COHESITY

Version 1.1

July 2024

Cohesity Connector Development Kit

Use Cohesity's CDK to Protect Your Databases

ABSTRACT

DB-Engines ranks more than 300 data sources on its website. As it is not feasible and necessary for Cohesity to add data protection capabilities for all these databases, we currently cover a tiny fraction of the existing database market. UDA provides a way to quickly onboard and support new databases using the Connector Development Kit (CDK).

Table of Contents

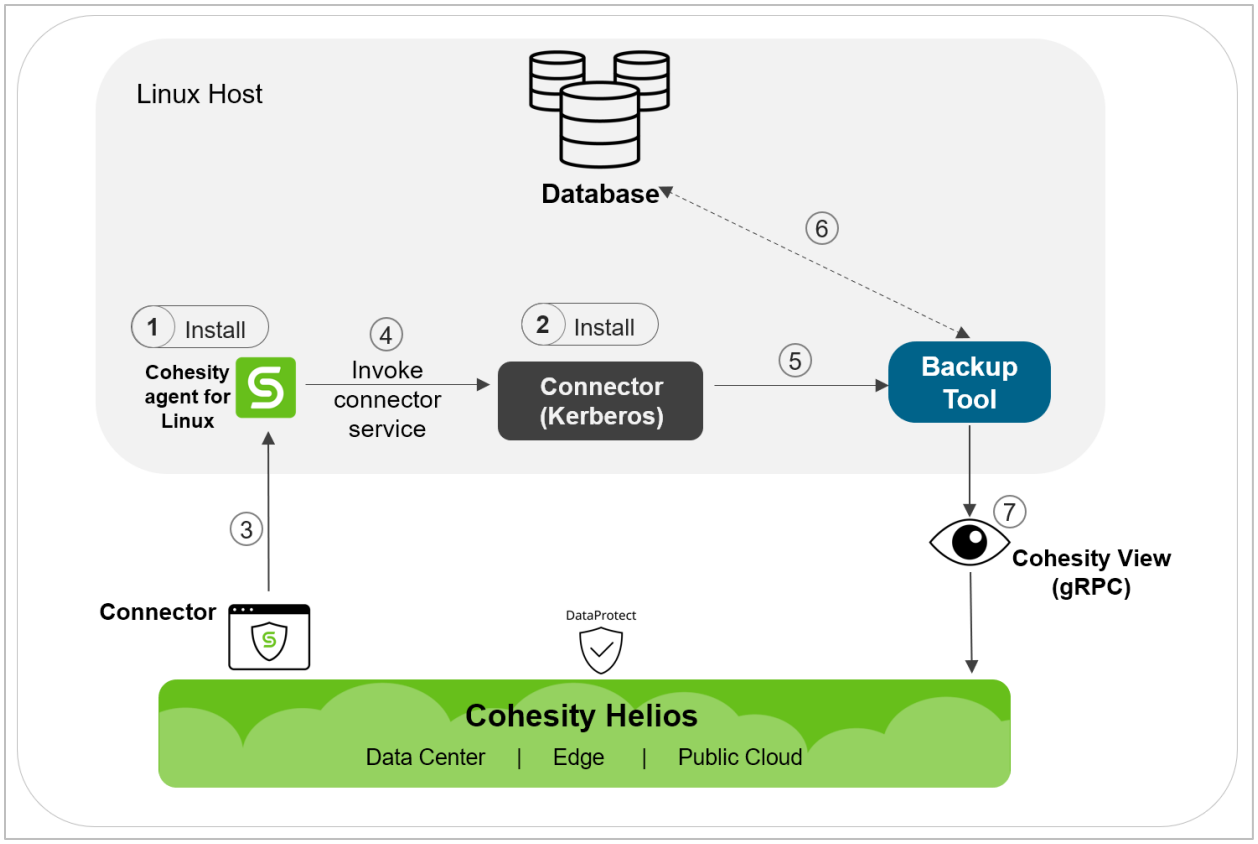
Figures

Tables

What is Cohesity's Connector Development Kit?

The Universal Data Adapter (UDA) framework provides the ability to backup and restore databases hosted on remote Linux machines. It offers specific pre-packaged connectors for backup and recovery of the supported databases. The pre-packaged connectors typically invoke the tools used by the respective databases for backup and recovery operations.

Figure 1: Universal Data Adapter Architecture



To get more information regarding UDA, refer to the Universal Data Adapter product documentation.

The UDA framework provides a unique way to add support for new databases. You can use the Cohesity Connector Development Kit (CDK) to write a connector service for the database for adding support for your database. Cohesity executes the connector service in response to operations like source registration, backups, and restore.

This Connector Development Kit provides an easy way to support protecting single node and multi-node databases.

NOTE: In a multi-node database, some of the nodes act as active nodes on which you perform operations like backup and recovery, while other nodes remain passive until an active node goes down.

Terminology

The following terminology would help you understand the process of writing your database connector using Cohesity CDK:

Cluster Nodes - All the nodes specified in the UI during source registration using UDA. You must install Cohesity Agents and the connector on all the cluster nodes. While using NFS/SMBmounts, the mounts are done on all the cluster nodes. The backup/restore is triggered only on one of the control nodes and may read and write the same view on the same path on all nodes of the cluster.

Non-Control nodes - Nodes on which you can run Source Registration but cannot run Backup and Restore.

Control Nodes- Nodes where you perform Source registration, Backup, and Restore.

Cohesity View: A Cohesity View provides a storage location in a storage domain on the Cohesity cluster and supports various protocols like NFS/SMB/S3 for read/write. To write the backup data to Cohesity, the developer first decides which protocol to use. If you use NFS/SMB, the framework does the mounts for you. If you use S3, the framework passes the S3 credentials.

Fresh View: A new empty view created for writing a full backup.

Live View: An existing view where Cohesity writes previous full and incremental backups. This view is available until the job is deleted for read/write, even when backup/restore is not running.

Non-Live View: Views available for read/write only during backup/restore jobs. Non-live views also have previous incrementals and full backup.

Backup capabilities: Backup capabilities that you can assign to your DB during source discovery.

- *FULL_BACKUP*: Captures all the data in the database at the time of the backup. Backup alone should be sufficient for a restore.
- *INCR_BACKUP*: Captures the changes in the database since the last data backup. This backup, along with previous incrementals and full backup, should be sufficient to do a restore.
- *LOG_BACKUP*: Captures all the transactions since the last log backup/first full backup.
- *AUTO_LOG_BACKUP*: Captures all the transactions since the last log backup/first full backup. Databases trigger AUTO_LOG_BACKUPs (Cohesity is unaware of when they are run).

> **NOTE**: You can configure these in full-backup.sh or incr-backup.sh. This option is for advanced users only.

Protect Your Database Using UDA

Now that we have gone through the UDA architecture and the terminology, let's dive into the Connector Development Kit details. The first step in this process is downloading the template files, which you will use to write your connector.

Download Template

The template package for the CDK is hosted on the Cohesity Github account as a private repository. To request access, reach out to the Cohesity Team at cohesity-api-sdks@cohesity.com.

Once the Cohesity team provides you access to this Github repository, you can clone the file on your machine. This repository contains bash scripts that you need to modify when you write the connector.

Register a Source

You must modify the source registration scripts template to support registering your custom DB on Cohesity and start protecting them.

The available registration scripts are:

1. discover-source.sh
2. verify-source.sh
3. register-utils.sh

Let's go over these scripts to understand the required changes and how you can support registering your database.

Discover Source Script

To discover your source, you need to use discover-source.sh. This script is called during source registration to discover the capabilities and topology of the source.

- The primary objective of this script is to define the **control nodes** on which to run the backup and restore scripts. You must define what type of backups will be supported by this database.

> **PRO-TIP:** discover-source.sh blocks the UI, therefore, time-intensive operations should be avoided here and should be done in the verify-source.sh script.

Let's go over the crucial sections in this script to understand where you need to make changes to support the discovery of your sources.

1. Here is the list of environment variables that will be made available to the connector developer by the UDA framework.

   Table 1: Environment Variables for Discover Source Script

ENVIRONMENT VARIABLES	DESCRIPTION
JQ	Path to the jq library deployed by the Cohesity agent.
USERNAME	The username specified in the UI.
PASSWORD	The password specified in the UI.
SOURCE_ARGS	The source args specified in the UI.
OUTPUT_JSON_FILE	Full file path for the JSON file to write the output.

   NOTE: The JQ library is installed along with the Cohesity agent on your cluster nodes.

2. The shell variable CLUSTER_NODES contains the list of nodes entered in the UI.

   NOTE: You don't need to initialize the shell/environment variables. The UDA framework creates and adds these variables to all other scripts for you to use. This way, you just need to focus on using the appropriate variable for your use case.

3. In the **discoverSource** function, the functions **prepareOutput** and **initialize** are called to read the input and initialize the environment/shell variables and prepare the output for this script. Check out the UDA developer Deep Dive section to know more about these functions.

```
function discoverSource {

  # prepareOutput and initialise must be called prior to setting the
fields.
  # prepareOutput Initialises output json file.
  prepareOutput
  initialise
```

4. The next step is to write the logic to compute the control nodes' list (**controlNodes** variable) from the CLUSTER_NODES shell variable. For example, the developer may need to query their DB to get the list of Control nodes, verify that it is reachable, and then populate the **controlNodes** variable.

   NOTE: Control nodes list can be a subset or the same as the cluster nodes list.

5. The connector developer also needs to decide the backup capabilities for your database by defining that in the capabilities variable.

6. The backup capabilities of this source can be FULL_BACKUP, INCR_BACKUP, LOG_BACKUP, AUTO_LOG_BACKUP.

   NOTE: You cannot set LOG_BACKUP and AUTO_LOG_BACKUP together.

7. The developer will need to write the logic to give your database a name and a UUID that is unique regardless of the cluster node from where the scripts are run.

```
# Set fields in the output json file.
  controlNodes=('10.2.xxx.88' '10.2.xxx.89')
  capabilities=('FULL_BACKUP' 'LOG_BACKUP')
  sourceName='OracleSalesDb'
  sourceUUID='123-456-789'
```

8. The next step is to set the variables computed in the previous step by calling the helper functions. See the UDA developer Deep Dive section to know more about these helper functions.

```
setControlNodes "${controlNodes[@]}"
setCapabilities "${capabilities[@]}"
setSourceInfo $sourceName $sourceUUID
```

9. There are some optional settings that you can specify when discovering a source. These optional variables are:

   a. **setFreshFullBackupViewFalse**: Set this variable to write full backups to a clone of the view of the last data backup instead of a fresh/empty view. By default, full backups are always written in a fresh view.

      NOTE: You can use **setFreshFullBackupViewFalse** only if you have not used **setLiveDataViewTrue**.

   b. **setLiveDataViewTrue**: Set this variable to use live data view for full backups.

      NOTE: When you set **setLiveDataViewTrue**, full backups will not be written to a fresh/empty view; instead they will be written to the same live view where all previous backups were written.

   c. **useS3View**: Use this variable to use s3 views for backup and restore.

Verify Source Script

The Verify Source script verifies the credentials provided in the UI by connecting to your database. You can check control nodes reachability, see whether Hostnames are resolvable, check whether DB is up, and more.

TIP: You can perform all time-intensive connectivity checks using the Verify Source script.

1. This script has access to the same environment and Shell variables as the Discover Source script.

2. The verifySource function is the only important section to change and add your source verification code. If the verification is unsuccessful for any reason, set the exit status to 1.

```
BASE_DIR=$(dirname "$0")

# Import some helper functions.
source $BASE_DIR'/register-utils.sh'

# Verify if the db is up and in the required state.
# Do exit 1 on failure
function verifySource {

  # initialise must be called first to set the input variables.
  initialise

  echo "Verifying the source"
  # eg, mysql -u $USERNAME -D dbname -p $PASSWORD -e "select Col1 from
Table1
  #   where Condition;"

}
```

Back Up Your Databases

To perform backups (full, incremental, and log) you must edit the following scripts. Use your DB backup tool in these scripts and then write the backups to Cohesity mounted on the control nodes.

The available backups scripts are:

1. full-backup.sh
2. incr-backup.sh
3. log-backup.sh
4. cancel-backup.sh
5. cleanup-backup.sh

Let's understand each of these scripts so that you can modify these scripts and start protecting your DBs.

Run Full Backup

This script is called for a full backup. You can also perform automatic log backups if it is configured during source registration by setting AUTO_LOG_BACKUP in the capabilities. Let's dive into the script and understand where you can make changes to support backing up your DB.

1. This script lets you access environment/shell variables that you can use to perform full backup. Similarly, these variables are initialized by the UDA framework for all other scripts. Below is the list of environment variables and their descriptions.

Table 2: Environment Variables for Full Backup Script

ENVIRONMENT VARIABLES	DESCRIPTION
JQ	Path to the jq library deployed by the Cohesity agent.
USERNAME	The username specified in the UI.
PASSWORD	The password specified in the UI.
JOBID	The job ID used to identify this job. Job ID helps identify any async processes linked to this job. When a job is deleted/canceled, the same job ID will be passed, to clean up such processes.
BACKUP_ARGS	The backup args specified in the UI.
RETENTION_MINS	Use the retention duration for full/incremental backups to drop data older than this duration from the data view.
PARALLEL_STREAMS	The number of parallel streams that can be used to write per mount directory/vip.
AUTO_LOG_BACKUP_ENABLED	Use this to specify whether automatic log backups are enabled (1 if true and 0 if false).
AUTO_LOG_BACKUP_RETENTION_MINS	The retention duration for automatic log backups. Use this to drop data older than this duration from the log view. Set it if automatic log backup is enabled.
PULSE_LOGS_FILE	Full file path. Append the logs to display on UI to this file.
S3_ENDPOINT	S3 endpoint, e.g. https://10.2.145.xx:3000, if using S3 view type (**useS3View** in discover-source.sh)
S3_ACCESS_KEY	S3 access key.
S3_SECRET_KEY	S3 secret key.

2. The UDA framework will also initialize some shell variables, which include information about the Cohesity view name and other information needed to write the backup to Cohesity. Below is the list of shell variables.

Table 3: Shell Variables for Full Backup Script

SHELL VARIABLES	DESCRIPTION
DATA_VIEW_NAME	The data view name (Used by non-mount option).
DATA_VIEW_MOUNT_DIRS	The array of directory paths to mount data view if you have selected the mount option.
VIEW_VIPS	The VIPs of the cluster views (used by the non-mount option).
OBJECTS	The array of object names to back up.

3. The following shell variables are set if you enable automatic log backup.

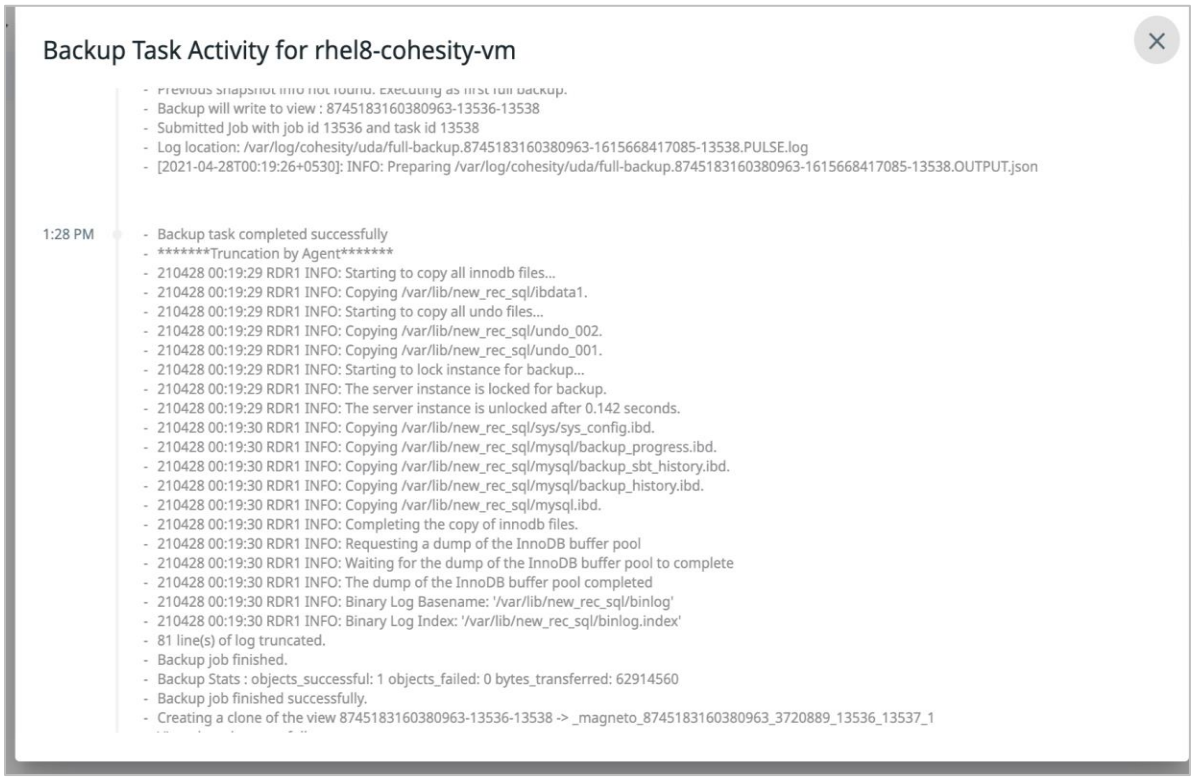
Table 4: Shell Variables for Full Backup with Automatic Log Backup

SHELL VARIABLES	DESCRIPTION
LOG_VIEW_NAME	The log view name (used by non-mount option).
LOG_VIEW_MOUNT_DIRS	The array of directory paths to mount log view if you have selected the mount option.

4. For backup operation logging in the UI, you can write logs to the PULSE_LOGS_FILE. The UI reads this file and displays it as part of backup operation logs. You can simply append logs to the PULSE_LOGS_FILE as follows:

```
echo 'This log goes to UI' >> $PULSE_LOGS_FILE
```

Figure 2: Pulse Log on the UI



IMPORTANT: The PULSE_LOGS_FILE only supports 100 logs. If more logs are written to it, the UI truncates the logs.

5. In the **startBackup** function, the **prepareOutput** and initialize functions are called in the beginning to initialize the input, output, and the environment/shell variables for this script so that the developers can start using it to perform the backup operation.

```
function startBackup {

  # prepareOutput and initialise must be called first.
  # Prepares the output file and sets some variables.
  prepareOutput
  initialise
```

6. The next step is to perform the actual backup for all the objects. You can increase the progress percent based on backup progress.

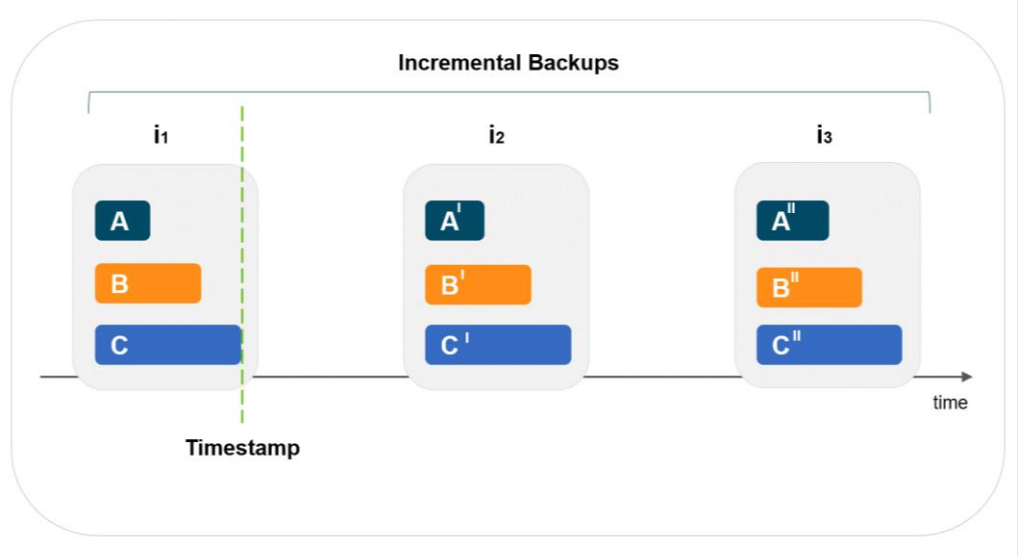
```
##### Do Backup #####
  # eg, mysqldump --databases ${OBJECTS[0]} > ${DATA_VIEW_MOUNT_DIRS[0]}
  for i in {1..10}
  do
    setProgressPercent $i'0'
    echo 'log'$i >> $PULSE_LOGS_FILE
    sleep 5
  done
  ####################
```

7. Once the backup is successfully completed, set the progress to 100%, set the backup timestamp, stats, and success/failure of the backup task. The UDA framework displays the timestamp and stats in the UI.

   a. **setDataBackupTime**. Timestamp of the backup you are creating. Any data that is changed, added, or deleted after this time will not be available for recovery from this backup.

      As a rule of thumb, If multiple objects are protected, the timestamp should be set equal to the time reported for the object backup run that was completed last.

      For example, if three objects — A, B, and C — are protected in a full backup, then: setDataBackupTime = maxTimestamp(A, B, C)



   b. **setStats**: This function sets the number of successfully protected and failed objects and the bytes transferred during the backup.

   c. **setStatusSuccess**: This function sets the status for the backup operation. Other operations are **setStatusPartialSuccess** and **setStatusFailed**.

```
# After backup is complete, set the progress percentage to 100.
  setProgressPercent '100'

  # Data backup timestamp for this full backup in epoch seconds.
  # args -> Backup time in epoch seconds
  setDataBackupTime '1602674625'

  # Set stats.
  # args -> objects successful, objects failed, bytes transferred
  setStats '5' '0' '50000'

  # Set status.
  # If successful -> setStatusSuccess
  # If partial success, that is only some objects were successful ->
  #   setStatusPartialSuccess
  # If failed -> setStatusFailed,
  setStatusSuccess
```

Incremental Backup Script

This script helps perform an incremental backup. This script performs the same way as `full-backup.sh`, except that the developer performs an incremental backup instead of a full backup.

The backup timestamp that the developer sets is for incremental backups instead of the full backup. All the environment/shell variables available for `full-backup.sh` are also available in this script.

NOTE: To set timestamp for an incremental backup, use the same **setDataBackupTime** guidelines as the full backup.

```
function startBackup {


  # prepareOutput and initialise must be called first.
  # Prepares the output file and sets some variables.
  prepareOutput
  initialise

  ##### Do Incremental Backup #####
  # eg, mysqldump --databases ${OBJECTS[0]} > ${DATA_VIEW_MOUNT_DIRS[0]}
  for i in {1..10}
  do
    setProgressPercent $i'0'
    echo 'log'$i >> $PULSE_LOGS_FILE
    sleep 5
  done
  ####################

  # After the backup is complete, set the progress percentage to 100.
  setProgressPercent '100'

  # Data backup timestamp for this incremental backup in epoch seconds.
  # args -> Backup time in epoch seconds
  setDataBackupTime '1602674625'

  # Set stats.
  # args -> objects successful, objects failed, bytes transferred
  setStats '5' '0' '50000'

  # Set status.
  # If successful -> setStatusSuccess
  # If partial success, that is only some objects were successful ->
  #   setStatusPartialSuccess
  # If failed -> setStatusFailed,
  setStatusSuccess
}
```

Log Backup Script

This script helps perform log backup for your database. This script is also very similar to the full and incremental backup scripts. There are some minor differences, which we will cover here.

1. In this script, the following shell variables are available for the developer. These shell variables contain information about the Cohesity view in which this backup will be stored. This Cohesity view will be different from the full and incremental backup view.

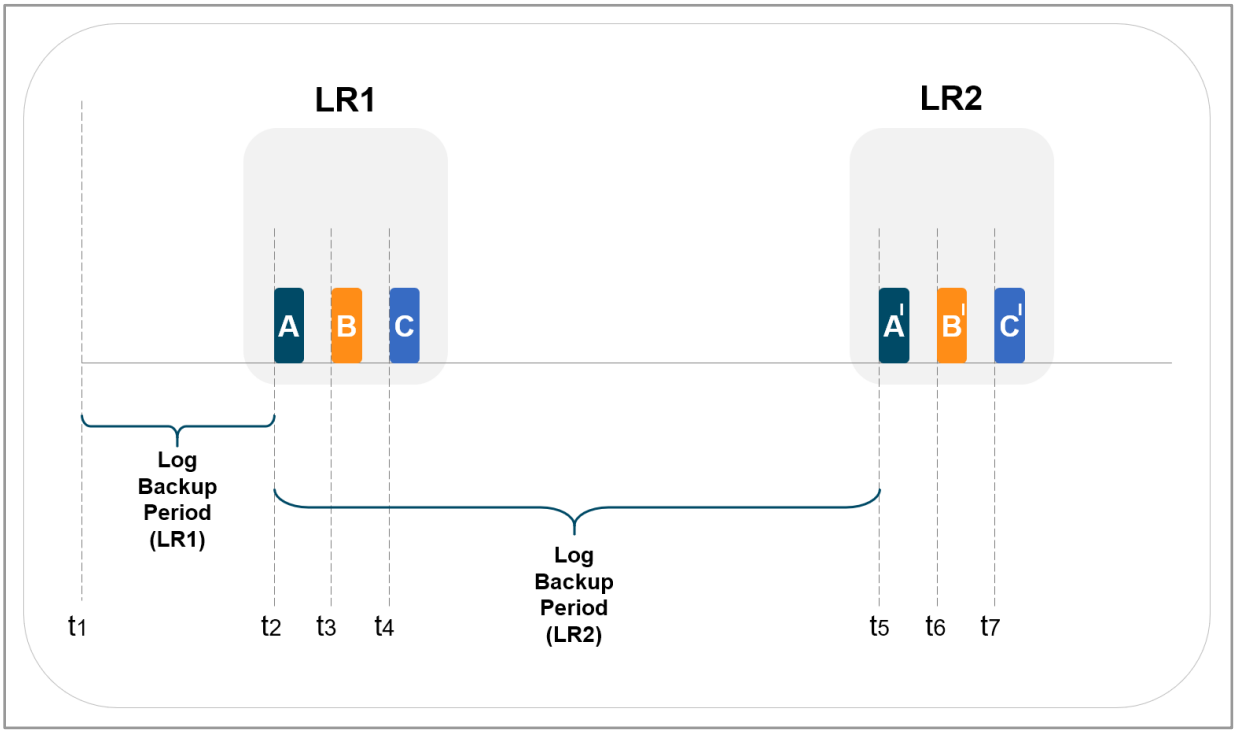
Table 5: Environment Variables for Log Backup Script

SHELL VARIABLES	DESCRIPTION
LOG_VIEW_NAME	The log view name (used by non-mount option).
LOG_VIEW_MOUNT_DIRS	The array of directory paths where log view is mounted if you've selected the mount option.
VIEW_VIPS	The VIPs of the cluster views (used by non-mount option).

2. The developer performs a log backup in this script, and the script needs to return the time range covered by the log transactions captured by this log backup.

   When using **setLogBackupPeriod**, use the following guidelines to set the log backup start time and end time. For the below guideline, let's assume that this log backup backs up three objects A, B, and C.

   - If this log backup is the first one (LR1), the start time should be equal to t1, which is when the backup first begins, and the end time should be equal to the log backup time for the first object in the list (t2 in this case).
   - If this log backup is one of the consequent ones (LR2 for example), after the first one, the start time should be equal to the end time of the previous log backup (t2 in this case), and the end time for LR2 will be the start of log backup of the first object in this run, which is t5 in this case.



```
function startBackup {

  # prepareOutput and initialise must be called first.
  # Prepares the output file and sets some variables.
  prepareOutput
  initialise

  ##### Do Log Backup #####
  # eg, mysqldump --databases ${OBJECTS[0]} > ${LOG_VIEW_MOUNT_DIRS[0]}
  for i in {1..10}
  do
    setProgressPercent $i'0'
    echo 'log'$i >> $PULSE_LOGS_FILE
    sleep 5
  done
  ######################

  # After backup is complete, set the progress percentage to 100.
  setProgressPercent '100'

  # The time period covered by this log backup in epoch seconds.
  # args -> start time, end time.
  setLogBackupPeriod '1602674625' '1602675625'

  # Set stats.
  # args -> objects succesfull, objects failed, bytes transferred
  setStats '5' '0' '50000'

  # Set status.
  # If successful -> setStatusSuccess
  # If partial success, that is only some objects were successful ->
  #   setStatusPartialSuccess
  # If failed -> setStatusFailed,
  setStatusSuccess
}
```

Cancel Backup Script

You can also cancel a running backup using the Cancel button on UI, triggering the `cancel-backup.sh` script. The developer can use this script to cancel the backup and perform any cleanup in the database or metadata necessary to stop the backup completely.

If this script does not successfully finish its execution in 60 seconds, the Cohesity agent will terminate it by killing the process group id. So the backup is expected to stop running after 60 seconds. This time can be modified as per your requirement using the uda_blocking_script_timeout_sec Cohesity agent gflag.

Let's go over the code blocks to understand where changes are needed to support your database.

1. Similar to other scripts, this script will have access to a bunch of environment variables, which the developer can use to write the Cancel Backup functionality.

Table 6: Environment Variables for Cancel Backup Script

ENVIRONMENT VARIABLES	DESCRIPTION
JQ	Path to the jq library deployed by the Cohesity agent.
USERNAME	The username specified in the UI.
PASSWORD	The password specified in the UI.
JOBID	The job ID used to identify this job. Any async processes linked to this job should be identifiable by this job ID. When a job is deleted/canceled, the same job ID will be passed, too clean up such processes.
BACKUP_ARGS	The backup args specified in the UI.
PGID	The process group id of the backup script, which needs to be canceled.
JOB_TYPE	String specifying the backup type. (FULL_BACKUP/INCREMENTAL_BACKUP/LOG_BACKUP)
S3_ENDPOINT	S3 endpoint, eg https://10.2.145.xx:3000, if using S3 view type (**useS3View** in `discover-source.sh`)
S3_ACCESS_KEY	S3 access key.
S3_SECRET_KEY	S3 secret key.

2. The UDA framework will also initialize some shell variables. Below is the list.

Table 7: Shell Variables for Cancel Backup Script

SHELL VARIABLES	DESCRIPTION
DATA_VIEW_NAME	The data view name (Used by non-mount option)
DATA_VIEW_MOUNT_DIRS	The array of path of directories where data view is mounted, if the mount option is selected.
VIEW_VIPS	The VIPs of the cluster views. (Used by non-mount option)
OBJECTS	The array of object names to back up.

3. Following Shell variables will be set if automatic log backup is enabled.

Table 8: Shell Variables for Cancel Backup Script with Auto Log enabled

SHELL VARIABLES	DESCRIPTION
LOG_VIEW_NAME	The log view name (Used by non-mount option)
LOG_VIEW_MOUNT_DIRS	The array of path of directories where log view is mounted, if the mount option is selected.

4. To cancel the running backups, you need to modify this code and add your code snippet in the **cancelBackup** function.

```
function cancelBackup {

  # initialise must be called first.
  # Sets some variables.
  initialise

  ##### Cancel backup #####
  # eg, hypothetical_dump --stop $JOB_TYPE --databases ${OBJECTS[0]}
  ##########################
}
```

Cleanup Backup Script

You can also delete a backup job, and when you do a delete operation from the UI, `cleanup-backup.sh` script is invoked. You can perform any cleanup operation needed on your source DB to delete the backup successfully.

This script should execute in 60 seconds, after which the Cohesity agent kills the process. You can extend the execution wait time by updating the uda_blocking_script_timeout_sec Cohesity gflag.

Another important thing to note is that since the cleanup happens in the background, you will have to check the status of the command run manually.

This script is pretty straightforward to understand. Let's quickly go over it and see where you need to make changes to support deleting the backups.

1. The following environment variables are available to this script.

Table 9: Environment Variables for Cleanup Backup Script

ENVIRONMENT VARIABLES	DESCRIPTION
JQ	Path to the jq library deployed by the Cohesity agent.
USERNAME	The username specified in the UI.

ENVIRONMENT VARIABLES	DESCRIPTION
PASSWORD	The password specified in the UI.
JOBID	The id of the job being deleted.

2. To perform cleanup for the backups created, add your code in the **cleanupBackup** function.

```
function cleanupBackup {

  # initialise must be called first.
  # Sets some variables.

  ##### Cleanup backup #####
  # eg, cleanup_deleted_job --job_id $JOBID --username ${USERNAME} --
password ${PASSWORD}
  ##########################
}
```

Recover Your Database Using UDA

Once the database is protected using the Backups scripts, you can recover these databases either to the same or a new location. You will need to modify the recovery scripts to do that.

Recover Your Data

In order to support the recovery workflow using the UDA framework, you need to modify recovery scripts to restore the backup objects or cancel a restore task.

The available backup scripts are:

1. restore.sh
2. cancel-restore.sh

Let's understand each of these scripts so that you can modify these scripts and perform a recovery workflow on your DBs.

Restore Script

To recover the database you protected using the backup scripts, you need to add a recovery code snippet in the `restore.sh`.

Let's understand this script and see where changes are needed to support recovery workflow.

1. The following environment variables will be made available by the UDA framework to this script.

Table 10: Environment Variables for Restore Script

ENVIRONMENT VARIABLES	DESCRIPTION
JQ	Path to the JQ library deployed by the Cohesity agent.
USERNAME	The username specified in the UI.
PASSWORD	The password specified in the UI.
JOBID	The job ID used to identify this job. Any async processes linked to this job should be identifiable by this job ID. When a job is deleted or canceled, the same job ID will be passed to clean up such processes.
RESTORE_ARGS	The restore args specified in the UI
PARALLEL_STREAMS	Number of parallel streams that can be used to read per mount directory/vip.
JOB_TYPE	String specifying the backup type. (RESTORE/PITR_RESTORE)

ENVIRONMENT VARIABLES	DESCRIPTION
S3_ENDPOINT	S3 endpoint, eg https://10.2.145.xx:3000, if using S3 view type (**useS3View** in `discover-source.sh`)
S3_ACCESS_KEY	S3 access key.
S3_SECRET_KEY	S3 secret key.

2. The UDA framework will also initialize some shell variables.

Table 11: Shell Variables for Recover Script

SHELL VARIABLES	DESCRIPTION
DATA_VIEW_NAME	The data view name (Used by non-mount option)
DATA_VIEW_MOUNT_DIRS	Array of path of directories on which the data view is mounted if the mount option is selected.
VIEW_VIPS	The VIPs of the cluster views. (Used by non-mount option)
OBJECTS	The array of object names restore.
RENAMED_OBJECTS	The array of renamed objects.
OVERWRITE	Whether to overwrite or keep the object if the object being recovered already exists in the destination. Array with elements "true" or "false".
DATA_BACKUP_EPOCH_SECS	The array of epoch seconds for each object. This is the time up to which the snapshot being recovered has backed up the data.

3. The following shell variables will be initialized if point in time recovery is used (PITR):

Table 12: Shell Variables for Restore Script with PITR

SHELL VARIABLES	DESCRIPTION
PITR_EPOCH_SECS	The array of epoch seconds for each object, the point in time to restore, if PITR is selected.

```
# Example,
# OBJECTS=("db1.col1" "db2.col2")
# RENAMED_OBJECTS=("db1.col1_restored" "db2.col2")
# PITR_EPOCH_SECS=("1602674625", "1602675625")
# DATA_BACKUP_EPOCH_SECS=("1602674840", "1602674840")
#
# OJBECTS[1] should be restored as RENAMED_OBJECTS[1] to
#   point in time PITR_EPOCH_SECS[1]
```

4. Similar to backup scripts, you can show logs in the UI by writing them to the `PULSE_LOGS_FILE`.

```
echo 'log'$i >> $PULSE_LOGS_FILE
```

5. In the **startRestore** function, the **prepareOutput** and **initialize** functions are called to initialize the input, output, and environment/shell variables for this script. The developers can start using it to perform the restore operation.

```
function startRestore {

  # prepareOutput and initialise must be called first.
  # Prepares the output file and sets some variables.
  prepareOutput
  initialise
```

6. The next step is to perform the actual restore for the objects. You can increase the progress percent based on the restore progress.

```
##### Do Restore #####

  # Example,
  # for i in "${!OBJECTS[@]}"; do
  #   # Do restore from data view.
  #   mysql -u $USERNAME <
${DATA_VIEW_MOUNT_DIRS[$i]}/"${OBJECTS[$i]}".dump
  #   if [ -n "$LOG_VIEW_MOUNT_DIRS" ]; then
  #     # Do restore from log view as well.
  #     if [ -n "$PITR_EPOCH_SECS" ]; then
  #       # Do PITR.
  #       hypothetical_log_restore --pitr ${PITR_EPOCH_SECS[$i]} < \
  #         ${LOG_VIEW_MOUNT_DIRS[$i]}/"${OBJECTS[$i]}".dump
  #     else
  #       # Not PITR, restore the complete log.
  #
  #       if [ "${OVERWRITE[$i]}" = "true" ]; then
  #         echo "Overwriting the object " "${OBJECTS[$i]}"
  #       fi
  #       hypothetical_log_restore  < \
  #       ${LOG_VIEW_MOUNT_DIRS[$i]}/"${OBJECTS[$i]}".dump
  #     fi
  #   fi
  # done

  for i in {1..10}
  do
    setProgressPercent $i'0'
    echo 'log'$i >> $PULSE_LOGS_FILE
    sleep 5
  done
  ######################
```

7. After successfully completing the restore, set the progress to 100%, set the stats and success/failure of the restore task. The UDA framework displays the stats in the UI.

   a. **setStats**: This function sets the number of successfully protected and failed objects, and the bytes transferred during the backup.

   b. **setStatusSuccess**: This function sets the status for the backup operation. Other operations are **setStatusPartialSuccess** and **setStatusFailed**.

```
setProgressPercent '100'

# Set stats.
# args -> objects succesfull, objects failed, bytes transferred
setStats '5' '0' '50000'

# Set status.
# If successful -> setStatusSuccess
# If partial success, that is only some objects were successful ->
#   setStatusPartialSuccess
# If failed -> setStatusFailed,
setStatusSuccess
```

Cancel Database Restore

You can also cancel a running restore task using the cancel-restore.sh script. You can use this script to cancel the restore and perform any cleanup necessary to stop the restore completely.

If this script does not complete its execution in 60 seconds, the Cohesity agent will terminate it by killing the process group id. So, the restore is expected to finish running after 60 seconds. This time can be modified as per your requirement using the uda_blocking_script_timeout_sec Cohesity gflag.

Let's go over the code blocks to understand where changes are needed to support your database.

1. Like every other script, the following environment variables will be available to your script.

Table 13: Environment Variables for Cancel Restore Script

ENVIRONMENT VARIABLES	DESCRIPTION
JQ	Path to the JQ library deployed by the Cohesity agent.
USERNAME	The username specified in the UI.
PASSWORD	The password specified in the UI.
JOBID	The job id used to identify this job. Any async processes linked to this job should be identifiable by this job id. When a job is deleted/ canceled, the same job ID will be passed to clean up such processes.
RESTORE_ARGS	The restore args specified in the UI
PGID	The process group id of the restore script, which needs to be canceled
JOB_TYPE	String specifying the backup type. (RESTORE/PITR_RESTORE)

ENVIRONMENT VARIABLES	DESCRIPTION
S3_ENDPOINT	S3 endpoint, eg https://10.2.145.xx:3000, if using S3 view type (useS3View in discover-source.sh)
S3_ACCESS_KEY	S3 access key.
S3_SECRET_KEY	S3 secret key.

2. The UDA framework also initializes some shell variables.

Table 14: Shell Variables for Cancel Restore Script

SHELL VARIABLES	DESCRIPTION
DATA_VIEW_NAME	The data view name (Used by non mount option)
DATA_VIEW_MOUNT_DIRS	The array of path of directories where data view is mounted, if the mount option is selected.
VIEW_VIPS	The VIPs of the cluster views. (Used by non mount option)
OBJECTS	The array of object names to restore.
RENAMED_OBJECTS	The array of renamed objects.

3. Following shell variables will be available if log view is also used for the restore.

Table 15: Shell Variables for Cancel Restore Script if Log view is used

SHELL VARIABLES	DESCRIPTION
LOG_VIEW_NAME	The log view name (Used by non-mount option)
LOG_VIEW_MOUNT_DIRS	The array of path of directories where log view is mounted if the mount option is selected.

4. The following shell variables will be initialized if point in time recovery is used (PITR).

Table 16: Shell Variables for Cancel Restore Script with PITR

SHELL VARIABLES	DESCRIPTION
PITR_EPOCH_SECS	The array of epoch seconds for each object, the point in time to restore, if PITR is selected.

5. To support the cancel restore operation, you will need to add your code snippet in the **cancelRestore** function.

```
function cancelRestore {

  # initialise must be called first.
  # Sets some variables.
  initialise

  ##### Cancel restore #####
  # eg, hypothetical_dump --stop --databases ${OBJECTS[0]}
  ##########################
}
```

Start Using the Connector

Once you have made changes in all the scripts per your environment, the next step is to start using it to protect your data source on Cohesity. There are two parts to it; the first part must be performed on all the nodes on which the data source to be protected is installed while the second part is registering this data source on Cohesity. Let's dive into these steps.

Copy Files and Install Cohesity Agent

1. SSH to the physical node(s) where the data source is installed.
2. Create a directory /opt/cohesity/<<***data_source_name***>>/uda-scripts/

```
# mkdir -p /opt/cohesity/uda/mydb/uda-scripts
```

3. Copy all the files that you downloaded from the Github repository. These files should include all the changes you made while editing the files. If you are using any additional files or scripts, copy them to this directory as well.

```
# ls -lrt /opt/cohesity/uda/mydb/uda-scripts
total 180
-rwxr-xr-x 1 root root  7446 Jul 26 16:46 cancel-restore.sh
-rwxr-xr-x 1 root root  7426 Jul 26 16:46 cancel-backup.sh
-rwxr-xr-x 1 root root  1696 Jul 26 16:46 util_methods.sh
-rwxr-xr-x 1 root root 38532 Jul 26 16:46 restore.sh
-rwxr-xr-x 1 root root  2736 Jul 26 16:46 register-utils.sh
-rwxr-xr-x 1 root root 26703 Jul 26 16:46 log-backup.sh
-rwxr-xr-x 1 root root  6400 Jul 26 16:46 job-utils.sh
-rwxr-xr-x 1 root root 26589 Jul 26 16:46 incr-backup.sh
-rwxr-xr-x 1 root root 26589 Jul 26 16:46 full-backup.sh
-rwxr-xr-x 1 root root  4383 Jul 26 16:46 discover-source.sh
-rwxr-xr-x 1 root root 11518 Jul 26 16:46 db2-utils.sh
-rwxr-xr-x 1 root root  3641 Jul 26 16:46 verify-source.sh
```

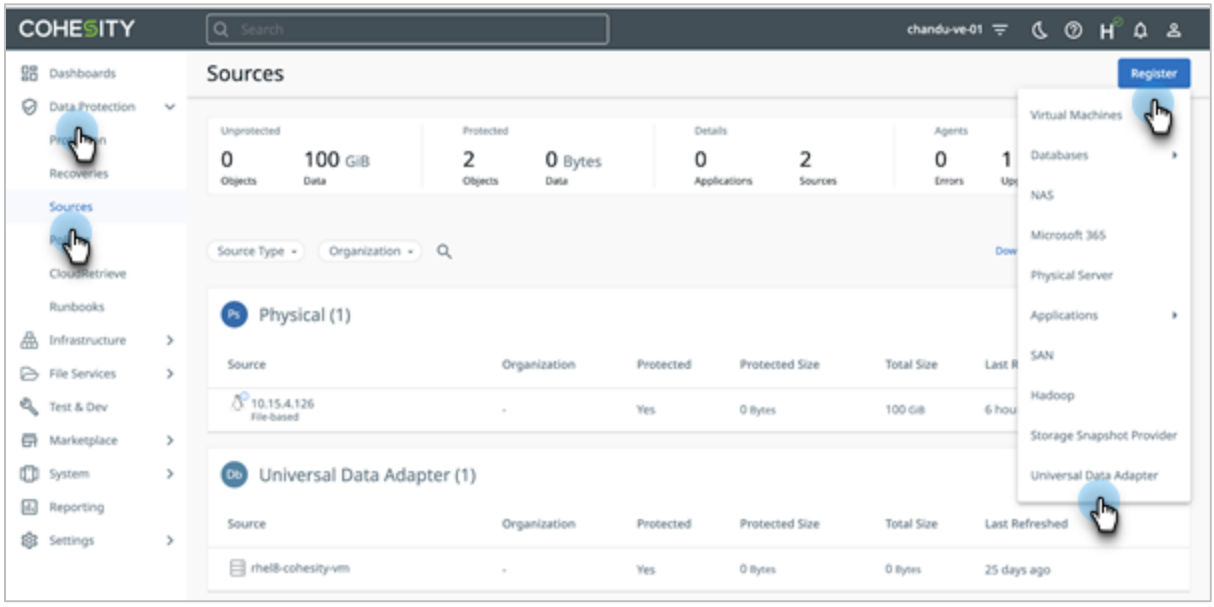
4. Download and Install Cohesity agent on all nodes.

Register Data Source on Cohesity

Once you have created a directory on all nodes and copied the scripts there, we can register the data source on Cohesity.

1. Log in to Cohesity as an administrator.

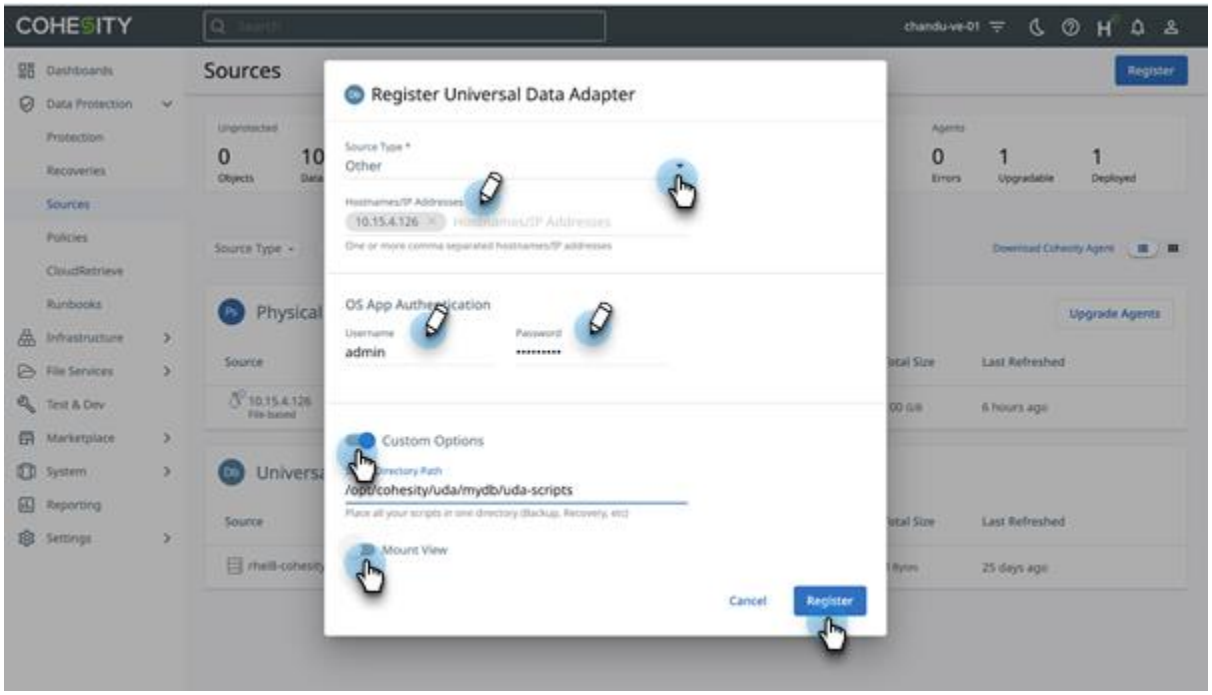
2. Navigate to **Data Protection** > **Sources** > **Register** > **Universal Data Adapter**.



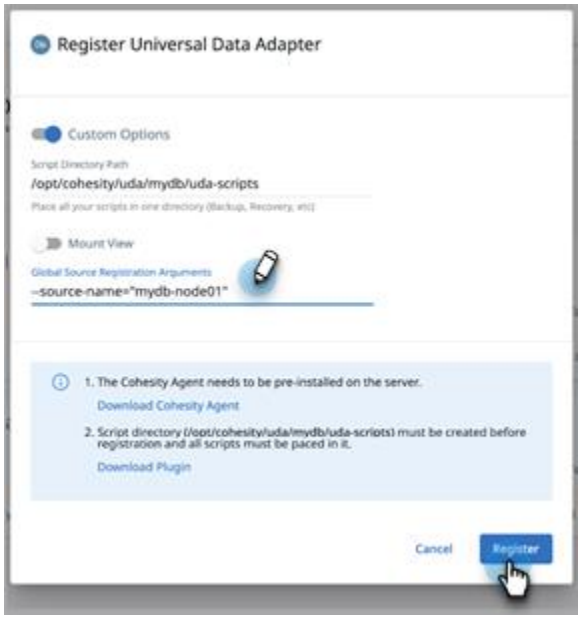
3. On the Register page, under **Source Type**, choose **Other**.
4. In the **Hostname/IP Addresses** field, enter the hostname or IP address of the node(s) you have identified to run the connector.
5. In the **OS APP Authentication** section, enter the **Username** and **Password** of the data source admin user.

> **NOTE**: This is an optional step. It may not be necessary depending on the database being protected.

6. Enable **Custom Options**, and in the Script Directory Path, enter the directory path on the node where you copied the scripts.



7. Enable **Mount View** if you want to mount Cohesity views and perform backups on these views.
8. Under **Global Source Registration Arguments**, you can pass any additional arguments that your scripts expect.
9. Click **Register**.



Debug and Troubleshoot Errors

After deploying the connector and registering the data source on Cohesity, you can start protecting your data source and perform recovery. When performing any of these steps, if you face errors, check the logs under /var/log/cohesity/uda/. Just list the log contents, sort it in ascending order, and check the last file logs to find more information about the error to debug the issue.

```
# ls -lrt /var/log/cohesity/uda/
-rw------- 1 cohesity db2grp1     42 Jul 19 09:20 discover-source.2F4BEB6A-43C1-4D09-
9AD5-850D19D39D96.INPUT.json
-rw-r--r-- 1 cohesity db2grp1    289 Jul 19 09:20 discover-source.2F4BEB6A-43C1-4D09-
9AD5-850D19D39D96.OUTPUT.json
-rw-r--r-- 1 cohesity db2grp1    138 Jul 19 09:20 discover-source.2F4BEB6A-43C1-4D09-
9AD5-850D19D39D96.STDOUT
-rw-r--r-- 1 cohesity db2grp1  27401 Jul 19 09:20 discover-source.2F4BEB6A-43C1-4D09-
9AD5-850D19D39D96.STDERR
-rw------- 1 cohesity db2grp1     42 Jul 19 09:21 verify-source.9BBDC2FB-AE6D-4D92-
A8E9-CF238F92AC6C.INPUT.json
-rw-r--r-- 1 cohesity db2grp1     87 Jul 19 09:21 verify-source.9BBDC2FB-AE6D-4D92-
A8E9-CF238F92AC6C.STDOUT
-rw-r--r-- 1 cohesity db2grp1  24816 Jul 19 09:21 verify-source.9BBDC2FB-AE6D-4D92-
A8E9-CF238F92AC6C.STDERR
-rw------- 1 cohesity db2grp1     57 Jul 19 22:37 full-backup.450127667805268-
1617054208155-40333.OBJECTS.json
-rw------- 1 cohesity db2grp1    126 Jul 19 22:37 full-backup.450127667805268-
1617054208155-40333.INPUT.json
-rw------- 1 cohesity db2grp1     23 Jul 19 22:37 full-backup.450127667805268-
1617054208155-40333.PROGRESS.json
-rw-r--r-- 1 cohesity db2grp1    173 Jul 19 22:37 full-backup.450127667805268-
1617054208155-40333.OUTPUT.json
-rw-r--r-- 1 cohesity db2grp1  10257 Jul 19 22:37 full-backup.450127667805268-
1617054208155-40333.STDOUT
-rw-r--r-- 1 cohesity db2grp1 115422 Jul 19 22:37 full-backup.450127667805268-
1617054208155-40333.STDERR
-rw-r--r-- 1 cohesity db2grp1    478 Jul 19 22:37 full-backup.450127667805268-
1617054208155-40333.PULSE.log
-rw------- 1 cohesity db2grp1    196 Jul 19 22:41 restore.450127667805268-
1617054208155-40334.OBJECTS.json
-rw------- 1 cohesity db2grp1    123 Jul 19 22:41 restore.450127667805268-
1617054208155-40334.INPUT.json
-rw------- 1 cohesity db2grp1     23 Jul 19 22:41 restore.450127667805268-
1617054208155-40334.PROGRESS.json
-rw-r--r-- 1 cohesity db2grp1   4540 Jul 19 22:41 restore.450127667805268-
1617054208155-40334.STDOUT
-rw-r--r-- 1 cohesity db2grp1    475 Jul 19 22:41 restore.450127667805268-
1617054208155-40334.PULSE.log
-rw-r--r-- 1 cohesity db2grp1    156 Jul 19 22:41 restore.450127667805268-
1617054208155-40334.OUTPUT.json
-rw-r--r-- 1 cohesity db2grp1  62189 Jul 19 22:41 restore.450127667805268-
1617054208155-40334.STDERR
```

UDA Developer Deep Dive

This section is for developers who want to dive deep into the template scripts and understand how the different functions in utils scripts work. Developers can modify these scripts if they want to update or write their own util functions or call another script to perform certain operations.

The utils script creates JSON files that store various information that the UI reads to show various details. Before diving into the utils script, let's take a look at the files that are created.

1. **INPUT_JSON_FILE** and **OBJECTS_JSON_FILE**: Scripts read this file, which the UDA framework creates, to initialize some shell variables.
2. **OUTPUT_JSON_FILE**: The helper functions write information to this file, which the UI reads to display the information. The object written is different based on the script; we will look into it in the following sections.
3. **PROGRES_JSON_FILE**: The progress percentage for any operation like backup and restore is written to this file. UI reads this file and displays the value.
4. **PULSE_LOGS_FILE**: The logs to display on the UI are appended to this file.

Register Utils Script

This script has helper functions that help initialize environment and shell variables by reading the INPUT_JSON_FILE. This script also creates and initializes the OUTPUT_JSON_FILE.

Sample INPUT_JSON_FILE

```
# Variables set using INPUT_JSON_FILE:
# -----------------------------------
#
# Example file,
# {
#   "cluster_nodes": [
#     "10.2.xxx.88",
#     "10.2.xxx.89"
#   ]
# }
#
# CLUSTER_NODES: List of node IPs forming the source cluster.
```

Sample OUTPUT_JSON_FILE

```
# Output:
# ------
#
# Output must be as follows (should be set using
#   helper functions).
#
# {
#   "source_name": "UdaSourceName",
#   "source_uuid": "UdaSourceUUID",
#   "fresh_full_backup_view": false,
#   "live_data_view": false,
#   "control_nodes": [
#     "10.2.xxx.88",
#     "10.2.xxx.89"
#   ],
#   "capabilities": [
#     "FULL_BACKUP",
#     "LOG_BACKUP"
#   ]
# }
#
# source_name: A name to identify the source in the UI.
# source_uuid: A UUID to uniquely identify this source. This must be same for
#   every call to this script. All nodes that are a part of the same source
#   ( in case of a clustered database ) should return the same UUID.
# fresh_full_backup_view: Specify if a new empty view is required for
#   full backups. This should always be true. There could be exceptions
#   like SAP HANA.
#
# live_data_view: Whether to use live data view for backups. This should
always
#   be false with exceptions like Cohesity SAP Hana and SAP Oracle
#   to support inquiries using SnapFS grpc.
#
# capabilities: The backup capabilities of this source.
#   FULL_BACKUP, INCR_BACKUP, LOG_BACKUP, AUTO_LOG_BACKUP.
#   LOG_BACKUP and AUTO_LOG_BACKUP cannot be set together.
# control_nodes: Nodes on which the backup/restore scripts can be triggered.
```

Job Utils Script

This script has helper functions that help initialize environment and shell variables by reading the INPUT_JSON_FILE and OBJET_JSON_FILE. This script also creates and initializes the OUTPUT_JSON_FILE, PROGRESS_JSON_FILE, and PULSE_LOGS_FILE.

Sample INPUT_JSON_FILE for Backup script:

```
# Variables set using INPUT_JSON_FILE:
# ------------------------------------
#
# Example file,
# {
#   "data_view_name": "data_view",
#   "data_view_mount_dirs": [
#     "dir1",
#     "dir2"
#   ],
#   "view_vips": [
#     "vip1",
#     "vip2"
#   ]
# }
#
# DATA_VIEW_NAME: The data view name (Used by non mount option)
# DATA_VIEW_MOUNT_DIRS: Array of path of directories where data view
#   is mounted, if mount option is selected.
# VIEW_VIPS: The vips of the cluster views. (Used by non mount option)
#
# Variables that will be set if automatic log backup is enabled:
#
# LOG_VIEW_NAME: The log view name (Used by non mount option)
# LOG_VIEW_MOUNT_DIRS: Array of path of directories where log view
```

Sample OUTPUT_JSON_FILE for backup script:

```
# Output:
# ------
#
# PULSE_LOGS_FILE
#
# OUTPUT_JSON_FILE should be as follows (should be set using
#   helper functions).
# {
#   "status": "SUCCESS",
#   "stats" : {
#     "objects_successful": 5,
#     "objects_failed": 1,
#     "bytes_transferred": 500000
#    }
# }
#
```

Sample OBJECTS_JSON_FILE for backup script:

```
# Variables set using OBJECTS_JSON_FILE:
# -------------------------------------
#
# Example file,
# {
#   "objects": [
#     {
#       "name": "obj1",
#     },
#     {
#       "name": "obj2",
#     }
#   ]
# }
#
# OBJECTS: Array of object names to be backed up.
```

Sample PROGRESS_JSON_FILE:

```
# PROGRESS_JSON_FILE should be as follows (should be set using
#   helper functions).
# {
#   "percentage": "75",
# }
```

Your Feedback

Was this document helpful? Send us your feedback!

About the Authors

Chandrashekar Dashudu is a Sr. Technical Marketing Engineer, focusing on API integrations and Apps.

Other essential contributors included:

- Sourish Kuthial, Member of Technical Staff
- Karan Sood, Sr. Member of Technical Staff
- Sapan Maniyar, Sr. Manager
- Apollo Tanting, Sr. Product Manager
- Subash Babu, Staff Technology Editor, Technical Marketing Engineering

Document Version History

VERSION	DATE	DOCUMENT HISTORY
1.0	Aug 2021	First release
1.1	July 2024	Republished

ABOUT COHESITY

Cohesity radically simplifies data management. We make it easy to protect, manage, and derive value from data -- across the data center, edge, and cloud. We offer a full suite of services consolidated on one multicloud data platform: backup and recovery, disaster recovery, file and object services, dev/test, and data compliance, security, and analytics -- reducing complexity and eliminating mass data fragmentation. Cohesity can be delivered as a service, self-managed, or provided by a Cohesity-powered partner.

Visit our website and blog, follow us on Twitter and LinkedIn and like us on Facebook.