Execute the command and capture the exit code
$ExitCode = (Invoke-Expression -Command $SSHCommand 2>&1 | Select-Object -Last
1).ExitCode
```

```
# Force the PowerShell script to exit with the SSH client's exit code
exit $ExitCode
```

3.4. Script on the Veeam Server to Start the Script to Unmount and Delete the Snapshots

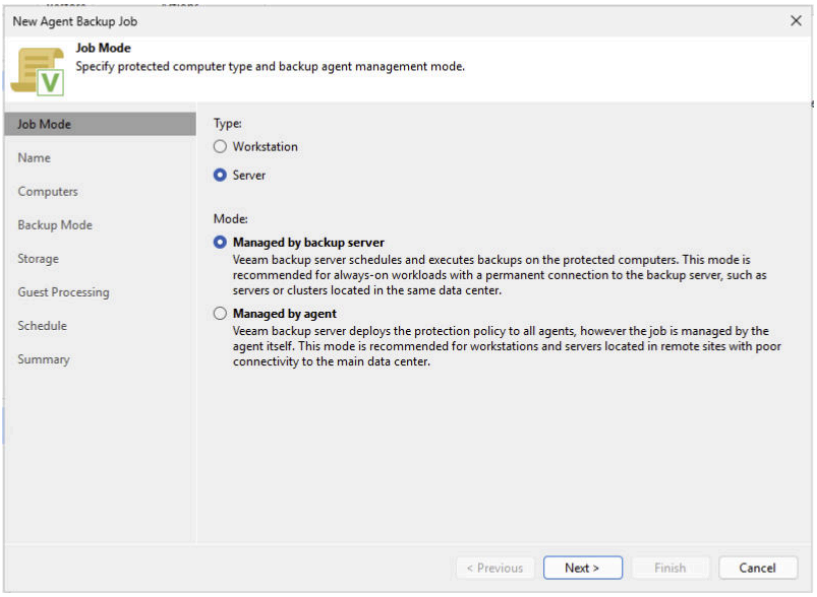
The script on the Veeam server that calls the delete script (**DeleteVolumesSnapshots.sh**) on the Veeam proxy server is as follows:

```
None

# CleanUp-Client-1.ps1
$SSHCommand = "ssh root@veeamproxy /root/DeleteVolumesSnapshots.sh"

# Execute the command and capture the exit code
$ExitCode = (Invoke-Expression -Command $SSHCommand 2>&1 | Select-Object -Last
1).ExitCode

# Force the PowerShell script to exit with the SSH client's exit code
exit $ExitCode
```

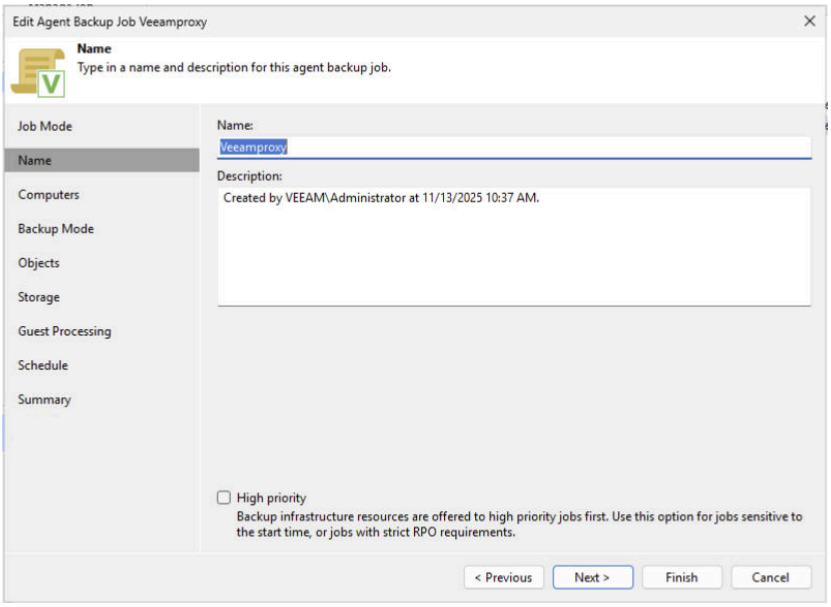
4. Creating the Backup Job on the Veeam Server

It is now time to configure the backup job in Veeam by using the scripts on the local server. In the top menu bar, click on Home. Then, click on the Backup Job icon and select the Linux computer.

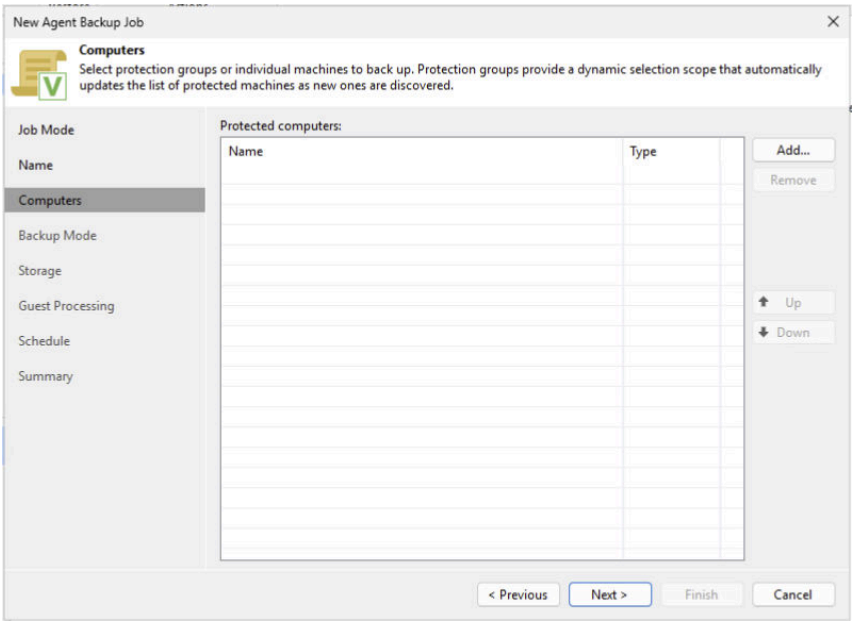
The following screen appears:



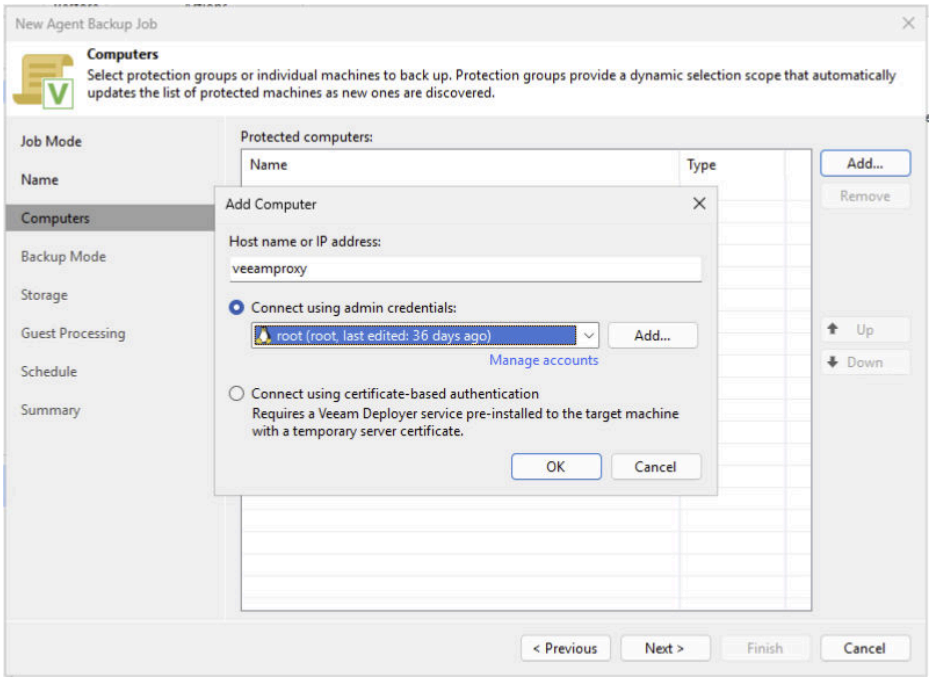
Click on Next. In the following screen, please provide a Name for the backup job. In this example, it is called: Veeamproxy



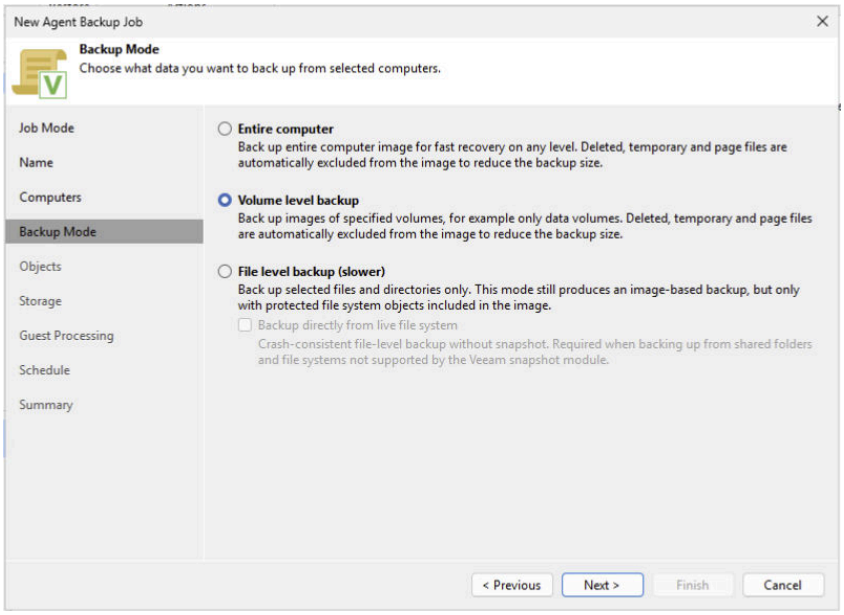
Click on Next. In the following screen, the Veeam proxy server needs to be added.



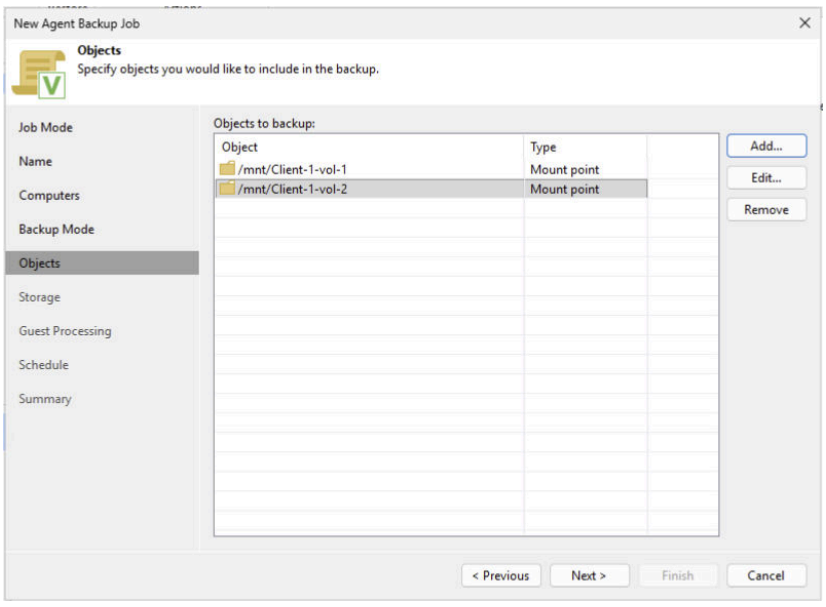
Click the Add button and select 'Individual computer'. Fill in the hostname and the admin credentials to connect to the Veeam proxy server. For this exampl,e the screen will look like this:



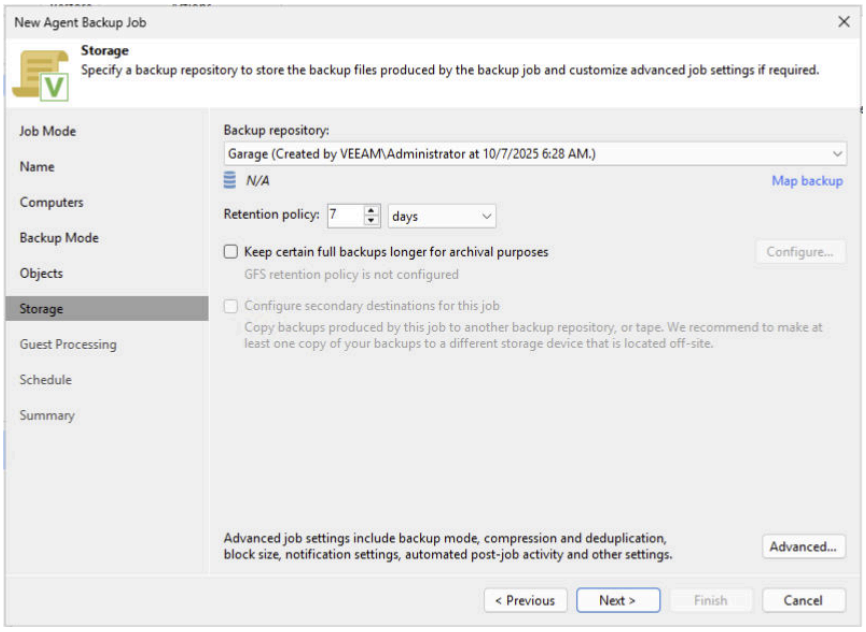
Click OK and then click Next. In the Backup mode screen, the Volume level backup is the one required. Select Volume level backup and click on Next.



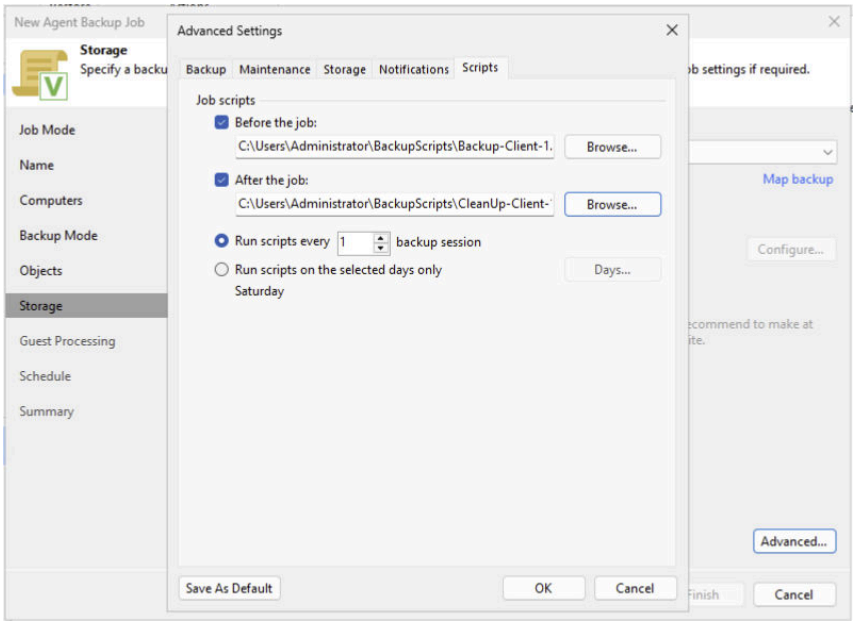
In the Objects screen, the mount points need to be selected from the Veeam proxy server. In the script to create the snapshots, the mount points in this example are: /mnt/Client-1-vol-1 and /mnt/Client-1-vol-2. Click on Add and select Mount point. Add the mount points. In this example, it looks as follows:



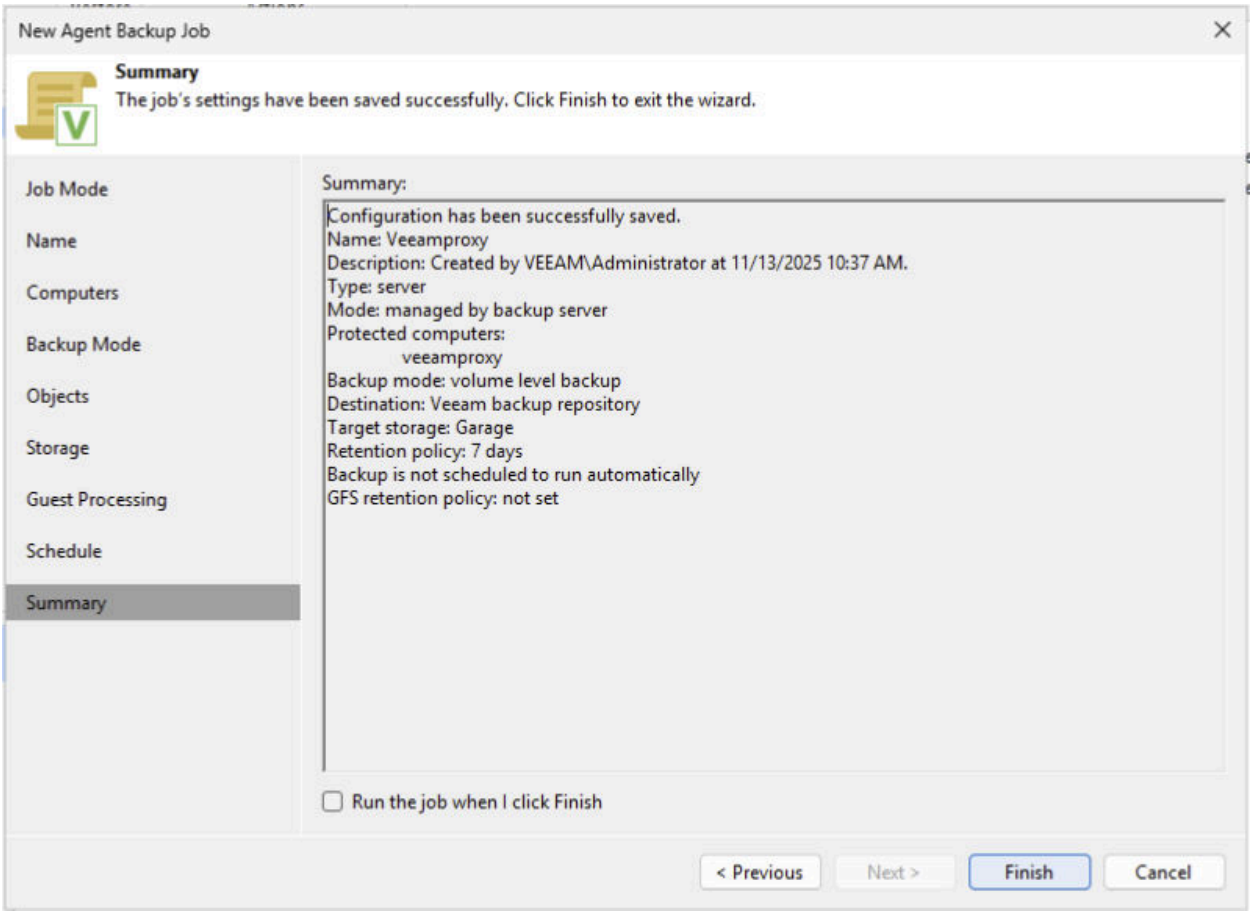
Click on Next. In the Storage screen, the Garage backup repository is directly selected because there is currently only one backup repository. In this screen, click on Advanced.



In the Advanced Settings screen, click the Scripts tab. In the Job scripts section, tick both Before the job and After the job. When the jobs are enabled, you can browse and select the scripts from the Veeam server. In this example, the screen will look as follows:



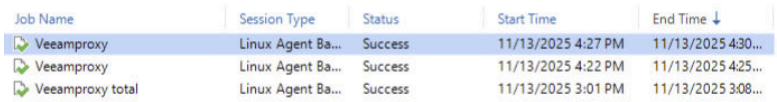
Before the job, the script is executed, which in turn calls the **GetReadyForBackup.sh** script on the Veeam proxy server. After the job is executed, the script is executed, which in turn calls the **DeleteVolumesSnapshots.sh** script on the Veeam proxy server. Click OK, and then click Next to proceed to the Guest Processing screen. On the screen, simply click on Next. In the Schedule screen, you can schedule the job; however, this step is skipped in this example. Click on Apply. The final screen is the summary. The summary for this example looks as follows:

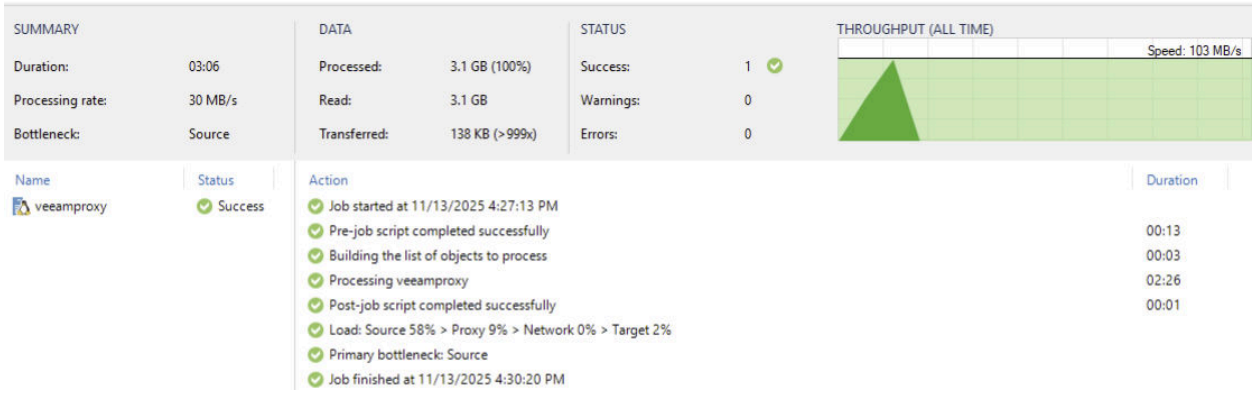


Simply click Finish, and the backup job will be created.

5. Executing the Backup Job

Now that the backup job has been created, hover over the Veeam proxy backup job, right-click on it, and select Start. The job progress will indicate the actions taken. At this point, you should review a successfully completed backup job.

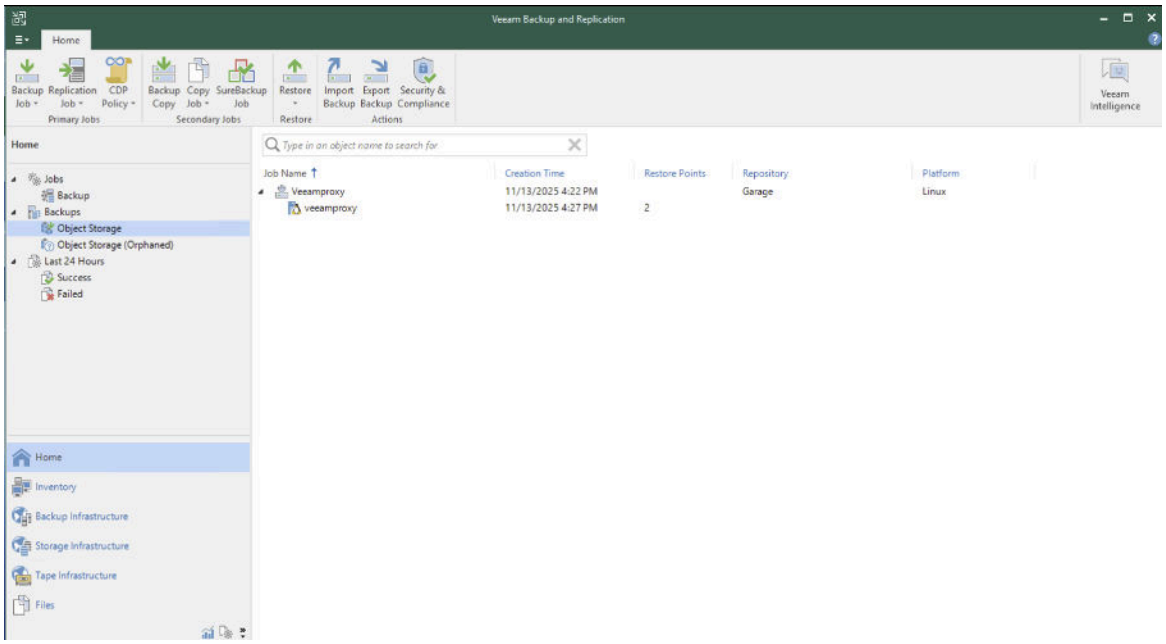




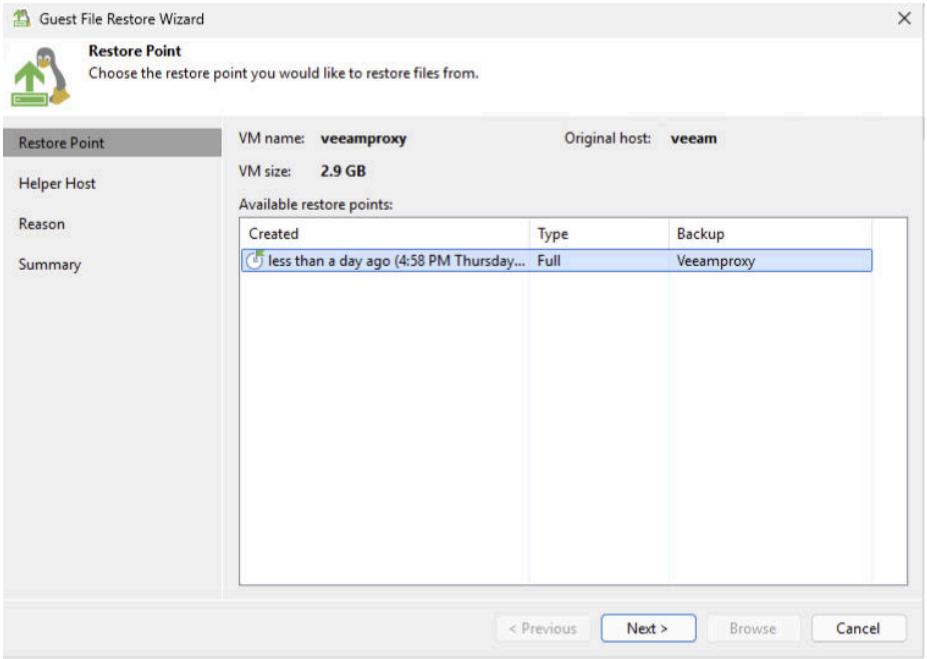
6. Restore the Files Directly to the Original Server Client-1

CommVault allows users to restore files directly from the Veeam proxy server back to the Client-1 server. Guidance follows on how to restore individual files on the original host.

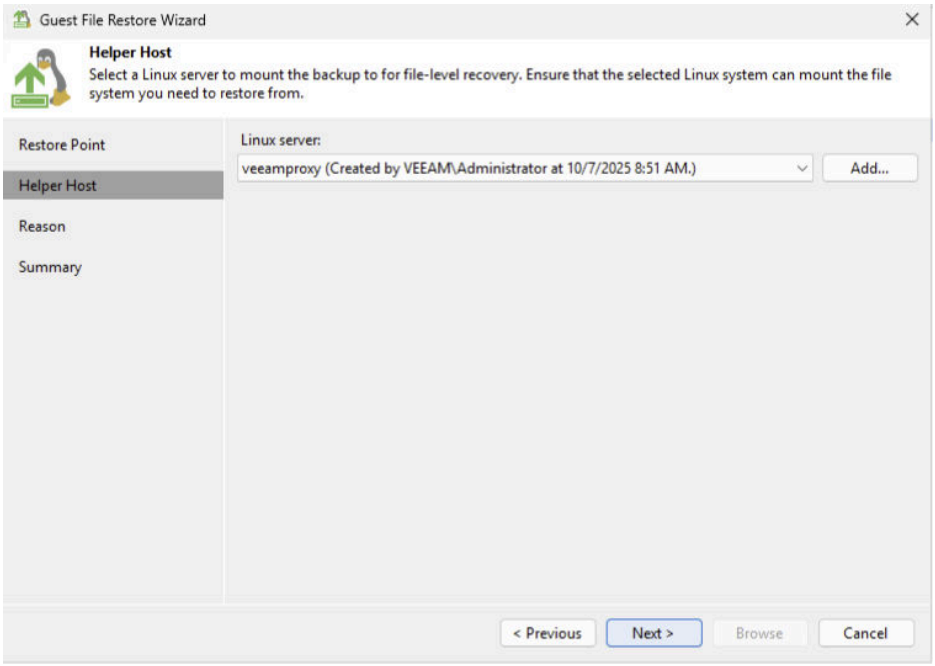
In the top menu bar, click on Home. Then, click "Backups" in the left menu bar, and Object Storage will appear. Click on the Object Storage. In the middle pane, select Veeamproxy. The screen will look like this:



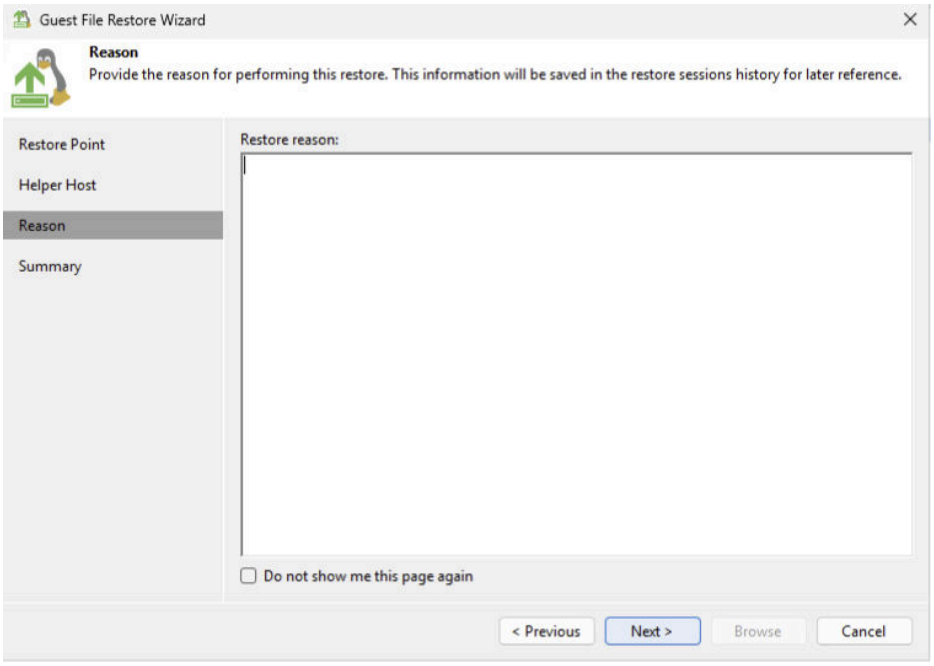
Right-click on the lowest Veeamproxy and select Restore guest files, and click on Linux and other. The following screen appears:



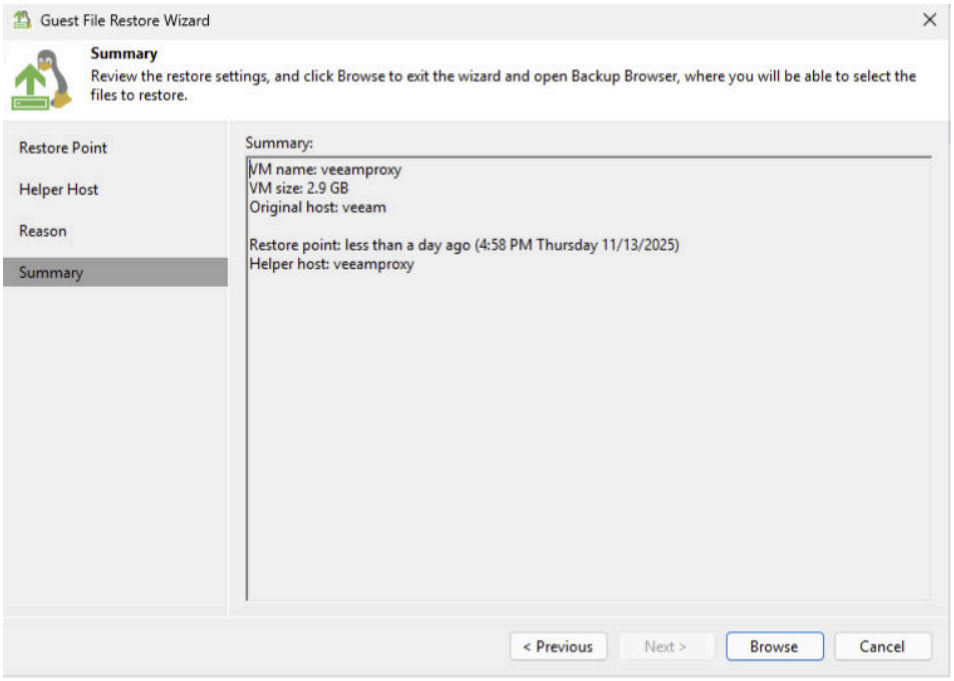
Click on Next. The following screen will appear:



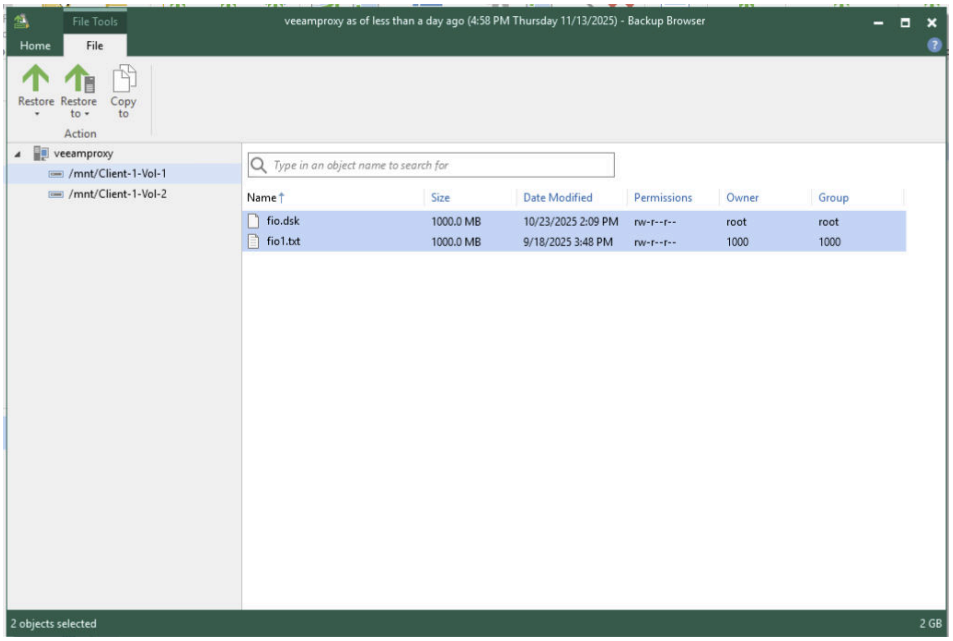
Click on Next. The following screen will appear:



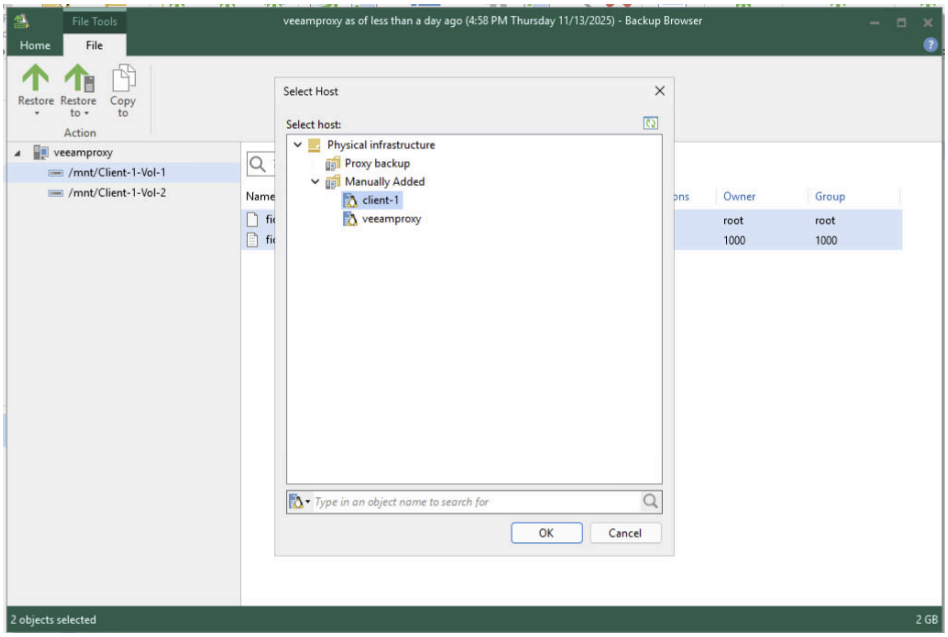
In the “Restore” field, input a reason. Click on Next. The following screen will appear:



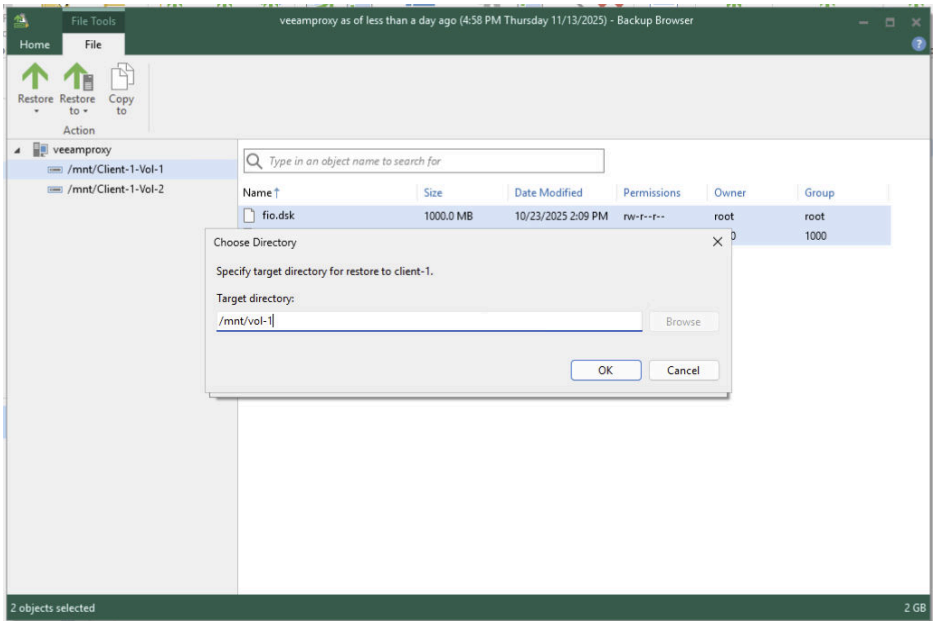
This screen provides a summary. To restore the individual files to another server, click on Browse. The following screen will appear:



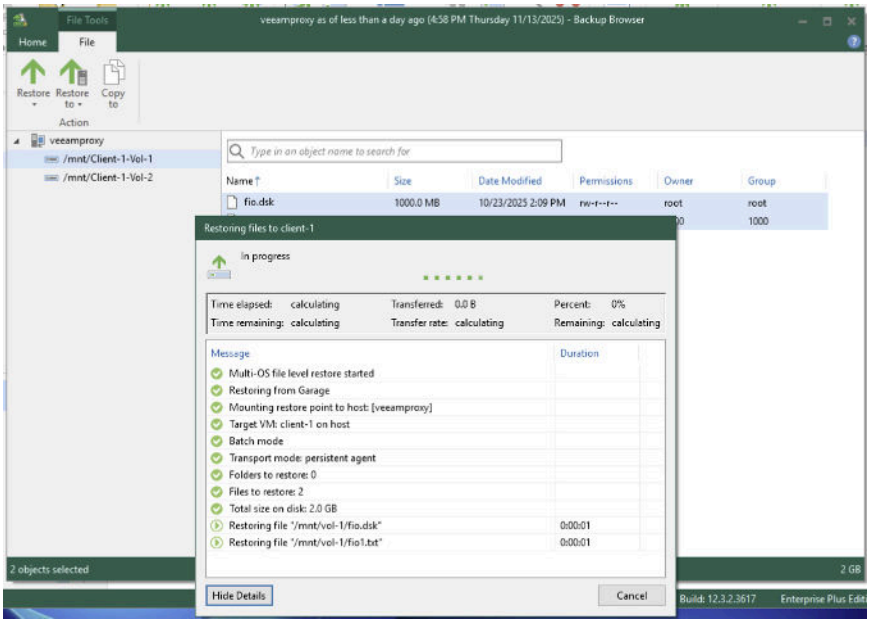
Select the file you want to restore. Click on the "Restore to" button. The following screen appears:



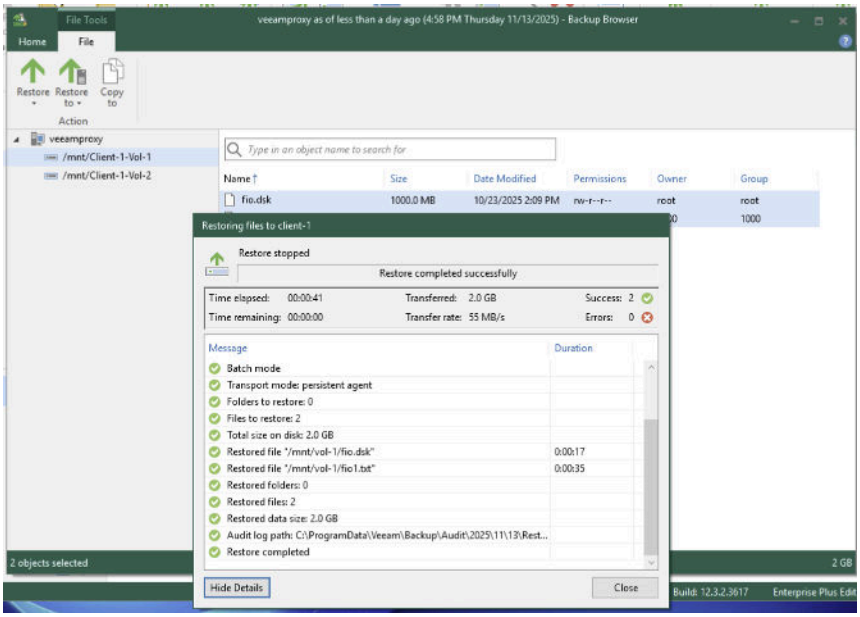
Open the Manual Added and select Client-1. Click the OK button. The target directory needs to be selected, and in this example /mnt/vol-1 is the target directory. Click on the OK button.



Veeam displays the restore progress on the screen below.



Once the restore is complete, click the Close button.



7. Conclusion

Veeam serves as the essential bedrock for modern, comprehensive data resilience, offering an array of capabilities that extend far beyond traditional backup. Its core strengths lie in the Veeam Data Platform's ability to deliver rapid, reliable recovery, cyber resilience, and unparalleled flexibility across virtual, physical, and multi-cloud environments. Veeam provides built-in immutable backups, advanced AI-powered threat detection, and automated orchestration, ensuring crucial capabilities that enable organizations to confidently recover from any incident—especially ransomware. This focus on minimizing downtime and achieving near-zero Recovery Time Objectives (RTOs) makes Veeam the trusted software layer for maintaining business continuity and security in an increasingly volatile digital landscape.

Lightbits Labs revolutionizes data infrastructure by delivering high-performance, software-defined block storage natively designed with NVMe over TCP. Lightbits' unique advantages are centered on its ability to provide consistent, sub-millisecond tail latency and massive throughput over standard, cost-effective Ethernet infrastructure, eliminating the complexity and expense of specialized Fibre Channel or RDMA networks. By enabling the complete disaggregation of compute and storage, Lightbits storage software allows organizations to scale performance and capacity independently. This leads to dramatically improved resource utilization, simplified storage management, and a significant reduction in Total Cost of Ownership (TCO) compared to legacy shared storage arrays.

The integration between Veeam and Lightbits software, as illustrated here, results in a backup and restore solution that delivers superior business value, transforming storage from a passive system into an ultra-fast, high-performance asset. By utilizing Lightbits' ultra-low latency NVMe/TCP storage fabric as the primary backup target, organizations can accelerate every critical operation: backup jobs run faster, and, more importantly, Instant Recovery operations are performed at near-production speeds. This combination ensures that the robust resilience, security, and orchestration of the Veeam Data Platform are optimized by the lightning-fast performance of Lightbits block storage. The result is a future-proof architecture that dramatically tightens RTOs and RPOs, streamlines storage operations, and guarantees instant, reliable recovery, elevating data protection to a true competitive advantage.

About Lightbits Labs

Lightbits Labs® (Lightbits) invented the NVMe over TCP protocol and offers best-in-class software-defined block storage that enables data center infrastructure modernization for organizations building a private cloud or cloud service. Built from the ground up for low consistent latency, scalability, resiliency, and cost-efficiency, Lightbits software delivers the industry's best price-performance value for real-time analytics, transactional, and AI/ML workloads. Lightbits Labs is backed by enterprise technology leaders [Cisco Investments, Dell Technologies Capital, Intel Capital, Lenovo, and Micron] and is on a mission to deliver the fastest and most cost-efficient data storage for performance-sensitive workloads at scale.

www.lightbitslabs.com

info@lightbitslabs.com

US Offices
1830 The Alameda,
San Jose, CA 95126,
USA

Israel Office
17 Atir Yeda Street,
Kfar Saba 4464313,
Israel

LBWP18/2025/11