Hyperscale Compliance Home

Hyperscale Compliance

Exported on 02/10/2025

Table of Contents

Welcome to the Hyperscale Compliance documentation!

When databases contain billions of rows of data, it can take weeks to protect sensitive data and PII using manual processes or bulk masking to anonymize the data. Hyperscale Compliance from Delphix provides incredibly fast masking speeds for large datasets enabling continuous compliant data delivery for CI/CD and DevOps initiatives.

Hyperscale Compliance does this by distributing the masking workload for a single job across multiple virtual Continuous Compliance Engines, reducing the time to mask large databases through increased scalability and efficiency.

This information explains how to deploy Hyperscale Compliance, use its features, or tune its configurations for optimal performance. The content has been organized into several categories, available from the lefthand navigation.

List of Hyperscale Compliance documentation versions in PDF format.

- 2025.1.0_HyperscaleCompliance.pdf
- 27.0.0_HyperscaleCompliance.pdf
- 26.0.0_HyperscaleCompliance.pdf
- 25.0.0_HyperscaleCompliance.pdf
- 24.0.0_HyperscaleCompliance.pdf
- 23.0.0_HyperscaleCompliance.pdf
- 22.0.0_HyperscaleCompliance.pdf
- 21.0.0_HyperscaleCompliance.pdf
- 20.0.0_HyperscaleCompliance.pdf
- 19.0.0_HyperscaleCompliance.pdf
- 18.0.0_HyperscaleCompliance.pdf
- 17.0.0_HyperscaleCompliance.pdf
- 16.0.0_HyperscaleCompliance.pdf
- 15.0.0_HyperscaleCompliance.pdf
- 14.0.0_HyperscaleCompliance.pdf
- 13.0.0_HyperscaleCompliance.pdf
- 12.0.0_HyperscaleCompliance.pdf
- 11.0.0_HyperscaleCompliance.pdf
- 10.0.0_HyperscaleCompliance.pdf
- 9.0.0_HyperscaleCompliance.pdf
- 8.0.0_HyperscaleCompliance.pdf
- 7.0.0_HyperscaleCompliance.pdf
- 6.0.0_HyperscaleCompliance.pdf
- 5.0.0_HyperscaleCompliance.pdf
- 4.1.0_HyperscaleCompliance.pdf
- 4.0.0_HyperscaleCompliance.pdf
- 3.0.0_HyperscaleCompliance.pdf
- 1.0.0_HyperscaleCompliance.pdf

Quick references

- Overview
- Architecture
- Data source support
- New features
- Fixed issues

Release notes

This section is used to learn what the newest version of Hyperscale Compliance has to offer. In addition, the fixed and known issues per version are detailed.

New features

Version naming convention change
Delphix has introduced a new version naming convention. From January 2025, Hyperscale Compliance will follow a YYYY.R.M format:
YYYY: Year of distribution
R: Release number
M: Minor release number

2025.1.0 release

This release supports the following features:

- **Snowflake Hyperscale connector**
  The new Snowflake connector provides enhanced export capabilities for splitting Snowflake source data into multiple CSV files. The Snowflake connector uses AWS S3 Cloud Object Storage for masking and securely imports the masked data back into the designated Snowflake target table.
  This Snowflake connector release also introduces a separate artifact known as the Snowflake Profiler. The Snowflake Profiler is integrated with Swagger UI, and this new feature profiles Snowflake table data, generates an inventory of sensitive columns and then submits the payload to the dataset API. Refer to the DataSets API for further information.
- **Azure Blob as a staging area**
  Azure blob containers can now be used as a staging area for Hyperscale Compliance. This feature is only available for Kubernetes deployment mode. For more information, refer to Configure Azure Blob containers as staging area and Commonly used properties.
- **MongoDB v7 and v8 certification**
  The Linux platform now supports MongoDB 7.0.x and MongoDB 8.0.x connectors.
- **File connector enhancements**
  The file connector can now generate a support bundle using APIs.

27.0.0 release

There are no new features in the 27.0.0 release.

26.0.0 release

This release supports the following feature:

File connector enhancements - new source and target parameters
This feature allows you to configure parameters such as writer type, spark configuration settings, and max worker threads for each job without restarting your containers. Previously, variables were set using environment variables, and if a change was made on a parameter, it required a container restart. From this release, you can now apply different settings in parallel because variables are job and not container-specific.
See Kubernetes and OpenShift for instructions on configuring the source and target connector types.

The existing method of setting environment variables is still supported. However, from this 26.0.0 release, if you provide values in both job configs and environment variables, job configs will take precedence.

25.0.0 release

This release supports the following features:

- **File connector enhancements**
  - **Staging push**
    In version 25.0.0, we've introduced a staging push feature that allows users to bypass the unload and load processing steps. This feature allows users to directly provide source files at a mount point mapped to the staging area. Instead of using writers to parse the source files during unload, a link to the files is created within the staging area and passed to the Continuous Compliance engine. During the load operation, the masked files will be available as links in the target location. Note, the staging push feature is restricted to delimited files.
  - **Fail running jobs on container restart**
    This feature ensures that any unload or load jobs running at the time of a container restart are marked as failed.
- **Oracle connector - "where clause" filter for source dataset**
  You can now filter the source data to be processed by a Hyperscale job by applying a "where clause" filter condition in a job's data set. The "where clause" condition is added in unload SQL queries when fetching the data from a source database. Refer to DataSets API for a sample request.

> When applying the "where clause," it is important to note that only filtered data will appear in the target database upon job completion.

24.0.0 release

This release supports the following feature/features:

- **Automated handling of multiple date formats for MSSQL connector**
  This release introduces a feature that automatically handles multiple date formats within a single job. This removes the limitation from jobs that require all date/timestamp columns in the dataset to have the same date formats. Also, starting from this release, you no longer need to configure any unload/load date format environment variables to mask date/timestamp columns. If you have the below properties set in the `docker-compose.yaml` / `values.yaml` files, these can be safely removed from unload and load services:
  - `SPARK_TIMESTAMP_FORMAT`
  - `SPARK_DATE_FORMAT`
  - `SPARK_SMALL_DATE_TIMESTAMP_FORMAT`
- **AWS S3 as a staging area**
  This release introduces the ability to use AWS S3 buckets as staging area for Hyperscale Compliance. This feature is only available for Kubernetes deployment mode. For more information, refer to Configure AWS S3 bucket as staging area and Commonly used properties.
- **Automatic calculation of Unload Split in Oracle Connector**
  The Automatic calculation of unload split for Oracle DataSource eliminates the need to manually provide the number of splits. This not only saves time but also makes the product more user-friendly and scalable. See Oracle - Automatic calculation of Unload Split for more details.
- **Resource Requests and Limits for Kubernetes deployment**
  Users can now specify Resource requests and limits for Kubernetes deployment. For more information, see Set Resource Requests and Limits
- **/mount-filesystems API removal**
  With this release, /mount-filesystems API has been removed. A single MountFileSystem is now set up using the configuration parameters during application startup, deleting all the previously existing instances of the

MountFileSystem. New read-only endpoints are available to get/validate the configured staging area details. For more information, refer to Commonly used properties and How to upgrade the Hyperscale Compliance Orchestrator.

⚠

- Application startup will fail if the required properties related to NFS Server/AWS S3 bucket are not specified in the corresponding yaml/.env file to configure the mount point. Controller-service logs shall indicate error message like below:

```
ERROR o.s.boot.SpringApplication.reportFailure - Application run failed
com.delphix.hyperscale.common.exception.UserException: Specify all of the
following configurations: 'NFS_STORAGE_HOST', 'NFS_STORAGE_EXPORT_PATH',
'NFS_STORAGE_MOUNT_TYPE'.
```

- In case of upgrade :
  - Existing MountFileSystem information will be deleted on application startup. Please ensure to take backup of the mount information, if needed.
  - Property `APPLICATION_NAME` value should be same as mount-name used for job executions.
  - Upgrade on Docker deployments would need re-mounting of NFS path. Refer to Upgrading the Hyperscale Compliance Orchestrator (Docker Compose) for details.

23.0.0 release

This release supports the following feature/features:

- **File connector**
  This release unifies the Delimited and Parquet connectors into a single, versatile file connector. All existing capabilities of both connectors are preserved. This consolidation eliminates the need for separate deployments for delimited and parquet file formats. The upgrade path for Parquet/Delimited connectors can be found here.
- **Hadoop Support through File connector**
  This release allows you to seamlessly integrate Hadoop File Systems as both data source and destination.
- **Automated handling of multiple date formats for Oracle connector**
  This release introduces a feature that automatically handles multiple date formats within a single job. This removes the limitation from jobs that require all date/timestamp columns in the dataset to have the same date formats. Also starting this release, the user no longer needs to configure any unload/load date format environment variables to mask date/timestamp columns. In case users have the below properties set in the `docker-compose.yaml` / `values.yaml` file, these can be safely removed :
  - Unload service - `JDBC_DATE_TIMESTAMP_FORMAT`
  - Load service - `SQLLDR_DATE_TIMESTAMP_FORMAT`
- **Performance improvement in MS SQL connector**
  - Unload
    This release enhances the performance when using date or datetime data types as filter keys. This improves the performance of unloading data from the source database when using filter keys of types `date`, `datetime`, `datetime2`, `smalldatetime`, or `time`.
  - Post load
    - This release enhances the performance of post load process. While enabling constraints and indexes, the creation of CLUSTERED and NON_CLUSTERED indexes in tables is optimized by ensuring the correct order.

- This release introduces connection pooling for the target database. Hyperscale Compliance now maintains a connection pool for post load process with a default pool size of 15. You can refer to the Configuration Settings page to customize the pool size property.

22.0.0 release

This release supports the following feature/features:

- **Automated handling of MSSQL partitioned indexes**
  This release automates the handling of MSSQL partitioned indexes (clustered and non-clustered), ensuring that partition indexes are automatically dropped before load starts and created again once data is loaded back in target database. Dropping of partitioned indexes (clustered and non-clustered) before load process starts will also have positive imapct on overall Hyperscale job performance.
- **Handling of WARNING status for Continuous Compliance Jobs**
  Continuous Compliance now includes a new Job Execution Status: WARNING. Hyperscale Compliance has been updated to handle this new status. The behavior of Hyperscale Jobs with Continuous Compliance jobs in WARNING status is controlled by a newly introduced flag, i.e. `consider_continuous_compliance_warning_event_as` . This flag has been added to the Hyperscale Job configurations. For configuration details and examples, refer to the Jobs API section on How to setup a Hyperscale Compliance job page.

21.0.0 release

This release supports the following feature/features:

- **Automated handling of Oracle BLOB/CLOB Columns**
  This release automates the handling of Oracle BLOB/CLOB columns, ensuring that null or empty string values are managed seamlessly. With this update, user intervention is no longer required to determine how a CLOB or BLOB value should load when null or empty. The target database will now accurately match the source database's values, whether null or empty. Consequently, the load_empty_lobs_as_null property under target_config in the Job API has been removed due to this automation.
- **Automated configuration of mount filesystem**
  This release introduces new configuration settings that will enable the automatic configuration of the mount filesystem during application startup. With the availability of these new configuration parameters, the /mount-filesystems API is deprecated as of this release and will be removed in future releases. For more information, refer to API Flow to Setup a Hyperscale Compliance Job and Commonly Used Properties.
- **MongoDB connector enhancements**
  - The MongoDB Hyperscale connector now supports Reduced Privilege Operations. This enhancement eliminates the clusterAdmin and clusterMonitor privileges requirement when the drop_collection parameter is set to “No” on the target datasets.
- **Deployment platform certifications**
  - AWS EKS - Parquet and Delimited connectors
  - Openshift - MongoDB connector

20.0.0 release

This release supports the following feature/features:

- **MongoDB connector enhancements**
  - Job cancellation - This release introduces an end-to-end job cancellation feature for the MongoDB connector, allowing you to cancel any Hyperscale job execution. All active unload, masking, and load tasks will be canceled.

  - Support bundle API - The MongoDB connector now supports generating a support bundle through APIs.

> This feature is available for Oracle, MSSQL, and MongoDB connectors. For the Delimited and Parquet connectors, the generated bundle will have information for only controller and masking services.

- **Deployment platform certifications**
  - AWS EKS - Oracle and MSSQL connector

19.0.0 release

This release supports the following feature/features:

- **Support for AWS S3 as source and target locations for Delimited connector**
  This release supports AWS S3 as source and target locations for the Delimited connector. You can now provide the S3 bucket path as your source and target locations in addition to the already supported mounted filesystem(FS) if required.
- **Added APIs to generate support-bundle**
  This release introduces substantial enhancements to the Hyperscale platform, empowering you to generate a support bundle efficiently through an API. This process will now operate asynchronously, delivering a more streamlined solution compared to the previous script-based method. For more information, refer to How to generate a support bundle.

> This feature is available for Oracle and MSSQL connectors. For Mongo, Delimited, and Parquet connectors, the generated bundle will have information for only controller and masking services.

- **Added support to sync masking jobs with structured data**
  This release brings significant improvements to the Hyperscale job sync functionality. You can now import jobs with rulesets containing structured data applied to data columns. As a result, you will receive a connector, structuredDataFormats, and a dataset ready for immediate use.
- **OpenShift deployment for Hyperscale and Enhanced volume management**
  This release introduces the capability to deploy the Hyperscale Compliance Orchestrator on an OpenShift cluster. In addition, multiple customization options have been added to allow you to use persistent volumes with different customer needs.

> Hyperscale Compliance deployment on an OpenShift cluster is supported only for Oracle, MSSQL, Parquet and Delimited Connector.

18.0.0 release

This release supports the following feature/features:

- **Support for mounted filesystems as source and target locations for Parquet connector**
  This release provides support for mounted filesystems for Parquet connectors. You can now provide mounted filesystems as their source and target locations in addition to the already supported AWS S3 buckets if you wish to do so.
- **Support for sharded MongoDB Atlas database**
  This release provides support for Sharded MongoDB databases (Atlas and on-prem) for Mongo connectors. The information within sharded collections from the source database can be masked and transferred into the sharded target database by ensuring the utilization of identical shard keys.

- **Ability to provide a name for the connector**
  This release enhances the connector API to provide a name. For more information, refer Hyperscale Compliance API.

17.0.0 release

This release supports the following feature/features:

- **Addition of Apache Spark as data writer in Delimited Files connector**
  The Delimited files connector now comes with PySpark (Apache Spark) to split the large files with added advantage. You can now select the backend data writer that works for your use case. To know more about these choices, refer to Delimited Files Connector.

16.0.0 release

This release supports the following feature/features:

- **Introduction of the Parquet connector**
  This release introduces a Parquet connector that can be used to mask large Parquet files available on AWS S3 buckets. The connector splits the large files into smaller chunks, passes them to the masking service, and joins them back on the target location.

15.0.0 release

This release supports the following feature/features:

- This release adds an option to configure whether you want to load empty values of Oracle BLOB/CLOB column as Null or empty string. A new property `load_empty_lobs_as_null` has been added under the 'target_config' job to configure the same. It can be configured at the job level. The default value of this property is false (i.e., load as empty string).

14.0.0 release

This release supports the following feature/features:

- **Oracle: Lookup references of a table recursively**
  In this release, we have improved the pre-load process. Now, the pre-load process has been enhanced to perform recursive searches for table references until the final reference is found.
- **Support for masking structured data(XML/JSON) embedded in a database column**
  This release introduces the support for masking the field values in XML / JSON data stored as CLOB in a database column. This has been achieved with the addition of a new entity called `structured-data-format` to the Hyperscale.

13.0.0 release

This release supports the following feature/features:

- **Job Cancellation - Phase 5**
  This release provides an end-to-end Job Cancellation feature, offering you to cancel any Hyperscale Job Execution. All the running processes of Unload, Masking, Load, and Post Load Task will be canceled.
- **Introduction of MongoDB Connector**
  This connector enhances data security by seamlessly masking extensive MongoDB collections. This ensures the continued usability of data while providing robust protection for sensitive information. The connector

now offers enhanced export capabilities, allowing you to split collections based on their preferences. Data masking is seamlessly applied through the dedicated masking service, and the masked data is seamlessly imported into the designated target collection.

- This release introduces a separate artifact known as the MongoDB Profiler that can be downloaded from the same location as Hyperscale. It is an optional tool designed for profiling MongoDB collection data, generating an inventory of sensitive columns, and submitting the payload to the dataset API. The Profiler artifact includes a README file that provides detailed usage instructions. For information on the format of the dataset API payload, refer to the DataSets API section in this document.
- **New execution endpoint**
  This release adds a new API GET endpoint (`/executions/summary`) to the existing JobExecution API to get the summarised overview of all the Executions of a particular Hyperscale Job.

12.0.0 release

This release supports the following feature/features:

- **Job cancellation - Phase 4**
  This release adds the capability for users to cancel a Hyperscale Job Execution once the unload task of the execution has been finished. This will result in the cancellation of all ongoing processes related to masking and load tasks.
- **Introduction of Delimited Files Connector**
  This release introduces a delimited files connector that can be used to mask large delimited flat files (with a delimiter of single character length) available on an NFS location. The connector splits the large into user-provided chunks, passes it to the masking service, and joins back.
- In this release, we're excluding pre-load and post-load processes for empty tables, leading to enhanced performance in scenarios where datasets contain empty tables.

11.0.0 release

This release supports the following feature/features:

- **Job cancellation - Phase 3**
  With this release, you can cancel the MSSQL Hyperscale Job Execution while the load task of the execution is running.
- **Improvements in the Hyperscale job sync feature**
  - This release introduces a new API endpoint `PUT /import/{datasetId}` to update the existing dataset and connector on the Hyperscale Compliance Orchestrator with refreshed ruleset from the Continuous Compliance Engine.
  - This release provides a utility script to automate the process of creating/updating a dataset and connector at Hyperscale, by exporting a masking job from the Continuous Compliance engine.

For more details, refer to How to Sync a Hyperscale Job documentation.

- **Oracle NLS Support for Japanese character set**
  This release adds Oracle NLS Support for the Japanese character set.

10.0.0 release

This release supports the following feature/features:

- **Job cancellation - Phase 2**
  With this release, you can cancel an Oracle Hyperscale Job Execution while the Load task of the execution is running. This feature is not available for MSSQL connectors.

9.0.0 release

This release supports the following feature/features:

- **Job cancellation - Phase 1**
  With this release, you can cancel a Hyperscale Job Execution while the Post Load task of the execution is running. For more details, refer to the How to Cancel a Hyperscale Job documentation.

8.0.0 release

This release supports the following feature/features:

- **Support for Kubernetes deployment**
  This release introduces the capability to deploy the Hyperscale Compliance Orchestrator on a single-node Kubernetes cluster by using the helm charts. For more details, refer to the (8.0.0) Installation and setup (Kubernetes) documentation.
- **Enhanced post-load task status**
  This release provides you with comprehensive visibility into the progress status of the post-load task (for example, ongoing process for constraints, indexes, and triggers) using the execution API endpoints. For more details, refer to the How to Setup a Hyperscale Compliance Job documentation.
- **Oracle connector - degree of parallelism for recreating indexes**
  This release provides you the ability to specify and configure the degree of parallelism(DOP) per Oracle job to recreate the index in the post-load process. Currently, the recreate index DDL utilizes only the default Degree of Parallelism set by the oracle but now you can specify the custom value that can enhance the performance of the index recreation process. For more details, refer to the (8.0.0) Hyperscale Compliance API documentation.
- **Introduction of semantic versioning (Major.Minor.Patch)**
  With this release, Hyperscale introduced support for Kubernetes deployment through helm charts. As helm charts support 3 part Semantic Versioning, hence release 8.0.0 onwards Hyperscale will also follow the 3 part Semantic Versioning instead of 4 parts semantic versioning.

7.0.0.0 release

This release supports the following feature/features:

- **Performance improvement**
  This release introduces impactful changes aimed at enhancing the performance of the masking task within the Hyperscale Job, ultimately resulting in improved overall job performance. The following key changes have been implemented:
  - Changes in Masking Service to increase the Compliance Engine utilization by Hyperscale.
  - Masking Service will no more process tables having 0 rows.
- **Oracle**
  This release supports tables with subpartition indexes during load.

6.0.0.0 release

This release supports the following feature/features:

- **New file based endpoints**
  - file-download: This release introduces a new API endpoint to download the execution and dataset responses. For more information, refer to the (6.0.0) Hyperscale Compliance API documentation.
  - file-upload: This release introduces a new API endpoint to upload a file that currently can be used to create or update the dataset using POST /data-sets/file-upload and PUT /data-sets/file-upload/{dataSetId} endpoints. For more information, refer to the (6.0.0) Hyperscale Compliance API documentation.

- **MSSQL database scalability improvements**
  This release improves the overall job performance by adding the handling of primary key constraints.

5.0.0.1 releases

5.0.0.1 is a patch release specifically aimed at addressing a critical bug. For more information, see (5.0.0) Fixed issues .

5.0.0.0 release

This release supports the following feature/features:

- **MS SQL connector**
  This release adds the MS SQL connector implemented as separate services that include unload and load services. These connector services enable Hyperscale Compliance for MS SQL databases.
- **New execution endpoints**
  This release adds a new API GET endpoint (`/executions/{id}/summary`) to the existing JobExecution API to get the overview of the progress of a Job Execution. In addition to this, the existing `GET /executions/{id}` endpoint has been extended to have additional filters based on the task name and the task's metadata object's status. For more information, refer to the Execution API in the (5.0.0) Hyperscale Compliance API section.
- **Multi-column algorithm support**
  With this release, Multi-Column Algorithms can also be provided in the `/data-sets` endpoint. For more information, refer to the Dataset API in the (5.0.0) Hyperscale Compliance API section. Additionally, existing Continuous Compliance jobs containing multi-column algorithm-related fields can now be imported `via/import` endpoint.

4.1.0 release

This release supports the following feature/features:

- **Capability to limit the number of connections**
  This release adds the capability to limit the number of connections to the source and target databases using the new API parameters as Max_concurrent_source_connection and Max_concurrent_target_connection under new source_configs and target_configs respectively. Using this property, you can fine-tune the number of connections as per source target infra to get better performance. For more information, refer to the (4.1.0) Hyperscale Compliance API documentation.
- **Increased API object limit**
  This release increases the API object limit from 1000 to 10000.

4.0.0 release

This release supports the following feature/features:

- **Hyperscale job sync**
  This release adds the ability to:
  - Import masking jobs inventory from Continuous Compliance engines into connector and dataset info of Hyperscale Compliance Orchestrator with the sync (4.0.0) Accessing the Hyperscale Compliance API endpoint.

    - Import global settings that include Algorithms/Domains from Continuous Compliance Engines to Hyperscale Clustered Continuous Compliance Engines using the sync (4.0.0) Accessing the Hyperscale Compliance API endpoint.
      For more information, refer to the (4.0.0) How to sync a Hyperscale job section.
- **Add configuration properties through .env file**
  This release adds an additional capability to override commonly used configuration properties through the .env file. You can now update application properties in this file before starting the application. For more information, refer to the (4.0.0) Configuration settings section.

3.0.0.1 release

3.0.0.1 is a patch release specifically aimed at addressing critical bugs. For more information, see (3.0.0) Fixed issues .

3.0.0.0 release

This release supports the following feature/features:

- **Oracle connector**
  This release includes the Oracle connector implemented as separate services, including unload and load services. These connector services enable Hyperscale Compliance for Oracle databases.
- **Parallel processing of tables**
  This release processes all tables provided through the data-set API in parallel through the four operational stages - unload, masking, upload, and post-load to minimize the total time it takes to mask the complete data set.
- **Monitoring**
  This release provides monitoring APIs so that you can track the progress of tables in your data set through the unload, masking, upload, and post-load phases. This API also provides a count of rows being processed through different stages.
- **Restartability**
  This release includes the ability to restart a failed process.
- **Clean up**
  This release supports cleaning data from previous job execution.

2.0.0.1 release

2.0.0.1 is a patch release specifically aimed at addressing critical bugs and has the following updates:

- Upgraded spring boot version to 2.5.12.
- Minor view-only changes in swagger-based API client.

2.0.0 release

2.0.0 is the initial release of Hyperscale Compliance. Hyperscale Compliance is an API-based interface that is designed to enhance the performance of masking large datasets. It allows you to achieve faster masking results using the existing Delphix Continuous Compliance offering without adding the complexity of configuring multiple jobs.

Fixed issues

This section describes the issues fixed in Hyperscale Compliance.

Release 2025.1.0

Key	Summary
HM-2593	MSSQL: Hyperscale tries to drop unwanted constraints/indexes/triggers of the auto-selected referenced table which causes referential integrity issues.
HM-4046	Oracle: Hyperscale tries to drop unwanted constraints/indexes/triggers of the auto-selected referenced table which can then cause performance issues if the table has references.
HM-4054	Oracle: The NOVALIDATE option is not preserved when enabling back constraints in post load.

Release 27.0.0

Key	Summary
ECO-10949	When using file connectors, you are no longer required to amend the unload container environment variable USER_PROVIDED_FORMAT_FILE to USER_PROVIDED_FORMAT_FILES in unload-deployment.yaml.
HM-3954	MSSQL: Fixed an issue where empty char and varchar were unloaded as a "" value.
HM-3956	MSSQL: Fixed an issue where leading and trailing whitespaces from a source table were not correctly loaded to the target table.

Release 26.0.0

Key	Summary
HM-3891	When masking in Hyperscale, if there were fewer split files (number of masking jobs on Continuous Compliance engines) than configured Continuous Compliance engines, this slowed down the masking process as the Hyperscale engine was unable to distribute jobs across multiple Continuous Compliance engines.
HM-3917	MSSQL: Fixed an issue where, for some users, column sequences were altered after recreating a clustered index.

Key	Summary
HM-3934	MSSQL: Resolved an issue where the application expected all of the columns of related composite foreign keys to be present in the data set. This issue caused Hyperscale jobs to fail when composite primary or unique keys were partially masked. (Note, not all of the composite key's columns were masked).
HM-3939	MSSQL: When loading data to a target database, an error message was returned because the "null" string value was interpreted as a NULL value. This issue is now fixed.
ECO-569	Resolved an issue that prevented the file connector `values.yaml unloadStorageType` from using Hadoop and FS connectors together. You can now use the Hadoop and FS connectors without restarting the containers.
ECO-9105	Global Spark context is not allowing multiple jobs to maintain their session specific credentials.

Release 25.0.0

There are no fixed issues in this release.

Release 24.0.0

Key	Summary
HM-3685	MSSQL: The JDWP (Java debug wire protocol) debugging configuration caused the unload service to fail startup on hosts that did not support IPv6.
HM-3708	MSSQL: Unload having filter key of datetime type failed with Caused by: java.lang.ClassNotFoundException: org.apache.commons.text.StringSubstitutor
HM-3733	MSSQL: The masking date/timestamp column would only work when using the default date format.
HM-3740	MSSQL: Fixed an issue that caused MSSQL jobs to fail at load service if the target database table had a column with "Time" datatype.
HM-3756	MSSQL - Masking stuck in 'Running' for table with no unique/primary key.
HM-3775	Resolved an issue with Hyperscale Oracle Connector that caused jobs to fail in load service. This error: "Process terminated with exit code 2" occurred if data contained enclosure characters that got truncated during masking with Continuous Compliance versions 20.0.0 to 26.0.0.

Release 23.0.0

Key	Summary
HSC-740	Replace Oracle JDK-21 with openjdk17
HM-3554	[MSSQL]-FILLFACTOR attribute is not included while recreating index on post load
HM-3630	Unload failed with date format exception for "TIMESTAMP WITH TIME ZONE" and "TIMESTAMP WITH TIME ZONE" data types column
HM-3678	MSSQL- Unable to truncate table if having view references

Release 22.0.0

Key	Summary
HM-3530	Hyperscale Masking Hangs due to WARNING job status introduced in CC v22
HM-3549	Hyperscale Job failed during masking phase with "String index out of range" error with continuous compliance version between 20.0.0 to 23.0.0
HM-3557	HS Job execution stuck in RUNNING state if Masking Task has completely FAILED and number of rows masked > 0
HM-3560	MSSQL: Failed to load data in date type column if timestamp column is also present in table.
HSC-713	Parquet connector > failing to write data: For connector-type = FS, an intermittent issue exists where target files generated are not parquet files. For connector-type = AWS, source file name containing the substring "part-", result in empty target file

Release 21.0.0

There are no fixed issues in this release.

Release 20.0.0

Key	Summary
HSC-550	Parquet connector will be able to process decimal type of data as well

Key	Summary
HSC-558	Delimited load service with “perform_join=false” fails for multiple source files in a single data set for S3 target location.

Release 19.0.0

Key	Summary
HM-3215	Masking service container json log grows too large and the huge log may cause controller service down because of local filesystem full
HSC-525	Delimited unload service for multiple source files for a single source key is being FAILED by the controller

Release 18.0.0

Key	Summary
HM-3100	Index creation on Local Partition table fails with ORA-14024
HSC-454	With pySpark we are removing the leading space for numbers

Release 17.0.0

Key	Summary
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error
HM-2962	Incorrect gathering target table group list in case of the table refered from other tables.
HSC-176	Delimited connector: values with data type double and int64 will always be converted into a string
HSC-177	Delimited connector: Irrespective of user provided enclosure character, the output strings will be quoted using double quotes
HSC-279	Wrong counts in Unload: Using HS 14 delimited file connector, while masking two files in the dataset, job shows failed status, because the counts are all wrong

Key	Summary
HSC-286	Load failed because the SQLite3 database locked up. Masking 15 files with the same data set. Unload has the wrong count.
HSC-361	Hyperscale delimited connector should not add double quotes to output file

Release 16.0.0

Key	Summary
HM-2748	Unable to upload sync bundle of more than 20 MB

Release 15.0.0

Key	Summary
HM-2505	ServiceUnavailableException in Masking Service as numeric value out of range
HM-2554	Missing security context causes controller to fail with permission denied for JDBC connection to SQLite DB
HM-2609	Configuration added to load empty values of oracle CLOB/BLOB columns as null or empty string

Release 14.0.0

Key	Summary
HSC-182	400 Bad Request from POST http://unload-service:8080/api/unload
HM-2413	NTRS: Pre Load failed with errors: Error executing statement: Error disabling constraint
HM-2505	ServiceUnavailableException in Masking Service as numeric value out of range
HM-2554	Missing security context causes controller to fail with permission denied for JDBC connection to SQLite DB
HM-2568	Hyperscale API's working without auth key in k8s setup

Release 13.0.0

Key	Summary
HSC-178	Delimited load service will show the status as FAILED while it is waiting for all split files to be masked in order to perform join.
HSC-179	While executing masking against a 1.1 TB second time, the load job failed mid-execution.
HM-2399	MSSQL: The lower bound of partitioning column is larger than the upper bound.
HM-2443	Masking service is mapping the wrong file metadata when multiple data_info objects with varying delimiters are passed during data-sets creation.

Release 12.0.0

Key	Summary
HM-2268	Add support to configure FileMetadata related characters for structured (i.e. XML) data
HM-2344	sync-compliance-engine endpoint failing with "Invalid input" error
HM-2399	MSSQL: The lower bound of partitioning column is larger than the upper bound.

Release 11.0.0

There are no fixed issues in this release.

Release 10.0.0

Key	Summary
HM-1360	Job is created with default value of 'retain_execution_data' parameter if an invalid value for the same parameter is passed while creating the job object
HM-2041	Capture high-level processing timings for each process for each object
HM-2084	cancel script (cancel.sh) does not read values from .env file
HM-2199	Masking service paginated GET call, skip the result of last page

Release 9.0.0

Key	Summary
HM-1845	java.sql.SQLException: Protocol violation While fetching metadata of the table
HM-1980	Post load status incorrectly reporting consolidated start and end timing.
HM-2017	The masking process is stuck in running when a newly registered engine is used in execution after the timeout time of the Engine

Release 8.0.0

Key	Summary
HM-1606	When tables have more than 100 columns on Oracle, the default API page at 100 causes an issue where we don't match the column name via the masking API
HM-1787	Oracle: Load service holds a lock on class level, causing parallel jobs stuck in the flow of preload
HM-1814	MSSQL: Load service holds the lock on class level, causing parallel jobs stuck in the flow of preload

Release 7.0.0.0

Key	Summary
HM-1617	Oracle - Hyperscale engine fails to mark partitions of the partitioned index as unusable in pre-load and fails to rebuild a partitioned index in post load leaving indexes in an unusable state
HM-1684	Running multiple Hyperscale jobs in parallel, using the same set of Masking Engines and the same value of env_name_prefix and app_name_prefix can cause job failure with the missing ruleset, any other masking objects, or execution error.

Release 6.0.0.0

Key	Summary
HM-1513	MSSQL - Slowness while performing load with large tables(more than 4M rows and 10 Columns)

Key	Summary
HM-1521	MSSQL - Post load fails with 'transaction log is full due to 'ACTIVE_TRANSACTION' for large tables
HM-1608	SQLLDR doesn't load the data when the table has enabled triggers owned by a different user
HM-1645	Failed clean up of previous execution causes "\"File Format already exists with identifier: HYPERSCALE_16_9fd11fae45861fdb18fb658fb950ceab.fmt\" issue for next execution of same job
HM-1656	nginx configuration client max body size limits the size of the payload to post for the dataset

Release 5.0.0.1

Key	Summary
HM-1561	Oracle Load Failure: SQL loader control files doesn't contain character length when column size is less than 256 CHAR

Release 5.0.0.0

Key	Summary
HM-971	For HTTP protocol-type requests, trust store fields are accepted and displayed in response.
HM-1472	Oracle Unload service doesn't release the lock if fails to initialize the connection pool and the job stuck in a runnning state
HM-1520	Masking service: Starting execution throwing NPE after container restart
HM-1522	Update Oracle JDBC Driver to 21.3.0.0

Release 4.1.0

Key	Summary
HM-1155	Diagnosability: How do I tell which masking job on the masking engine relates to the failed message on the HS Jobs API status
HM-1168	The error text in Post Load failures is misleading/unclear
HM-1191	Optimization: We should execute Select Count (*) in parallel to the Oracle Unload process. It could take a significant amount of time to count data in large tables as well as 1000 objects.
HM-1201	Unable to import a ruleset with 5000 tables to HS 4.0, getting an error message.
HM-1210	While trying to process an HS job for 5000 table schema, ran out of Oracle cursor, and then the SQLite database locked up.
HM-1265	Oracle - Any index on a column having no constraint on it, is not getting dropped
HM-1334	Can't drop the index, because the index is not owned by the user we used to connect.
HM-1378	Oracle - Load service needs to include index owner name when attempting to modify/rebuild partition indexes owned by different db user

Release 4.0.0

Key	Summary
HM-177	Able to POST /hyperscale-masking/jobs with min job memory > max job memory
HM-530	POST/PUT request dataSet API error response received with empty/missing/invalid ‘source’/’target’ object/values can be improved
HM-754	POST/PUT connector jdbc_url, username, and password should be mandatory for Oracle load and unload service
HM-789	Error message upon not setting the ‘SSL’ field to False indicates ‘insecure_ssl’ property which no longer exists in the schema

Key	Summary
HM-858	Status of sub-task coming wrong when overall execution failed
HM-932	Suppress the password message for the controller log
HM-1138	The description in the swagger doesn't match the API call
HM-1140	The error needs more information to diagnose a connector issue.

Release 3.0.0.1

Key	Summary
HM-858	Status of sub-task coming wrong when overall execution failed
HM-873	Intermittently there is a mismatch in loaded_rows displayed in the load task vs the actual rows loaded in the target table
HM-915	Load: driver support plugin throws ORA-02297: cannot disable constraint - dependencies exist error for foreign key

Release 3.0.0

Key	Summary
HM-294	The updated file format is not POST'ed on the Continuous Compliance Engine if the file format name is the same

Known issues

This section describes the known issues in Hyperscale Compliance.

Release 2025.1.0

Key	Summary	Workaround
DLPXECO-11402	Snowflake Connector – Jobs may intermittently fail with larger data sizes (over 500 GB) when S3 is used as the staging area.	None
HSC- 759 (ECO-9149)	Pyspark doesn’t support INT64 timestamp with nanoseconds for parquet file_types.	You can use pyarrow writer as an alternative.
HM-3947	Oracle: The application expects all of the columns of related composite foreign keys to be present in the Data Set and as a result it fails the job execution when composite primary/unique keys are partially masked (not all of the composite key's columns are masked).	None
HM-3636	MSSQL: Unexpected movement of count from constraints (Primary & Unique) to index in postload response.	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-663	Oracle : Load process is failing with "Error disabling constraint" for identity columns.	None

Release 27.0.0

Key	Summary	Workaround
HSC- 759 (ECO-9149)	Pyspark doesn’t support INT64 timestamp with nanoseconds for parquet file_types.	You can use pyarrow writer as an alternative.

Key	Summary	Workaround
HM-3636	MSSQL: Unexpected movement of count from constraints (Primary & Unique) to index in postload response.	None
HM-2593	MSSQL - In PreLoad table references not fetched until last level in relationship hierarchy.	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-663	Oracle : Load process is failing with "Error disabling constraint" for identity columns.	None

Release 26.0.0

Key	Summary	Workaround
HSC- 759	Pyspark doesn’t support INT64 timestamp with nanoseconds for parquet file_types.	You can use pyarrow writer as an alternative.
HM-3636	MSSQL: Unexpected movement of count from constraints (Primary & Unique) to index in postload response.	None
HM-2593	MSSQL - In PreLoad table references not fetched until last level in relationship hierarchy.	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-663	Oracle : Load process is failing with "Error disabling constraint" for identity columns.	None

Release 25.0.0

Key	Summary	Workaround
HSC-725	The spark session isn't getting modified for different connector-infos such as FS, AWS-with-credentials, AWS-without-credentials.	None

Key	Summary	Workaround
HSC- 759	Pyspark doesn’t support INT64 timestamp with nanoseconds for parquet file_types.	You can go for pyarrow writer.
HM-3636	MSSQL: Unexpected movement of count from constraints (Primary & Unique) to index in postload response.	None
HM-2593	MSSQL - In PreLoad table references not fetched until last level in relationship hierarchy.	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-663	Oracle : Load process is failing with "Error disabling constraint" for identity columns.	None

Release 24.0.0

Key	Summary	Workaround
HSC-725	The spark session isn't getting modified for different connector-infos such as FS, AWS-with-credentials, AWS-without-credentials.	None
HSC- 759	Pyspark doesn’t support INT64 timestamp with nanoseconds for parquet file_types.	You can go for pyarrow writer.
HM-3636	MSSQL: Unexpected movement of count from constraints (Primary & Unique) to index in postload response.	None
HM-2593	MSSQL - In PreLoad table references not fetched until last level in relationship hierarchy.	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-663	Oracle : Load process is failing with "Error disabling constraint" for identity columns.	None

Release 23.0.0

Key	Summary	Workaround
HSC-725	The spark session isn't getting modified for different connector-infos, i.e FS, AWS-with-credentials, AWS-without-credentials, etc	None
HSC-759	Pyspark doesn’t support INT64 timestamp with nanoseconds for parquet file_types	You can go for pyarrow writer.
HM-3685	MSSQL: The Unload service fails startup on hosts that do not support IPv6. This issue is caused by the JDWP (Java debug wire protocol) debugging configuration.	For Docker or Podman Compose, add the environment variable JAVA_TOOL_OPTIONS having value agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=0.0.0.0:9093 under `unload-service` in the `docker-compose.yaml` or `podman-compose.yaml` file. For Kubernetes, include it in `unload-deployment.yaml`
HM-3636	MSSQL: Unexpected movement of count from constraints (Primary & Unique) to index in postload response	None
HM-2593	MSSQL - In PreLoad table references not fetched till last level in relationship hierarchy	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-663	Oracle : Load process is failing with "Error disabling constraint" for identity columns	None

Release 22.0.0

Key	Summary	Workaround
HM-3554	[MSSQL]-FILLFACTOR attribute is not included while recreating index on post load	None

Key	Summary	Workaround
HM-2593	MSSQL - In PreLoad table references not fetched till last level in relationship hierarchy	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-715	Parquet connector: compression of source parquet files do not match compression of target parquet files - when using PySpark writer.	None
HM-709	Parquet connector: There is an issue when the source file paths have folder names with only number, ex: '/source/files/1/*'.	None
HM-701	Parquet connector: Providing a delimiter character other than the default character comma will result in a issue when using PySpark writer	None
HM-663	Oracle : Load process is failing with "Error disabling constraint" for identity columns	None

Release 21.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-2593	MSSQL - In PreLoad table references not fetched till last level in relationship hierarchy	None

Release 20.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None

Key	Summary	Workaround
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-2593	MSSQL - In PreLoad table references not fetched till last level in relationship hierarchy	None

Release 19.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-2593	MSSQL - In PreLoad table references not fetched till last level in relationship hierarchy	None
HSC-558	Delimited load service with "perform_join=false" fails for multiple source files in a single data set for S3 target location	None

Release 18.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-2593	MSSQL - In PreLoad table references not fetched till last level in relationship hierarchy	None

Release 17.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-2593	MSSQL - In PreLoad table references not fetched till last level in relationship hierarchy	None

Release 16.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None
HM-2593	MSSQL - In PreLoad table references not fetched till last level in relationship hierarchy	None

Release 15.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Key	Summary	Workaround
HM-2593	MSSQL - In PreLoad table references not fetched till last level in relationship hierarchy	None

Release 14.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None
HM-2413	In PreLoad table references not fetched till last level in relationship hierarchy	None

Release 13.0.0

Delimited Files Connector

Key	Summary	Workaround
HSC-174	Delimited unload service does not show incremental unload row count, it always should unloaded count is the same as the total count	None
HSC-176	Delimited connector: values with data type double and int64 will always be converted into a string	None
HSC-177	Delimited connector: Irrespective of user provided enclosure character, the output strings will be quoted using double quotes	None

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None
HM-2413	In PreLoad table references not fetched till last level in relationship hierarchy	None

Release 12.0.0

Delimited Files Connector

Key	Summary	Workaround
HSC-174	Delimited unload service does not show incremental unload row count, it always should unloaded count is the same as the total count	None
HSC-176	Delimited connector: values with data type double and int64 will always be converted into a string	None
HSC-177	Delimited connector: Irrespective of user provided enclosure character, the output strings will be quoted using double quotes	None
HSC-178	Delimited load service will show the status as FAILED while it is waiting for all split files to be masked in order to perform join	None
HM-2443	Masking service is mapping the wrong file metadata when multiple data_info objects with varying delimiters are passed during data-sets creation.	Create individual data-sets for files with different delimiter character.

Release 11.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 10.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 9.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for tables having triggers only with SQL*Loader-937 error	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 8.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for tables having triggers only with SQL*Loader-937 error	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 7.0.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle Load fails for tables having triggers only with SQL*Loader-937 error	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 6.0.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-1397	Oracle: Dataset having any one entry with invalid schema leaves indexes of other tables as UNUSABLE	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 5.0.0.0

Key	Summary	Workaround
HM-291	Hyperscale job execution with an intelligent load balancer configured is stuck in a loop if the job's max memory is more than totalAllocatedMemoryForJobs	Change the max memory to a value under the value of `totalAllocatedMemoryForJobs` the property configured on the Continuous Compliance Engine.
HM-652	Job execution is stuck in a running state if the mount server is powered off	Check the health of the mount server before starting a job.
HM-663	Oracle: Load process is failing with "Error disabling constraint" for identity columns	None
HM-718	Not all data on the mount server is cleaned up if the continuous compliance engine is stopped	Cleanup up the data manually from the mount server.
HM-745	The table name is not present in the error message while enabling/disabling triggers, indexes, constraints	Check the logs in container logs to get table details
HM-817	Intermittently job fails with ORA-02270: no matching unique or primary key for this column-list	Restart the job using `PUT /executions/{id}/restart` and it will succeed.
HM-821	Hyperscale job does not handle post-load task properly during restart if failed in pre-load (disabling trigger/indexes/constraints) steps	After job execution is completed successfully, check and manually enable the disabled constraints.
HM-1251	MSSQL: Row is not loaded if masked BIT type column has values other than true(1),false(0) or NULL	None
HM-1366	NPE displayed in hyperscale masking service logs just after masking task is done	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-1512	received_objects in Load step are more than succeeded_objects in Masking step intermittently	None

Key	Summary	Workaround
HM-1513	MSSQL - Slowness while performing load with large tables(more that 4M rows and 10 Columns)	None
HM-1521	MSSQL - Post load fails with 'transaction log is full due to 'ACTIVE_TRANSACTION' for large tables	None
HM-1528	Initial delay in updating response of unload/masking/ load execution objects with large number of tables	None
HM-1561	Oracle Load Failure: sql loader control files doesn't contain character length when column size is less than 256 CHAR	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 4.1.0

Key	Summary	Workaround
HM-291	Hyperscale job execution with intelligent load balancer configured is stuck in a loop if job's max memory is more than totalAllocatedMemoryForJobs	Change the max memory to a value under the value of `totalAllocatedMemoryForJobs` property configured on Continuous Compliance Engine.
HM-652	Job execution is stuck in running state if mount server is powered off	Check the health of mount server before starting a job.
HM-663	Load process is failing with "Error disabling constraint" for identity columns	None
HM-718	Not all data on mount server is cleaned up if masking engine is stopped	Cleanup up the data manually from the mount server.
HM-745	Table name is not present in error message while enabling/disabling triggers,indexes,constraints	Check the logs in container logs to get table details

Key	Summary	Workaround
HM-817	Intermittently job fails with ORA-02270: no matching unique or primary key for this column-list	Restart the job using `PUT /executions/{id}/restart` and it will succeed.
HM-821	Hyperscale job does not handle post load task properly during restart if failed in pre-load (disabling trigger/indexes/constraints) steps	After job execution is completed successfully, check and manually enable the disabled constraints.
HM-1155	Diagnosibility: How do I tell which masking job on the masking engine relates to the failed message on the HS Jobs API status	Check the error details in masking service logs
HM-1168	The error text is inaccurate, and doesn't contain enough information to diagnose it without accessing logs on the Hyperscale server.	Check the error details in the logs
HM-1561	Oracle Load Failure: sql loader control files doesn't contain character length when column size is less than 256 CHAR	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 4.0.0

Key	Summary	Workaround
HM-291	Hyperscale job execution with intelligent load balancer configured is stuck in a loop if job's max memory is more than totalAllocatedMemoryForJobs	Change the max memory to a value under the value of `totalAllocatedMemoryForJobs` property configured on Continuous Compliance Engine.
HM-652	Job execution is stuck in running state if mount server is powered off	Check the health of mount server before starting a job.
HM-663	Load process is failing with "Error disabling constraint" for identity columns	None
HM-718	Not all data on mount server is cleaned up if masking engine is stopped	Cleanup up the data manually from the mount server.

Key	Summary	Workaround
HM-745	Table name is not present in error message while enabling/disabling triggers,indexes,constraints	Check the logs in container logs to get table details
HM-817	Intermittently job fails with ORA-02270: no matching unique or primary key for this column-list	Restart the job using `PUT /executions/{id}/restart` and it will succeed.
HM-821	Hyperscale job does not handle post load task properly during restart if failed in pre-load (disabling trigger/indexes/constraints) steps	After job execution is completed successfully, check and manually enable the disabled constraints.
HM-1366	NPE displayed in hyperscale masking service logs just before cleanup is performed	None
HM-1397	Load fails for table having triggers only with SQL*Loader-937 error	None
HM-1561	Oracle Load Failure: sql loader control files doesn't contain character length when column size is less than 256 CHAR	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 3.0.0.1

Key	Summary	Workaround
HM-177	Able to POST /hyperscale-masking/jobs with min job memory > max job memory	Change the max job memory value to higher than min job memory in API request.
HM-291	Hyperscale job execution with intelligent load balancer configured is stuck in a loop if job’s max memory is more than totalAllocatedMemoryForJobs	Change the max memory to a value under the value of `totalAllocatedMemoryForJobs` property configured on Continuous Compliance Engine.
HM-652	Job execution is stuck in running state if mount server is powered off	Check the health of mount server before starting a job.

Key	Summary	Workaround
HM-663	Load process is failing with “Error disabling constraint” for identity columns	None
HM-684	Hypescale does not support other TIMESTAMP(6) datatype variations apart from TIMESTAMP	None
HM-718	Not all data on mount server is cleaned up if batch masking service is stopped	Cleanup up the data manually from mount server.
HM-745	Table name is not present in error message while enabling/disabling triggers,indexes,constraints	Check the logs in container logs to get table details.
HM-754	Able to POST/PUT a connector with whitespace as jdbc_url, username, password	Remove white space and use valid values for jdbc_url, username and password.
HM-789	Error message upon not setting ‘ssl’ field to False indicates ‘insecure_ssl’ property which no longer exists in the schema	None
HM-817	Intermittently job fails with ORA-02270: no matching unique or primary key for this column-list	Restart the job using `PUT /executions/{id}/restart` and it will succeed.
HM-821	Hyperscale job does not handle post load task properly during restart if failed in pre-load (disabling trigger/indexes/constraints) steps	After job execution is completed successfully, check and manually enable the disabled constraints.
HM-935	Load service fails when source DB contains BLOB type data that is not simple text file data	None
HM-1561	Oracle Load Failure: sql loader control files doesn't contain character length when column size is less than 256 CHAR	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 3.0.0

Key	Summary	Workaround
HM-177	Able to POST /hyperscale-masking/jobs with min job memory > max job memory	Change the max job memory value to higher than min job memory in API request.
HM-291	Hyperscale job execution with intelligent load balancer configured is stuck in a loop if job's max memory is more than totalAllocatedMemoryForJobs	Change the max memory to a value under the value of `totalAllocatedMemoryForJobs` property configured on Continuous Compliance Engine.
HM-652	Job execution is stuck in running state if mount server is powered off	Check the health of mount server before starting a job.
HM-663	Load process is failing with "Error disabling constraint" for identity columns	None
HM-684	Hypescale does not support other TIMESTAMP(6) datatype variations apart from TIMESTAMP	None
HM-718	Not all data on mount server is cleaned up if batch masking service is stopped	Cleanup up the data manually from mount server.
HM-745	Table name is not present in error message while enabling/disabling triggers,indexes,constraints	Check the logs in container logs to get table details.
HM-754	Able to POST/PUT a connector with whitespace as jdbc_url,username,password	Remove white space and use valid values for jdbc_url, username and password.
HM-789	Error message upon not setting 'ssl' field to False indicates 'insecure_ssl' property which no longer exists in the schema	None
HM-817	Intermittently job fails with ORA-02270: no matching unique or primary key for this column-list	Restart the job using `PUT /executions/{id}/restart` and it will succeed.
HM-821	Hyperscale job does not handle post load task properly during restart if failed in pre-load (disabling trigger/indexes/constraints) steps	After job execution is completed successfully, check and manually enable the disabled constraints.

Key	Summary	Workaround
HM-858	Status of sub task coming wrong when overall execution failed	None
HM-873	Intermittently there is a mismatch in loaded_rows displayed in load task vs the actual rows loaded in target table	None
HM-915	Load: driver support plugin throws ORA-02297: cannot disable constraint - dependencies exist error for foreign key	None
HM-935	Load service fails when source DB contains BLOB type data that is not simple text file data	None
HM-1561	Oracle Load Failure: sql loader control files doesn't contain character length when column size is less than 256 CHAR	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 2.0.0

The following is a list of the known issues in the Hyperscale Compliance version 2.0.0.

Key	Summary	Workaround
HM-294	Updated file format is not POST'ed on the Continuous Compliance Engine if file format name is same	• After modifying the file format content, rename the file name of the file format. • Delete the existing uploaded file format from the attached Continuous Compliance Engines before executing the Hyperscale job with an updated file format.
HM-291	Hyperscale job execution with intelligent load balancer configured is stuck in a loop if job's max memory is more than totalAllocatedMemoryForJobs	None

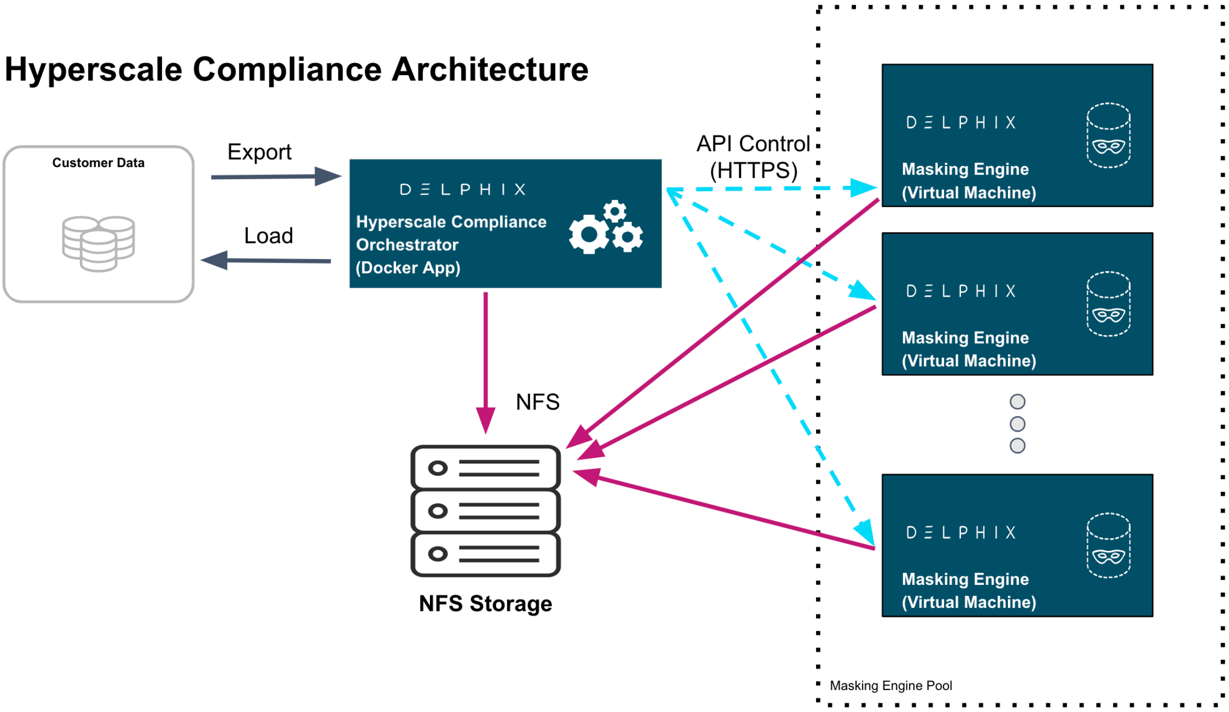
Key	Summary	Workaround
HM-216	POST/PUT /data-sets accepts duplicate values as source_files path in Single File Info Object	None
HM-177	Able to POST /hyperscale-masking/jobs with min job memory > max job memory	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Overview

Hyperscale Compliance is an API-based interface that is designed to enhance the performance of masking large datasets. It allows you to achieve faster masking results using the existing Delphix Continuous Compliance offering without adding the complexity of configuring multiple jobs. Hyperscale Compliance first breaks the large and complex datasets into numerous modules and then orchestrates the masking jobs across multiple Continuous Compliance Engines. In general, datasets larger than 10 TB in size will see improved masking performance when run on the Hyperscale architecture. refer to <section name> on <page>.

Hyperscale Compliance deployment architecture

For achieving faster masking results, Hyperscale Compliance uses bulk import or export utilities of data sources. Using these utilities, it exports the data into smaller chunks of delimited files. The Hyperscale Compliance Orchestrator then configures the masking jobs of all the respective chunks across multiple Continuous Compliance Engines. Upon successful completion of the masking jobs, the masked data is imported back into the database.


Hyperscale Compliance Architecture
Customer Data
Export
Load
DELPHIX
Hyperscale Compliance Orchestrator (Docker App)
API Control (HTTPS)
NFS
NFS Storage
DELPHIX
Masking Engine (Virtual Machine)
DELPHIX
Masking Engine (Virtual Machine)
DELPHIX
Masking Engine (Virtual Machine)
Masking Engine Pool

Hyperscale Compliance components

The Hyperscale Compliance architecture consists of four components mainly; the Hyperscale Compliance Orchestrator, Source/Target Connectors, the Continuous Compliance Engine Cluster, and the Staging Server.

Hyperscale Compliance Orchestrator

The Hyperscale Compliance Orchestrator is responsible for unloading the data from the source and horizontally scaling the masking process by initiating multiple parallel masking jobs across nodes in the Continuous Compliance Engine cluster. Once data is masked, it loads it back to the target data sources. Depending on the number of nodes in the cluster, you can increase or decrease the total throughput of an individual masking job. In the case of relational databases as source and target data sources, it also handles the pre-load (disabling indexes, triggers, and constraints) and post-load (enabling indexes, triggers, and constraints) tasks like disabling and enabling indexes, triggers, and constraints. Currently, the Hyperscale Compliance Orchestrator supports the following two strategies to distribute the masking jobs across nodes available :

- **Intelligent Load Balancing (Default)**: This strategy considers each Continuous Compliance Engine's current capacity before assigning any masking jobs to the node Continuous Compliance Engines. It calculates the capacity using available resources on node Continuous Compliance Engines and already running masking jobs on the engines. Below is the formula used to calculate the capacity of the Continuous Compliance Engines:

```
Engine’s current jobCapacity = Engine’s total jobCapacity - no of currently running
jobs on Engine

Engine’s total jobCapacity = Minimum of {CapacityBasedOnMemory, CapacityBasedOnCores}

where
CapacityBasedOnMemory = (TotalAllocatedMemoryForJobs on Engine / MaxMemory assigned
to each Engine Job)
CapacityBasedOnCores = [Engine’s CpuCoreCount - 1]
```

- **Round robin load balancing**: This strategy simply distributes the masking jobs to all the node Continuous Compliance Engines using the round robin algorithm.

Staging area

The Staging Area is where data from the SOR is unloaded to a series of files by the Hyperscale Compliance Orchestrator. It can be a file system that supports the NFS protocol. The file system can be attached to volumes, or it can be supplied via the Delphix Continuous Data Engine empty VDB feature. In either case, there must be enough storage available to hold the dataset in an uncompressed format. The staging area should be accessible by the Continuous Compliance Engine cluster as well for masking.

Continuous Compliance Engine cluster

The Continuous Compliance Engine Cluster is a group of Delphix Continuous Compliance Engines (version 6.0.14.0 and later) leveraged by the Hyperscale Compliance Orchestrator to run large masking jobs in parallel. For installing and configuring the Continuous Compliance Engine procedures, see Continuous Compliance Documentation.

Source and target data sources

The Hyperscale Compliance Orchestrator is responsible for unloading data from the source data source into a series of files located in the staging area. The Hyperscale Compliance Orchestrator requires network access to the source from the host running the Hyperscale Compliance Orchestrator and credentials to run the appropriate unload commands. After files are masked, the masked data from the files get uploaded to the target data source.

In the case of Oracle and MS SQL data sources, a failure in the load may leave the target data source in an inconsistent state since the load step truncates the target when it begins. If the source and target data source are configured to be the same data source and a failure occurs in the load step, it is recommended that the single data source be restored from a backup (or use the Continuous Data Engine's rewind feature if you have a VDB as the single data source) after the failure in the load step as the data source may be in an inconsistent state. After the data source is restored, you may proceed to kick off another hyperscale job. If the source and target data source are configured to be different, you may use the Hyperscale Compliance Orchestrator restart ability feature to restart the job from the point of failure in the load/post-load step.

Additional data sources:

- **File Connector:** File connector enables efficient masking of large delimited and parquet files at hyperscale. Data can be sourced from or stored in various locations, including NFS, AWS S3, or Hadoop file systems. Users have the flexibility to select the appropriate writer type for both data unload and load operations. During the unload process, data is fetched from the specified source, split into manageable files, and transmitted to the compliance engine. For load operations, data is written to the designated target location.
- **MongoDB Connector:** We support Hyperscale masking of large MongoDB database collections. Load step of MongoDB connector drops the target database collection if it is already present. In Hyperscale MongoDB connector, we strongly recommend DONOT use the same collection for both the source and target, particularly when dealing with same MongoDB instances. Utilizing the same collection for in-place masking in such a scenario can pose risks, including potential data deletion, especially when unload, masking, and load operations are occurring asynchronously. It's crucial to maintain a clear separation between source and target entities to ensure data integrity and avoid unintended consequences.
- **Snowflake Connector**: We support Hyperscale masking of Snowflake’s platform. Load step of Snowflake connector drops the target database table if it is already present. In Hyperscale Snowflake connector, we strongly recommend DONOT use the same table for both the source and target database, particularly when dealing with same Snowflake warehouse. Utilizing the same table for in-place masking in such a scenario can pose risks, including potential data deletion, especially when unload, masking, and load operations are occurring asynchronously. It's crucial to maintain a clear separation between source and target entities to ensure data integrity and avoid unintended consequences.

In-Place Masking is NOT supported.

The Continuous Compliance platform

Delphix Continuous Compliance is a multi-user, a browser-based web application that provides complete, secure, and scalable software for your sensitive data discovery, masking, and tokenization needs while meeting enterprise-class infrastructure requirements. To read further about Continuous Compliance features and architecture, read the Continuous Compliance Documentation.

Next steps

- Read about Installation and Setup (Kubernetes) .
- Read about the Network requirements .
- Read about Accessing the Hyperscale Compliance API .

Getting started

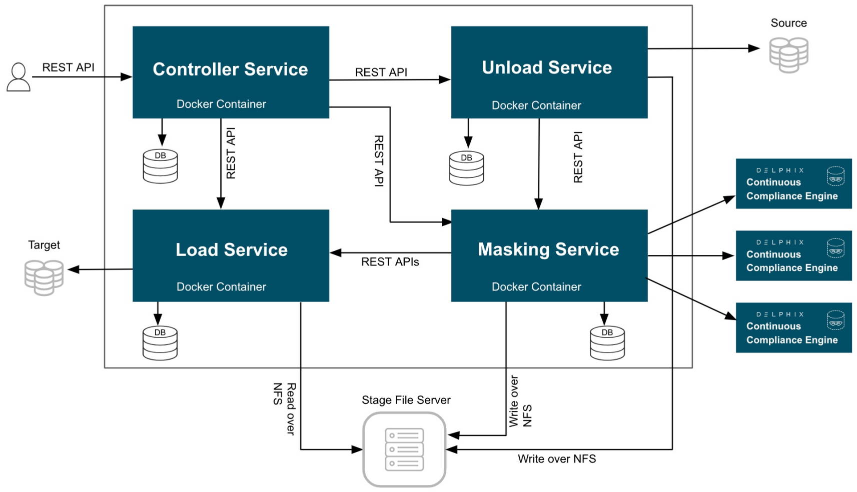
This section covers the following topics:

- Hyperscale Compliance architecture
- Data source support
- Supported platforms
- Network requirements
- Deployment
- NFS server installation
- Configuring AWS S3 bucket as staging area
- Configuring Azure Blob containers as staging area
- Accessing the Hyperscale Compliance API

Hyperscale Compliance architecture

The Hyperscale Compliance architecture comprises four components mainly; Controller Service, Unload Service, Masking Service, and Load Service.

Hyperscale Compliance Architecture



Controller service

The following are the main functions of a controller service:

- Exposes user-accessible API.
- Once the controller service receives user requests (for example, register engine, create a dataset, create a connector, create Job, etc.), it will split the request and sends a request for further processing to downstream services (Unload, Masking, Load) and once response is received from downstream service, the same will be processed by controller service and returned to the user.
- The controller service accepts request job execution from the user and invokes the job execution process by invoking unload service asynchronously.
- The controller service will keep polling data job execution data from the downstream service until execution completes.
- The controller service will also determine the status of job execution and store execution data in the database.
- Controller service allows you to restart a failed (Failed during File Loader, Post Load) execution

Unload service

The following are the main functions of a unload service:

- Exposes APIs that are accessible to internal services only.
- Unload service exposes required APIs that help the caller (controller service) to create required inputs (source info, dataset, etc.) for job execution.
- Unload service exposes an API to trigger unload from the source data source. As part of the unload process, it performs the following operations:
    - Reads metadata of source data source (e.g. number of rows in a source file/table) and stores that in the unload service database.

    - Reads data from source data source parallelly (by starting multiple parallel processes for each source entity like tables in case of a relational database) and stores this data in `.csv` files.
    - Once data is loaded into one `.csv` file, unload service triggers the masking service to start the masking process for that `*.csv` file.
- For running execution, Unload service maintains metadata data (number of rows processed, table/file names processed, etc.) in its database. This data can be retrieved by calling an API.
- Once execution completes execution data in the database and file system gets cleaned by invoking the corresponding API.

Masking service

The following are the main functions of a masking service:

- Exposes APIs that are accessible to internal services only.
- Masking services expose required APIs that help the caller (controller service) to create required inputs (Continuous Compliance engine info, dataset, job, etc.) for job execution.
- Masking service exposes an API to trigger the masking process. As part of the masking process, it performs the following operations after receiving a masking request from unload service for a CSV file:
    - Based on Intelligent load balancing, create and start jobs for unloaded files on Continuous Compliance Engines (based on the capacity of Continuous Compliance Engines associated with the hyperscale job).
    - Monitor Continuous Compliance Engine jobs triggered in the previous step.
    - Once monitoring determines that a Continuous Compliance Engine has successfully masked the file, send an async request to the load service (to load data into the target data source) for that masked file.
- For running execution, the Masking service maintains metadata data (number of rows processed, table/file names processed, etc.) in its database. This data can be retrieved by calling an API.
- Once execution completes execution data in the database and file system gets cleaned by invoking the corresponding API.

Load service

The following are the main functions of a Load service:

- Exposes APIs that are accessible to internal services only.
- Load service exposes required APIs that help the caller to create required inputs (target data source info, dataset, job, etc.) for job execution.
- Load service exposes an API to trigger the Load process. As part of the Load process, it performs the following operations after receiving a load request from the masking service for a masked CSV file:
    - Perform preload step (for example, cleaning up the target directory or disabling constraints/triggers/indexes). These may be performed once for an execution process (not for each request from the masking service).
    - Load masked files into the target data source.
    - Once Loading for a masked is completed, the metadata for this “file load“ will be stored in the load service database.
- For running execution, the Load service maintains metadata data (number of rows processed, table/file names processed, etc.) in its database. This data can be retrieved by calling an API.
- Once execution completes execution data in the database and file system gets cleaned by invoking the corresponding API.
- If the Load service is for a data source that requires post-load steps (e.g. Oracle, MS SQL), then it will include post-load steps which will be triggered by the controller service once all files are successfully loaded into the target data source.

- Load service also allows restarting for the post-load step, if post-load fails for an execution.

Data source support

Oracle connector

Oracle Database (commonly referred to as Oracle RDBMS or simply as Oracle) is a multi-model database management system produced and marketed by Oracle Corporation. The following table lists the versions that have been tested in the lab setup:

Platforms	Version
Linux	• Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production - AWS • Oracle Database 18c Enterprise Edition Release 18.0.0.0.0 - Production - GCP

- User on source database must select privileges
- User on target database side must have all privileges and *SELECT_CATALOG_ROLE*.

Supported Data Types

The following are the different data types that are tested in our lab setup:

- VARCHAR
- VARCHAR2
- NUMBER
- FLOAT
- DATE
- TIMESTAMP(default)
- CLOB
- BLOB(with text)
- User Defined Types:
    - Collection (Nested table only)
- Structured data types:
    - XML
    - JSON

- Hyperscale Compliance restricts the support of the following special characters for a user defined type name: `` ~!@#$%^&*()\\\"?:;,/\\\\`+=[]{}|<>'-.\")] `` and also restricts collection of CLOB and BLOB in user defined type.
- Hyperscale Compliance restricts the support of the following special characters for a database column name: `` ~!@#$%^&*()\\\"?:;,/\\\\`+=[]{}|<>'-.\")] ``

Property values

Property	Value
`SKIP.LOAD.SPLIT.COUNT.VALIDATION`	`false`
`SKIP.UNLOAD.SPLIT.COUNT.VALIDATION`	`false`

For default values, see https://delphixdocs.atlassian.net/wiki/spaces/~5ceab98e6f51c10fbd18ae6d/pages/230229335.

Known limitations

The length of the algorithm's generated masked data may exceed the target database table's column length resulting in a job failure if the target table columns use CHAR data type with BYTE length semantics to store the multibyte characters in the corresponding column. The workaround is to use an algorithm that should generate mask data with a smaller length.

MS SQL Connector

Supported versions

Microsoft SQL Server 2019

Supported data types

The following are the different data types that are tested in our lab setup:

- VARCHAR
- CHAR
- DATETIME
- INT
- TEXT
- VARBINARY (only unload/load)
- SMALLINT
- SMALLMONEY
- MONEY
- BIGINT
- NVARCHAR
- TINYINT
- NUMERIC(X,Y)
- DECIMAL(X,Y)
- FLOAT
- NCHAR
- BIT
- NTEXT
- MONEY
- Structured data types:
  - XML
  - JSON

Property Values

Property	Value
`SKIP.LOAD.SPLIT.COUNT.VALIDATION`	`false`
`SKIP.UNLOAD.SPLIT.COUNT.VALIDATION`	`false`

For default values, see https://delphixdocs.atlassian.net/wiki/spaces/~5ceab98e6f51c10fbd18ae6d/pages/230229335 .

Known Limitations

- If the applied algorithm's produced mask data exceeds the corresponding target table columns datatype's max value range, then job execution will fail in load service.
- Schemas, tables, and column names having special characters are not supported.
- Masking of columns with `VARBINARY` datatype is not supported.
- Hyperscale Compliance can mask up to a maximum 1000 tables in a single job.

File connector

In earlier versions of Hyperscale Compliance, Delimited and Parquet connectors were released as separate connectors. From the 23.0.0 version of Hyperscale Compliance onwards, Delimited and Parquet connectors are merged into a single file connector.
The connector can be used to mask large delimited and parquet files. The file connector’s unload service splits the large files into smaller chunks and passes them onto the masking service. After the masking is completed, the files are sent to the load service which joins back the split files (the end user also has a choice to disable the join operation).
Delphix supports Delimted and Parquet as file types for file connectors. The splitting and joining of files are handled by the writers embedded in the file connector. You can choose an appropriate writer for your operations by setting an environment variable `DATA_WRITER_TYPE` . Users can also skip the processing of source files in unload service by enabling the staging push feature, setting an environment variable `SKIP_UNLOAD_WRITERS` to true will enable the staging push at the unload service. Users can also facilitate staging push in load service by setting up the environment variable to `SKIP_LOAD_WRITERS` .
The supported data writers are:

1. pyarrow
2. pyspark
3. cat

You can refer to the below matrix when choosing the writer type.

Source Type	File Type	DATA_WRITER_TYPE (Unload)	DATA_WRITER_TYPE (Load)
FS	Delimited/Parquet	pyarrow(default)/pyspark	pyarrow/cat (default)
AWS	Delimited/Parquet	pyarrow(default)/pyspark	pyspark(default)

Source Type	**File Type**	**DATA_WRITER_TYPE (Unload)**	**DATA_WRITER_TYPE (Load)**
HADOOP-DB/HADOOP-FS	Delimited/Parquet	pyarrow*	pyarrow*
HADOOP-DB/HADOOP-FS	Delimited/Parquet	pyspark	pyspark

*If the source is Hadoop and the user is willing to use pyarrow as a writer, they must provide the Hadoop client as a volume mount.

> 1. If Parquet file type contains complex data types, then use pyspark for both services.
> 2. If Delimited file type has a target location in an S3 bucket, then use pyspark for both services.
> 3. For Hadoop, ensure that you use same writer for both the services.

Prerequisites

- The source and target (NFS) locations have to be mounted onto the docker containers of unload and load service. Please note that the locations on the containers are what needs to be used when creating the connector-info’s using the controller.

```
# As an example
unload-service:
     image: delphix-file-connector-unload-service-app:<HYPERSCALE VERSION>
     ...
     volumes:
          ...
          - /path/to/nfs/mounted/source1/files:/mnt/source1
          - /path/to/nfs/mounted/source2/files:/mnt/source2
...
load-service:
     image: delphix-file-connector-load-service-app:<HYPERSCALE VERSION>
     ...
     volumes:
          ...
          - /path/to/nfs/mounted/target1/files:/mnt/target1
          - /path/to/nfs/mounted/target2/files:/mnt/target2
```

- Set the required data writer using the DATA_WRITER_TYPE environment variable.

```
unload-service:
     image: delphix-file-connector-unload-service-app:<HYPERSCALE VERSION>
     ...
     volumes:
          ...
          - DATA_WRITER_TYPE=pyspark
...
load-service:
     image: delphix-file-connector-load-service-app:<HYPERSCALE VERSION>
```

```
    ...
    environment:
         ...
         - DATA_WRITER_TYPE=pyspark
```

- The connector should be able to access the AWS S3 buckets (the source and target locations). The following approaches are supported by the connector and can be used to authenticate with the S3 bucket:
    - Attaching the IAM role to the EC2 instance where the hyperscale masking services will be deployed.
        - IAM Roles are designed for applications to securely make AWS-API requests from EC2 instances, without the necessity to manage the security credentials that the applications use.
            - Using the AWS console UI or AWS CLI, attach the IAM role to the EC2 instance running the Hyperscale services. To know more, check the AWS Documentation.
            - With IAM role authentication, there is no need to pass the AWS credentials during the connector-info creation.

            ```
            # Example connector-info payload
            {
              "source": {
                "type": "AWS",
                "properties": {
                    "server": "S3",
                    "path": "aws_s3_bucket/sub_folder(s)"
                }
              },
              "target": {
                "type": "AWS",
                "properties": {
                    "server": "S3",
                    "path": "aws_s3_bucket/sub_folder(s)"
                }
              }
            }
            ```

    - Passing the AWS Access Key ID & AWS Secret Access Key attached to an AWS role:
        - Access keys are long-term credentials generated for an IAM user or role. These keys can be for programmatic requests to the AWS CLI or AWS API (directly or using the AWS SDK). To know more, check the AWS Documentation.
        - These credentials can be passed during the connector-info creation.

        ```
        # Example connector-info payload
        {
          "source": {
            "type": "AWS",
            "properties": {
                "server": "S3",
                "path": "aws_s3_bucket/sub_folder(s)",
                "aws_region": "us-west-2",
                "aws_access_key_id": "AWS_ACCESS_KEY_ID",
                "aws_secret_access_key": "AWS_SECRET_ACCESS_KEY"
            }
          },
        ```

```
  "target": {
    "type": "AWS",
    "properties": {
        "server": "S3",
        "path": "aws_s3_bucket/sub_folder(s)",
        "aws_region": "us-west-2",
        "aws_access_key_id": "AWS_ACCESS_KEY_ID",
        "aws_secret_access_key": "AWS_SECRET_ACCESS_KEY"
    }
  }
}
```

  - They can also be set as environment variables when bringing up the connector services.

```
unload-service:
    ...
    environment:
      - AWS_DEFAULT_REGION=us-east-1
      - AWS_ACCESS_KEY_ID=<aws_access_key_id>
      - AWS_SECRET_ACCESS_KEY=<aws_secret_access_key>

  ...
  load-service:
    ...
    environment:
      - AWS_DEFAULT_REGION=us-east-1
      - AWS_ACCESS_KEY_ID=<aws_access_key_id>
      - AWS_SECRET_ACCESS_KEY=<aws_secret_access_key>
```

- Set the required data writer using the DATA_WRITER_TYPE environment variable.

```
unload-service:
     image: delphix-file-connector-unload-service-app:<HYPERSCALE VERSION>
     ...
     volumes:
          ...
          - DATA_WRITER_TYPE=pyspark
...
load-service:
     image: delphix-file-connector-load-service-app:<HYPERSCALE VERSION>
     ...
     environment:
          ...
          - DATA_WRITER_TYPE=pyspark
```

- Set **SKIP_UNLOAD_WRITERS** and **SKIP_LOAD_WRITERS** environment variables to **true** to enable staging push.

```
unload-service:
     image: delphix-file-connector-unload-service-app:<HYPERSCALE VERSION>
```

```
    ...
    volumes:
        ...
        - SKIP_UNLOAD_WRITERS=true
...
load-service:
    image: delphix-file-connector-load-service-app:<HYPERSCALE VERSION>
    ...
    environment:
        ...
        - SKIP_LOAD_WRITERS=true
```

- In case the source is HADOOP-DB, the connector requires Kerberos authentication to access the Hadoop cluster. Users must provide a principal name and keytab file that grant access to both the Hive database and the underlying HDFS. In case source is HADOOP-FS, the Kerberos principal name is necessary to retrieve files from the specified HDFS path. In both cases, the target location is always an HDFS file system. Below are the pre-requisites for using Hadoop as a connector type:
    - The user has to volume mount the `core-site.xml`, `hadoop.keytab` and `krb5.conf` in unload and load services. Below is an example of the same.

```
# As an example
unload-service:
    image: delphix-file-connector-unload-service-app:<HYPERSCALE VERSION>
    ...
    volumes:
        ...
        - /path/to/hadoop/keytab/hdfs.keytab:/app/hadoop.keytab
        - /path/to/hadoop/krb5.conf:/etc/krb5.conf
        - /path/to/hadoop/core-site.xml:/app/hadoop/etc/hadoop/core-
site.xml
...
load-service:
    image: delphix-file-connector-load-service-app:<HYPERSCALE VERSION>
    ...
    volumes:
        ...
        - /path/to/hadoop/keytab/hdfs.keytab:/app/hadoop.keytab
        - /path/to/hadoop/krb5.conf:/etc/krb5.conf
        - /path/to/hadoop/core-site.xml:/app/hadoop/etc/hadoop/core-
site.xml
```

    - In case the user is using DATA_WRITER_TYPE as pyarrow then they also have to provide Hadoop client location as volume mount in addition to above volume mounts in unload and load services. Below is example of same.

```
- /path/to/hadoop-client/hadoop:/app/hadoop
```

- Below are a few connector info examples with hadoop as source and target:
    - The source and target are HADOOP-FS

```
{
  "connectorName": "Hadoop_Hdfs_connector",
  "source": {
    "type": "HADOOP-FS",
    "properties": {
      "server": "<Hadoop server hostname>",
      "port": "<HDFS port>",
      "path": "Source file location",
      "principal_name": "<Kerberos principal name>",
      "protocol": "hdfs"
    }
  },
  "target": {
    "type": "HADOOP-FS",
    "properties": {
      "server": "<Hadoop server hostname>",
      "path": "<target file location in HDFS>",
      "protocol": "hdfs",
      "port": "<HDFS port>",
      "principal_name": "<Kerberos principal name>"
    }
  }
}
```

- The source is HADOOP-DB and target is HADOOP-FS

```
{
  "connectorName": "Hadoop_Hive_connector",
  "source": {
    "type": "HADOOP-DB",
    "properties": {
      "server": "<Hadoop server hostname>",
      "database_name": "<database name where the tables are present>",
      "database_type": "hive",
      "database_port": "<Thrift SSL port for Apache hive>",
      "principal_name": "<Kerberos principal name>",
      "protocol": "hdfs"
    }
  },
  "target": {
    "type": "HADOOP-FS",
    "properties": {
      "server": "<Hadoop server hostname>",
      "path": "<target file location in HDFS>",
      "protocol": "hdfs",
      "port": "<HDFS port>",
      "principal_name": "<Kerberos principal name>"
    }
  }
}
```

Property values

Property	Value
SOURCE_KEY_FIELD_NAMES	unique_source_files_identifier
LOAD_SERVICE_REQUIREPOSTLOAD	false
DATA_WRITER_TYPE	1. "pyarrow" (Default for delimited-unload-service) 2. "pyspark" 3. "cat" (Default as well as only applicable to delimited-load-service)
UNLOAD_SPARK_DRIVER_MEMORY	90% of available memory
UNLOAD_SPARK_DRIVER_CORES	90% of available cores
MAX_WORKER_THREADS_PER_JOB	512
SKIP_UNLOAD_WRITERS	false
SKIP_LOAD_WRITERS	false

For default values, see Configuration settings.

Supported Compression types for Parquet files

- SNAPPY
- GZIP
- BROTLI
- ZSTD
- LZ4
- UNCOMPRESSED

Supported data types

The following are the supported data types for delimited files:

- String/Text
- Double
- Int64
- Timestamp

The following are the supported data types for parquet files :

- BOOLEAN
- INT32
- INT64
- INT96
- FLOAT

- DOUBLE
- BYTE_ARRAY

Known limitations

Delimited File types

- Supports only Single-character ASCII delimiters
- The end-of-record character can only be \n, \r, or \r\n.
- Limitations with PyArrow Data Writer:
    - Output files will exclusively enclose all string types with double quotes (`“`).

      Col umns with double data types will be converted to strings. For example, 63779742372828869945O5 will be converted to “363779742372828869945O5”.
    - Columns with int64 data type will be converted to strings. For example, 00094353043917225568O5 will be converted to “000094353043917225568O5".
- Limitation with PySpark Data Writer:
    - PySpark is more memory intensive, so in case we are processing data that is more in size in comparison to the available memory then we may run into issues related to resource exhaustion. Caution: The size of split files multiplied by the number of cores must not exceed the system memory.
    - With PyAarrow as the data writer, the split files are generated one after the other, so the masking-service is called as and when a split is created. With PySpark as the data writer, all split files are available only after the split process is complete. So the masking service will be only called after all splits are completed. Due to this, the overall time taken to complete the hyperscale masking execution will be more compared to the former.
    - There is a possibility that the number of splits created in the end will be less than the requested number, this generally happens when the file size is small, and spark doesn’t create as many partitions as the requested split number.

Parquet file types

- Generally, the parquet files are compressed and the compression factor could vary from 2x to 70x or even more. So, when working with such larger files the connector will need a host which has large enough memory to accommodate the parallel execution of multiple large parquet files. In case the sum of the uncompressed size of parquet files that are getting executed in parallel exceeds 80% of RAM size then the chances of having an “out of memory” error are high. To avoid OOM, the end user can reduce the MAX_WORKER_THREADS_PER_JOB (i.e. reduce the number of parallel threads), ultimately reducing the memory usage.
- Struct and list type values are treated as strings for the Delphix Continuous Compliance Engine, therefore, you can not add individual elements of any struct and list to the masking inventory property of the dataset payload.

MongoDB connector

The connector can be used to mask large MongoDB files. The Mongo unload service splits the large collections into smaller chunks and passes them onto the masking service. After the masking is completed, the files are sent to the Mongo load service, which imports the masked files into the target collection.

Supported versions

Platforms	**Version**
Linux	MongoDB 4.4.x MongoDB 5.0.x MongoDB 6.0.x MongoDB 7.0.x MongoDB 8.0.x

Roles and privileges

MongoDB users should have the following roles and privileges:

Topology of Database	**Source Database User Privileges**		**Target Database User Privileges**	
	Default	**drop_collection : No**	**Default**	**drop_collection : No**
Sharded Replica Set	role: clusterMonitor db: admin	role: read db: <source database>	role: clusterAdmin db: admin”	role: readWrite db: target database
	role: read db: <source database>		role: readWrite db: <target database>	
Non Sharded Replica Set	role: clusterMonitor db: admin	role: read db: <source database>	role: clusterMonitor db: admin	role: readWrite db: <target database>
	role: read, db: <source database>		role: readWrite db: <target database>	

Prerequisites

1. Mongo Unload and Mongo Load service image names are to be used under unload-service and load-service. The NFS location has to be mounted onto the Docker containers for unload and load services. Example for mounting `/mnt/hyperscale`.

```
# As an example docker-compose.yaml
```

```
unload-service:
     image: delphix-mongo-unload-service-app:${VERSION}
volumes:
   # Uncomment below lines to mount respective paths.
   - /mnt/hyperscale:/etc/hyperscale

load-service:
     image: delphix-mongo-load-service-app:${VERSION}
volumes:
   # Uncomment below lines to mount respective paths.
   - /mnt/hyperscale:/etc/hyperscale
```

2. Uncomment the below lines from `docker-compose.yaml` file under `controller > environment`:

```
# uncomment below for MongoDB connector
#- SOURCE_KEY_FIELD_NAMES=database_name,collection_name
#- VALIDATE_UNLOAD_ROW_COUNT_FOR_STATUS=${VALIDATE_UNLOAD_ROW_COUNT_FOR_STATUS:-false
}
#- VALIDATE_MASKED_ROW_COUNT_FOR_STATUS=${VALIDATE_MASKED_ROW_COUNT_FOR_STATUS:-false
}
#- VALIDATE_LOAD_ROW_COUNT_FOR_STATUS=${VALIDATE_LOAD_ROW_COUNT_FOR_STATUS:-false}
#- DISPLAY_BYTES_INFO_IN_STATUS=${DISPLAY_BYTES_INFO_IN_STATUS:-true}
#- DISPLAY_ROW_COUNT_IN_STATUS=${DISPLAY_ROW_COUNT_IN_STATUS:-false}
```

3. Set the value of `LOAD_SERVICE_REQUIRE_POST_LOAD=false` inside the “ `.env` ” file.

```
# Set LOAD_SERVICE_REQUIRE_POST_LOAD=false for MongoDB Connector
LOAD_SERVICE_REQUIRE_POST_LOAD=false
```

4. Uncomment the below lines from “ `.env` ” file.

```
# Uncomment below for MongoDB Connector
#VALIDATE_UNLOAD_ROW_COUNT_FOR_STATUS=false
#VALIDATE_MASKED_ROW_COUNT_FOR_STATUS=false
#VALIDATE_LOAD_ROW_COUNT_FOR_STATUS=false
#DISPLAY_BYTES_INFO_IN_STATUS=true
#DISPLAY_ROW_COUNT_IN_STATUS=false
```

5. To leverage **Reduced Privilege Operations**, you must set the `drop_collection` property to No. After this property is set, the connector will no longer require `clusterAdmin` and `clusterMonitor` privileges. The following are the implications of setting `drop_collection` property to No:
    - The connector will not validate `clusterMonitor` privilege at the source and `clusterMonitor` and `clusterAdmin` privileges at the target. For more information, refer to the *Roles and Privileges* table above.
    - The connector will **skip** the following operations on a target collection:
        i. shard collection

ii. create shard key
iii. create index

You will be responsible for executing the above operations on a target collection.

Property values

Mandatory changes are required for the MongoDB Connector in the `docker-compose.yaml` and `.env` files:

Property	Value
SOURCE_KEY_FIELD_NAMES	database_name,collection_name
LOAD_SERVICE_REQUIRE_POST_LOAD	false
VALIDATE_UNLOAD_ROW_COUNT_FOR_STATUS	false
VALIDATE_MASKED_ROW_COUNT_FOR_STATUS	false
VALIDATE_LOAD_ROW_COUNT_FOR_STATUS	false
DISPLAY_BYTES_INFO_IN_STATUS	true
DISPLAY_ROW_COUNT_IN_STATUS	false

For default values, see Configuration settings.

Known limitation:

- In-Place Masking is not supported.

Snowflake connector

The connector can be used to mask table residing under Snowflake platform. The Snowflake unload service splits data exported from source tables into smaller chunks (CSV files) and passes them onto the masking service. After the masking is completed, the files are sent to the Snowflake load service, which imports the masked files into the target database table.

Supported versions

The Snowflake platform is version-agnostic. We do not need to manage or specify snowflake database versions, as updates and maintenance are handled automatically by Snowflake.

Prerequisites

1. For accessing Snowflake warehouse, you will need a service account user with key-pair authentication. (Key-pair authentication provides enhanced security).

a. To configure key-pair authentication for a Snowflake service account, refer to the instructions provided in Snowflake's configuring key-pair authentication documentation.

b. Use the following commands to generate base64 encoded value of encrypted private key and passphrase.

To generate base64 encoded value of encrypted private key:

```
echo -n `cat /home/delphix/keys/MKK_TEST_SFHSC_ADMIN_key.p8|base64`
```

To generate base64 encode value of passphrase:

```
echo -n <passphrase>|base64 -w 0
```

2. Configure Snowflake storage integration to read data from and write data to an Amazon S3 bucket referenced in an external (i.e. S3) stage

   a. To configure the Snowflake storage integration, refer to the instructions provided in Snowflake's configuring a Snowflake storage integration to access Amazon S3 documentation.

3. Configure Snowflake external stage (i.e. S3) where the data files are staged.

   a. To configure the Snowflake storage integration, refer to the instructions provided in Snowflake's creating an S3 stage documentation.

Roles and privileges

- The table below lists all the privileges required to unload and load data from/to Snowflake platform.
- The sample names (e.g., DLPX_WAREHOUSE, DLPX_SNOW_DB, DLPX_SNOWHSC, etc.) are for example and explanation purposes only.
- Replace these names with your actual warehouse, database, schema, and role names as per your Snowflake environment.

Source Snowflake database

Object type	Object name	Privilege	Granted to role
Warehouse	DLPX_WAREHOUSE	USAGE, OPERATE	DLPX_SNOWHSC
Database	DLPX_SNOW_DB	USAGE	DLPX_SNOWHSC
Schema	DLPX_SNOW_SCHEMA	USAGE	DLPX_SNOWHSC
All Tables	DLPX_SNOW_DB.DLPX_SNOW_SCHEMA	SELECT	DLPX_SNOWHSC
External Stage	DLPX_SNOW_DB.DLPX_SNOW_SCHEMA.DLPX_S3_EXT_STAGE	USAGE	DLPX_SNOWHSC

Target Snowflake database

Object type	Object name	Privilege	Granted to role
Warehouse	DLPX_WAREHOUSE	USAGE, OPERATE	DLPX_SNOWHSC
Database	DLPX_SNOW_DB	USAGE	DLPX_SNOWHSC
Schema	DLPX_SNOW_SCHEMA	USAGE, MODIFY, CREATE TABLE	DLPX_SNOWHSC
All Tables	DLPX_SNOW_DB.DLPX_SNOW_SCHEMA	SELECT, INSERT, UPDATE, DELETE	DLPX_SNOWHSC
External Stage	DLPX_SNOW_DB.DLPX_SNOW_SCHEMA.DLPX_S3_EXT_STAGE	USAGE	DLPX_SNOWHSC

Warehouse privileges: USAGE allows the role to use the warehouse and OPERATE enables suspending and resuming it.
Database privileges: USAGE allows the role to see the database and use objects within.
Schema privileges: Additional permissions such as CREATE TABLE and MODIFY are granted for schema-level management.
Table privileges: SELECT, INSERT, UPDATE, DELETE permissions for all existing tables ensure role-based access for managing data in the schema.
Stage privileges: USAGE is necessary for unloading data to and loading data from the external stage.

Property values

Snowflake Hyperscale Connector currently supports only AWS S3 as staging area.

Mandatory changes are required for the Snowflake Connector in the values.yaml file:

Property	Value
stagingStorageType	AWS_S3
applicationName	<staging-area-name>
authMechanism	AWS_ROLE

Property	Value
awsBucketName	<aws-s3-bucket-name>
awsBucketRegion	<s3-region-name>
awsBucketDelimiter	/

For instructions on how to configure AWS S3 as staging area, see configuring-aws-s3-bucket-as-staging-area

Mandatory changes are required for the Snowflake connector in the values-snowflake.yaml file:

Property	Value
sourceKeyFieldNames	database_name,schema_name,table_name,stage_name
loadServiceRequirepostload	false
validateUnloadRowCountForStatus	false
validateMaskedRowCountForStatus	false
validateLoadRowCountForStatus	false
displayBytesInfoInStatus	true
displayRowCountInStatus	true
snowflakePrivateKey	<SNOWFLAKE_PRIVATE_KEY>
snowflakePassphrase	<SNOWFLAKE_PASSPHRASE>
unload: imageName	delphix-snowflake-unload-service-app
load: imageName:	delphix-snowflake-load-service-app

For default values, see Configuration settings.

Known limitation

In-Place Masking is not supported.

Oracle - Automatic Calculation of Unload Split

With auto-calculation, by decoupling extraction from writing into files, it is possible to choose an optimal parallel degree for unloading but still produce smaller files.

Before the release of the auto-calculation feature and in the previous u*nload_split in data set* approach, having unload splits beyond a specific number was counterproductive as the unloaded files were larger and masking was time-consuming.

Therefore, by reducing the number of connections used in reading, you can have more jobs running in parallel or more tables unloaded in parallel within the same job. This also contributes to building the masking pipeline and parallelizing the masking of files in a distributed manner.

With the auto-calculation feature, the need to manually provide the number of splits is eliminated. This not only saves time but also makes the product more user-friendly and scalable.

Parameter description

The following properties are available at the unload service level so that they can be set at the time of deployment. Additionally, you will also have an option to set these values at job level.

Parameter	**Description**
max_records_per_split	This parameter defines the maximum records unloaded in an extracted file. A default of 100,000,000 defines that each extracted file will be 100,000,000 records or less.
max_split_per_connection	This param indicates how many splits each database connection can take. The value 4 indicates that connection can result into 4 extracted files.
max_parallel_connections_per_table	This param defines how many maximum parallel connections are allowed for a table during the unload process. The value 4 indicates that four parallel extraction connection can be executed against the table, with each process theoretically extracting 25% of table data.
auto_calculate_unload_split: false	If false, the unload service will unload data based legacy unload_split (unload_split in data set) approach. If true, the unload service will unload data based 'Automatic Calculation of Unload Split ' approach by using max_records_per_split , max_parallel_connections_per_table, max_split_per_connection.

Process Description

Depending on the size of the table (number of rows fetched from all_tables), parallel unload connections will be computed based on the parameters max_split_per_connection and max_parallel_connections_per_table. Each unload connection will be executing a select sql based on modulo of rowid and writes every 'n' records into a file. 'n' is defined by the parameter max_records_per_split (splitSize).

In the following three examples, we consider the default values of all the parameters.

Example 1: Table size 10000000 (100 Million)

Legacy (unload_split in data set) approach
If unload_split (in data set) is defined as 1, the table will be extracted into a single file.

Automatic Calculation of Unload Split approach
The table will be extracted with a single unload process (1 select SQL query) into a single file.

(Effectively no change)

Example 2: Table size 20000000 (200 Million)

Legacy (unload_split in data set) approach
If unload_split (in data set) is defined as 2, the table will be extracted into 2 files by running 2 unload processes (2 select SQL queries).

Automatic Calculation of Unload Split approach
Calculation:
Table size: 20000000

max_records_per_split: 100000000

Number of splits needed = 20000000 /10000000 = 2

max_split_per_connection: 4

# connections needed = 2/4 = 0.5 , use next positive integer = 1

The table is extracted with a single unload connection (1 select SQL query) spilt into two files.

Example 3: Table size 500000000 (5 Billion)

Legacy (unload_split in data set) approach
If unload_split (in data set) is defined as 10, the table will be extracted into 25 files (each file 200 Mi) by running 25 unload connection(25 select SQL queries)

Automatic calculation of unload split approach

Calculation:

Table size: 500000000

max_records_per_split: 100000000

Number of splits needed = 500000000 /10000000 = 50

max_split_per_connection: 4

# connections needed = 50/4 = 12.5 , use next positive integer = 13

However, max_parallel_connections_per_table: 10 therefore, only 10 connections will be opened for the table.

The table will be extracted with 10 unload connections in parallel (10 select SQL queries), with each unload extracting approximately 500 Mi rows and writing into 5 files (every 100 Mi records a new file is created).
NOTE: Though max_split_per_connection =4, yet system will unload data into 5 files to respect the value of max_parallel_connections_per_table.

Supported platforms

Delphix supports Hyperscale Compliance for many data platforms and operating systems.

Supported Continuous Compliance/Data versions

Delphix Engine	Minimum supported version	Recommended version
Continuous Compliance	6.0.14.0 15.0.0.0 (for embedded XML/JSON masking)	Latest
Continuous Data	6.0.14.0	Latest

All Continuous Compliance Engines must be of the same versions and must be used only by Hyperscale Compliance for masking. Already existing or running masking/profiling jobs on Continuous Compliance engines would impact Hyperscale Compliance performance and results.

Supported browsers (only API client)

Hyperscale Compliance API Client is using Swagger UI-3.48.0 which works in the latest versions of Chrome, Safari, Firefox, and Edge. For more information about the supported browser versions, see the **Browser Support** section on GitHub.

If you encounter `Chrome NET::ERR_CERT_INVALID` error code, perform the following steps to resolve the above error:

- Type `https://<hyperscale-compliance-host address>/hyperscale-compliance` in the address bar and click **Enter**.
- Right-click on the page and click **Inspect**.
- Click the **Console** tab and run the following command: `sendCommand(SecurityInterstitialCommandId.CMD_PROCEED)` .
- Click on **Authorize** and provide the key. For more information about the key, refer to step 7 in Generate a New Key.

Network requirements

This section describes the network requirements for Hyperscale Compliance. Ensure that you meet all the network requirements before you install the Hyperscale Compliance Orchestrator.

The following are the inbound/outbound rules for the Hyperscale Compliance Orchestrator:

Type (Inbound/Outbound)	Port	Reason
Inbound and Outbound	80	HTTP connections to/from the Hyperscale Compliance Orchestrator to/from the Continuous Compliance Engines part of the Continuous Compliance Engine Cluster and to access the Hyperscale Compliance API.
Inbound and Outbound	443	HTTPs connections to/from the Hyperscale Compliance Orchestrator to/from the Continuous Compliance Engines part of the Continuous Compliance Engine Cluster and to access the Hyperscale Compliance API.
Outbound	53	Connections to local DNS servers.
Inbound	22	SSH connections to the Hyperscale Compliance Orchestrator host.

Deployment

This section covers the following topics:

- Docker compose
- Kubernetes
- OpenShift
- Podman Compose

Docker compose

⚠ Docker Compose deployments for Hyperscale Compliance will be deprecated from January 2026. All versions will continue to be supported until January 2026; however, Kubernetes is now the primary deployment tool for Hyperscale Compliance. You can find more information on this Delphix Community page.
Note: the original deprecation date was changed from January 2025 and postponed until January 2026. This extension will provide you with additional time to transition to an alternative deployment method.

This section covers the following topics:

- Host requirements (Docker compose)
- HTTPS certificate for Hyperscale (Docker Compose)
- Installation and setup (Docker compose)
- Custom configuration
- Upgrading the Hyperscale Compliance Orchestrator (Docker Compose)
- Managing the storage space
- Migrating to Kubernetes

Host requirements (Docker compose)

Type	**Host Requirement**	**Explanation**
User	A user (hyperscale_os) with the following permissions are required: • Should have permissions to install `docker` and `docker-compose`. • Should be part of the 'docker' OS group or must have the permission to run `docker` and `docker-compose` commands. • Permission to run mount, unmount, mkdir and rmdir as a super-user with NOPASSWD. • Should have either GID=50 and/or UID=65436.	This will be a primary user responsible to install and operate the Hyperscale Compliance.
Installation Directory	There must be a directory on the Hyperscale Compliance Orchestrator host where the Hyperscale Compliance can be installed.	This is a directory where the Hyperscale Compliance tar archive file will be placed and extracted. The extracted artifacts will include docker images(tar archive files) and a configuration file(docker-compose.yaml) that will be used to install the Hyperscale Compliance.
Log File Directory	An optional directory to place log files.	This directory (can be configured via docker-compose.yaml configuration file) will host the runtime/log files of the Hyperscale Compliance Orchestrator.
NFS Client Services	NFS client services must be enabled on the host.	NFS client service is required to be able to mount an NFS shared storage from where the Hyperscale Compliance Orchestrator will be able to read the source files and write the target files. For more information, see NFS Server Installation.

Type	Host Requirement	Explanation
Hardware Requirements	• Minimum: 8 vCPU, 64 GB of memory, 100GB data disk. • Recommended: 16 vCPU, 128GB of memory, 500GB data disk.	OS disk space: 50 GB

HTTPS certificate for Hyperscale (Docker Compose)

By default, Hyperscale creates a unique self-signed certificate to enable HTTPS when starting for the first time. To use your certificates, these default values need to be replaced.

> You may need to contact the IT team to get the corresponding certificate files.

For example,

- Place your files on the Hyperscale host at the following location:

```
<LOCAL_PATH>/nginx-selfsigned.key
<LOCAL_PATH>/nginx-selfsigned.crt
```

- Then, update volumes under **proxy** service as below in `docker-compose.yaml` :

```
volumes:
<LOCAL_PATH>/nginx-selfsigned.key:/etc/config/nginx/ssl/nginx.key
<LOCAL_PATH>/nginx-selfsigned.crt:/etc/config/nginx/ssl/nginx.crt
```

- After this is completed, restart Hyperscale.

Installation and setup (Docker compose)

⚠ Docker Compose deployments for Hyperscale Compliance will be deprecated from January 2026. All versions will continue to be supported until January 2026; however, Kubernetes is now the primary deployment tool for Hyperscale Compliance. You can find more information on this Delphix Community page.
Note: the original deprecation date was changed from January 2025 and postponed until January 2026. This extension will provide you with additional time to transition to an alternative deployment method.

This section describes the steps you must perform to install the Hyperscale Compliance Orchestrator.

Hyperscale Compliance installation

Pre-requisites

Ensure the following requirements are met before installing the Hyperscale Compliance Orchestrator.

- Download the Hyperscale tar file (`delphix-hyperscale-masking-x.0.0.tar.gz`) from download.delphix.com, where `x.0.0` should be changed to the version of Hyperscale being installed.
- Create a user that has permission to install Docker and Docker Compose.
- Install Docker on the VM. The minimum supported Docker version is 20.10.7.
- Install Docker Compose on the VM. The minimum supported Docker Compose version is 1.29.2.
- Check if Docker and Docker Compose are installed by running the following command:
    - `docker-compose -v`
        - The above command displays an output similar to the following: `docker-compose version 1.29.2, build 5becea4c`
    - `docker -v`
        - The above command displays an output similar to the following: `Docker version 20.10.7, build 3967b7d`
- **(Only Required for Oracle Load Service)** Download and install Linux-based Oracle's instant client on the machine where the Hyperscale Compliance Orchestrator will be installed. The client should essentially include `instantclient-basic` (Oracle shared libraries) along with `instantclient-tools` containing `Oracle's SQL*Loader` client. Both the packages instantclient-basic and instantclient-tools should be unzipped in the same directory. A group ownership id of 50 with a permission mode of 550 or a user id of 65436 with a permission mode of 500 must be set recursively on the directory where Oracle's instant client `binaries/libraries` will be installed. This is required by the Hyperscale Compliance Orchestrator to be able to read or execute from the directory.

ℹ Oracle Load does not support Object Identifiers(OIDs).

Procedure

Perform the following procedure to install the Hyperscale Compliance Orchestrator.

1. Unpack the Hyperscale tar file (where `x.0.0` should be changed to the version of Hyperscale being installed).

`tar -xzf delphix-hyperscale-masking-x.0.0.tar.gz`

2. Upon unpacking, you will find the Docker image tar files categorized as below:

- **Universal images common for all connectors.**
  - `controller-service.tar`
  - `masking-service.tar`
  - `proxy.tar`
- **Oracle (required only for Oracle data source masking)**
  - `unload-service.tar`
  - `load-service.tar`
- **MSSQL (required only for MS SQL data source masking)**
  - `mssql-unload-service.tar`
  - `mssql-load-service.tar`
- **File Connector (required only for Delimited and Parquet File masking)**
  - `file-connector-unload-service.tar`
  - `file-connector-load-service.tar`
- **MongoDB (required only for MongoDB database masking)**
  - `mongo-unload-service.tar`
  - `mongo-load-service.tar`
- **Snowflake (required only for Snowflake database masking)**
  - `snowflake-unload-service.tar`
  - `snowflake-load-service.tar`

Each deployment set consists of 5 images (3 Universal images and 2 images related to each dataset type). Proceed to load the required images into Docker as below:

- For Oracle data source masking:

```
docker load --input unload-service.tar
docker load --input load-service.tar
docker load --input controller-service.tar
docker load --input masking-service.tar
docker load --input proxy.tar
```

- For MS SQL data source masking:

```
docker load --input mssql-unload-service.tar
docker load --input mssql-load-service.tar
docker load --input controller-service.tar
docker load --input masking-service.tar
docker load --input proxy.tar
```

- For Delimited and Parquet Files masking using file connector:

```
docker load --input file-connector-unload-service.tar
docker load --input file-connector-load-service.tar
docker load --input controller-service.tar
docker load --input masking-service.tar
docker load --input proxy.tar
```

- For MongoDB data source masking:

```
docker load --input mongo-unload-service.tar
docker load --input mongo-load-service.tar
docker load --input controller-service.tar
docker load --input masking-service.tar
docker load --input proxy.tar
```

- For Snowflake data source masking:

```
docker load --input snowflake-unload-service.tar
docker load --input snowflake-load-service.tar
docker load --input controller-service.tar
docker load --input masking-service.tar
docker load --input proxy.tar
```

3. Create an NFS shared mount that will act as a **Staging Area** on the Hyperscale Compliance Orchestrator host where the Hyperscale Compliance Orchestrator will perform read/write/execute operations.
    a. Create a 'Staging Area' directory. For example: `/mnt/provision`. Note you do not need to create a `mount-name` directory as required in previous releases. The shared path from the NFS server is the same as the value defined for config `NFS_STORAGE_EXPORT_PATH`. The user(s) within each of the docker containers part of the Hyperscale Compliance Orchestrator and the appliance OS user(s) in the Continuous Compliance Engine(s), all have User ID 65436 and the group ownership ID is 50. Therefore, the directory `provision`, requires the following permissions, based on the UID/GID of the OS user, so that the Hyperscale Compliance Orchestrator and the Continuous Compliance Engine(s) can perform read/write/execute operations on the staging area:
        i. If the Hyperscale Compliance OS user has a UID of 65436, then the directory `provision`, must have a UID of 65436 and 700 permission mode.
        ii. If the Hyperscale Compliance OS user has a GID of 50 and does not have a UID of 65436, then the directory `provision`, must have a GID of 50 and 770 permission mode.
    b. Mount the NFS shared directory on the staging area directory(`/mnt/provision`). This NFS shared storage can be created and mounted in two ways as detailed in the NFS Server Installation section. Based on the umask value for the user which is used to mount, the permissions for the staging area directory could get altered after the NFS share has been mounted. In such cases, the permissions(i.e. 770 or 700 whichever applies based on point 3a) must be applied again on the staging area directory.
4. After unpacking the tar, you will find the following sample docker-compose files available: `docker-compose-sample.yaml`, `docker-compose-oracle-sample.yaml`, `docker-compose-mssql-sample.yaml`, `docker-compose-file-connector-sample.yaml`, `docker-compose-mongo-sample.yaml` and `docker-compose-snowflake-sample.yaml.` These sample files can be used to create a `docker-compose.yaml` file based on the connector you want to

use. Configure the following docker container volume bindings for the docker containers by editing the `docker-compose.yaml`:

a. For each of the docker containers, except the ‘proxy’ container, add a volume entry binding the staging area path (from 3(a), `/mnt/hyperscale`) to the Hyperscale Compliance Orchestrator container path(`/etc/hyperscale`) as a volume binding under the ‘volumes’ section.

b. **(Only Required for Oracle Load Service)** For the **load-service** docker container, add a volume entry that binds the path of the directory on the host where both the Oracle instant Client packages were unzipped to the path on the container (`/usr/lib/instantclient`) under the ‘volumes’ section.

c. **(Only Required for File Connector Unload Service)** For File connector unload-service, the source NFS location has to be mounted to the container as docker volume in order for it to access the source files. The path mounted on the container is passed during the creation of the source connector-info.

```
# Mount your source NFS location onto your Hyperscale Engine server
sudo mount [-t nfs4] <source_nfs_endpoint>:<source_nfs_location>
<nfs_mount_on_host>
```

```
# Later mount <nfs_mount_on_host> as a docker volume to the file-connector
unload-service container (in docker-compose.yaml, created using docker-
compose-delimitedfiles-sample.yaml)
unload-service:
  image: delphix-file-connector-unload-service-app:<HYPERSCALE VERSION>
     ...
     volumes:
          ...
     # Source files should be made available within the unload-service
container file system
     # The paths within the container should be configured in the source
section of connector-info [with type=FS]
     -
<nfs_mount_on_host>:<source_files_mount_passed_during_connector_info_creat
ion_path_in_contianer>
```

d. **(Only Required for File Connector load Service)** For the File Connector load-service, the target NFS location has to be mounted to the container as docker volume in order for it to access the target location where masked files will be placed. The path mounted on the container is passed during the creation of the target connector-info.

```
# Mount your target NFS location onto your Hyperscale Engine server
sudo mount [-t nfs4] <target_nfs_endpoint>:<target_nfs_location>
<target_nfs_mount_on_host>
```

```
# Later mount <nfs_mount_on_host> as a docker volume to the file-connector
load-service container (in docker-compose.yaml, created using docker-
compose-delimitedfiles-sample.yaml)
load-service:
  image: delphix-file-connector-load-service-app:${VERSION}
     ...
     volumes:
          ...
     # Target location should be made available within the load-service
container file system
     # The paths within the container should be configured in the target
section of connector-info [with type=FS]
     -
<target_nfs_mount_on_host>:<target_location_passed_during_connector_info_c
reation_in_container>
```

e. **(Only Required for File Connector load Service)** If using the staging push feature, you will need to provide source files on the NFS mount point which is mapped to the staging area. Alternatively, you can also provide format files with the source files. If providing format files, set the environment variable `USER_PROVIDED_FORMAT_FILE` to true, and by default, pyarrow writer will create the format file. In the below example `/source/files/path` and staging area must share the same mount point.

```
unload-service:
  image: delphix-file-connector-unload-service-app:${VERSION}
     ...
     volumes:
          ...
     # Source location should be made available within the unload-service
container file system
     # The paths within the container should be configured in the source
section of connector-info [with type=FS]
     - /source/files/path:/etc/hyperscale/source

load-service:
  image: delphix-file-connector-load-service-app:${VERSION}
     ...
     volumes:
          ...
     # Target location should be made available within the load-service
container file system
     # The paths within the container should be configured in the target
section of connector-info [with type=FS]
     - /target/files/path:/etc/hyperscale/target
```

f. **(Optional)** Some data (for example, logs, configuration files, etc.) that is generated inside the docker containers may be useful to debug possible errors or exceptions while running the hyperscale jobs, and as such it may be beneficial to persist these logs outside docker containers. The following data can be persisted outside the docker containers:
   i. The logs generated for each service i.e. unload, controller, masking, and load services.
   ii. The sqlldr utility logs and control files at opt/sqlldr location in the load-service container.
   iii. The file-upload folder at `/etc/hyperscale/uploads` in the controller-service container

If you would like to persist the above data on your host, then you have the option to do the same by setting up volume bindings in the respective service as indicated below, that map locations inside the docker containers to locations on the host in the `docker-compose.yaml` file. The host locations again must have a group ownership id of 50 with a permission mode of 770 or a user id of 65436 with a permission of 700, due to the same reasons as highlighted in step 3a.

Here are examples of the docker-compose.yaml file for Oracle, MS SQL, MongoDB data sources, Delimited file masking, and Parquet file masking:

- For Oracle data source masking:

```
version: "3.7"
services:
  controller-service:
    image: delphix-controller-service-app:${VERSION}
    healthcheck:
      test: 'curl --fail --silent http://localhost:8080/actuator/health | grep UP || exit 1'
      interval: 30s
      timeout: 25s
      retries: 3
      start_period: 30s
    depends_on:
      - unload-service
      - masking-service
      - load-service
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-controller-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples. Configure the respective mount paths.
      - /home/hyperscale_user/logs/controller_service:/opt/delphix/logs
      - /mnt/hyperscale:/etc/hyperscale
    environment:
      - API_KEY_CREATE=${API_KEY_CREATE:-false}
      - EXECUTION_STATUS_POLL_DURATION=${EXECUTION_STATUS_POLL_DURATION:-12000}
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_CONTROLLER_SERVICE:-INFO}
      - API_VERSION_COMPATIBILITY_STRICT_CHECK=${API_VERSION_COMPATIBILITY_STRICT_CHECK:-false}
      - LOAD_SERVICE_REQUIREPOSTLOAD=${LOAD_SERVICE_REQUIRE_POST_LOAD:-true}
      - SKIP_UNLOAD_SPLIT_COUNT_VALIDATION=${SKIP_UNLOAD_SPLIT_COUNT_VALIDATION:-false}
      - SKIP_LOAD_SPLIT_COUNT_VALIDATION=${SKIP_LOAD_SPLIT_COUNT_VALIDATION:-false}
      - CANCEL_STATUS_POLL_DURATION=${CANCEL_STATUS_POLL_DURATION:-60000}
      - NFS_STORAGE_HOST=${NFS_STORAGE_HOST:-}
      - NFS_STORAGE_EXPORT_PATH=${NFS_STORAGE_EXPORT_PATH:-}
      - NFS_STORAGE_MOUNT_TYPE=${NFS_STORAGE_MOUNT_TYPE:-}
      - NFS_STORAGE_MOUNT_OPTION=${NFS_STORAGE_MOUNT_OPTION:-}
      - APPLICATION_NAME=${APPLICATION_NAME:-hs-staging-area}
    unload-service:
    image: delphix-unload-service-app:${VERSION}
    init: true
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_UNLOAD_SERVICE:-INFO}
      - UNLOAD_FETCH_ROWS=${UNLOAD_FETCH_ROWS:-10000}
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-unload-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/unload_service:/opt/delphix/logs
  masking-service:
    image: delphix-masking-service-app:${VERSION}
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-masking-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/masking_service:/opt/delphix/logs
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_MASKING_SERVICE:-INFO}
      - INTELLIGENT_LOADBALANCE_ENABLED=${INTELLIGENT_LOADBALANCE_ENABLED:-true
}
  load-service:
    image: delphix-load-service-app:${VERSION}
    init: true
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_LOAD_SERVICE:-INFO}
      - SQLLDR_BLOB_CLOB_CHAR_LENGTH=${SQLLDR_BLOB_CLOB_CHAR_LENGTH:-20000}
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-load-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /mnt/hyperscale:/etc/hyperscale
      - /opt/oracle/instantclient_21_5:/usr/lib/instantclient
      - /home/hyperscale_user/logs/load_service:/opt/delphix/logs
      - /home/hyperscale_user/logs/load_service/sqlldr:/opt/sqlldr/
  proxy:
    image: delphix-hyperscale-masking-proxy:${VERSION}
        init: true
        networks:
          - hyperscale-net
        ports:
          - "443:8443"
          - "80:8080"
    restart: unless-stopped
    depends_on:
      - controller-service
      #volumes:
      # Uncomment to bind mount /etc/config
      #- /nginx/config/path/on/host:/etc/config
networks:
  hyperscale-net:
volumes:
  hyperscale-load-data:
  hyperscale-unload-data:
  hyperscale-masking-data:
  hyperscale-controller-data:
```

- For MS SQL data source masking:

```
version: "3.7"
services:
  controller-service:
    image: delphix-controller-service-app:${VERSION}
    healthcheck:
      test: 'curl --fail --silent http://localhost:8080/actuator/health | grep
UP || exit 1'
      interval: 30s
      timeout: 25s
      retries: 3
      start_period: 30s
    depends_on:
      - unload-service
      - masking-service
      - load-service
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-controller-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /home/hyperscale_user/logs/controller_service:/opt/delphix/logs
      - /mnt/hyperscale:/etc/hyperscale
    environment:
      - API_KEY_CREATE=${API_KEY_CREATE:-false}
      - EXECUTION_STATUS_POLL_DURATION=${EXECUTION_STATUS_POLL_DURATION:-12000}
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_CONTROLLER_SERVICE:-
INFO}
      - API_VERSION_COMPATIBILITY_STRICT_CHECK=$
{API_VERSION_COMPATIBILITY_STRICT_CHECK:-false}
```

```
      - LOAD_SERVICE_REQUIREPOSTLOAD=${LOAD_SERVICE_REQUIRE_POST_LOAD:-true}
      - SKIP_UNLOAD_SPLIT_COUNT_VALIDATION=$
{SKIP_UNLOAD_SPLIT_COUNT_VALIDATION:-false}
      - SKIP_LOAD_SPLIT_COUNT_VALIDATION=$
{SKIP_LOAD_SPLIT_COUNT_VALIDATION:-false}
      - CANCEL_STATUS_POLL_DURATION=${CANCEL_STATUS_POLL_DURATION:-60000}
      - NFS_STORAGE_HOST=${NFS_STORAGE_HOST:-}
      - NFS_STORAGE_EXPORT_PATH=${NFS_STORAGE_EXPORT_PATH:-}
      - NFS_STORAGE_MOUNT_TYPE=${NFS_STORAGE_MOUNT_TYPE:-}
      - NFS_STORAGE_MOUNT_OPTION=${NFS_STORAGE_MOUNT_OPTION:-}
      - APPLICATION_NAME=${APPLICATION_NAME:-hs-staging-area}
  unload-service:
    image: delphix-mssql-unload-service-app:${VERSION}
    init: true
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_UNLOAD_SERVICE:-INFO}
      - UNLOAD_FETCH_ROWS=${UNLOAD_FETCH_ROWS:-10000}
      - SPARK_DATE_TIMESTAMP_FORMAT=${DATE_TIMESTAMP_FORMAT:-yyyy-MM-dd
HH:mm:ss.SSSS}
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-unload-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/unload_service:/opt/delphix/logs
  masking-service:
    image: delphix-masking-service-app:${VERSION}
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-masking-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/masking_service:/opt/delphix/logs
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_MASKING_SERVICE:-INFO}
      - INTELLIGENT_LOADBALANCE_ENABLED=${INTELLIGENT_LOADBALANCE_ENABLED:-true
}
  load-service:
    image: delphix-mssql-load-service-app:${VERSION}
    init: true
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_LOAD_SERVICE:-INFO}
      - SQLLDR_BLOB_CLOB_CHAR_LENGTH=${SQLLDR_BLOB_CLOB_CHAR_LENGTH:-20000}
      - SPARK_DATE_TIMESTAMP_FORMAT=${DATE_TIMESTAMP_FORMAT:-yyyy-MM-dd
HH:mm:ss.SSSS}
```

```
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-load-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/load_service:/opt/delphix/logs
  proxy:
    image: delphix-hyperscale-masking-proxy:${VERSION}
        init: true
        networks:
          - hyperscale-net
        ports:
          - "443:8443"
          - "80:8080"
    restart: unless-stopped
    depends_on:
      - controller-service
      #volumes:
      # Uncomment to bind mount /etc/config
      #- /nginx/config/path/on/host:/etc/config
networks:
  hyperscale-net:
volumes:
  hyperscale-load-data:
  hyperscale-unload-data:
  hyperscale-masking-data:
  hyperscale-controller-data:
```

- For File connector masking: A sample file is available in the package called `docker-compose-file-connector-sample.yaml.`

```
# Copyright (c) 2021, 2024 by Delphix. All rights reserved.
version: "3.7"
services:
    controller-service:
        image: delphix-controller-service-app:${VERSION}
        healthcheck:
          test: 'curl --fail --silent http://localhost:8080/actuator/health |
grep UP || exit 1'
          interval: 30s
          timeout: 25s
          retries: 3
          start_period: 30s
        depends_on:
          - unload-service
          - masking-service
          - load-service
        init: true
        networks:
          - hyperscale-net
        restart: unless-stopped
        volumes:
          - hyperscale-controller-data:/data
        environment:
          - API_KEY_CREATE=true
          - EXECUTION_STATUS_POLL_DURATION=120000
          - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=INFO
          - API_VERSION_COMPATIBILITY_STRICT_CHECK=false
          - LOAD_SERVICE_REQUIREPOSTLOAD=false
          - SKIP_UNLOAD_SPLIT_COUNT_VALIDATION=false
          - SKIP_LOAD_SPLIT_COUNT_VALIDATION=false
          - CANCEL_STATUS_POLL_DURATION=60000
          - SOURCE_KEY_FIELD_NAMES=unique_source_files_identifier
          # NFS storage configuration for staging area
          - NFS_STORAGE_HOST=${NFS_STORAGE_HOST:-}
          - NFS_STORAGE_EXPORT_PATH=${NFS_STORAGE_EXPORT_PATH:-}
          - NFS_STORAGE_MOUNT_TYPE=${NFS_STORAGE_MOUNT_TYPE:-}
          - NFS_STORAGE_MOUNT_OPTION=${NFS_STORAGE_MOUNT_OPTION:-}
          - APPLICATION_NAME=${APPLICATION_NAME:-hs-staging-area}
    unload-service:
        image: delphix-file-connector-unload-service-app:${VERSION}
        init: true
        networks:
          - hyperscale-net
        restart: unless-stopped
        volumes:
          - hyperscale-unload-data:/data
          # Staging area volume mount, here /mnt/parent_staging_area is used as
an example
          - /mnt/parent_staging_area:/etc/hyperscale
          # Source files should be made available within the unload-service
container file system
          # The paths within the container should be configured in the source
section of connector-info [with type=FS]
          - /mnt/source_files:/mnt/source
          #- /mnt/source_files2:/mnt/source2
          # In case source is hadoop
          #- /path/to/keytab_file/hadoop.keytab:/app/hadoop.keytab
          #- /path/to/hadoop/core-site.xml:/app/hadoop/etc/hadoop/core-site.xml
          #- /path/to/etc/krb5.conf:/etc/krb5.conf
          # In case user want to provide external Hadoop client for pyarrow
writer then they can provide
          #- /path/to/hadoop/client/hadoop:/app/hadoop
    masking-service:
        image: delphix-masking-service-app:${VERSION}
        init: true
        networks:
          - hyperscale-net
        restart: unless-stopped
        volumes:
          - hyperscale-masking-data:/data
          # Staging area volume mount, here /mnt/parent_staging_area is used as
an example
          - /mnt/parent_staging_area:/etc/hyperscale
        environment:
          - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=INFO
          - INTELLIGENT_LOADBALANCE_ENABLED=true
    load-service:
        image: delphix-file-connector-load-service-app:${VERSION}
        init: true
        networks:
          - hyperscale-net
        restart: unless-stopped
        volumes:
          - hyperscale-load-data:/data
          # Staging area volume mount, here /mnt/parent_staging_area is used as
an example
          - /mnt/parent_staging_area:/etc/hyperscale
          # Target location should be made available within the load-service
container file system
          # The paths within the container should be configured in the target
section of connector-info [with type=FS]
          - /mnt/target_files:/mnt/target
          #- /mnt/target_files2:/mnt/target2
          # In case target is hadoop
          #- /path/to/keytab_file/hadoop.keytab:/app/hadoop.keytab
          #- /path/to/hadoop/core-site.xml:/app/hadoop/etc/hadoop/core-site.xml
          #- /path/to/etc/krb5.conf:/etc/krb5.conf
          # In case user want to provide external Hadoop client for pyarrow
writer then they can provide
          #- /path/to/hadoop/client/hadoop:/app/hadoop
    proxy:
        image: delphix-hyperscale-masking-proxy:${VERSION}
        init: true
        networks:
          - hyperscale-net
        ports:
          - "443:8443"
          - "80:8080"
        restart: unless-stopped
        depends_on:
          - controller-service
networks:
    hyperscale-net:
volumes:
    hyperscale-load-data:
    hyperscale-unload-data:
    hyperscale-masking-data:
    hyperscale-controller-data:
```

- For MongoDB data source masking:

```
# Copyright (c) 2021, 2023 by Delphix. All rights reserved.
version: "3.7"
services:
    controller-service:
        build:
            context: controller-service
            args:
              - VERSION=${VERSION}
        image: delphix-controller-service-app:${VERSION}
        healthcheck:
          test: 'curl --fail --silent http://localhost:8080/actuator/health |
grep UP || exit 1'
          interval: 30s
          timeout: 25s
          retries: 3
          start_period: 30s
        depends_on:
          - unload-service
          - masking-service
          - load-service
        init: true
        networks:
          - hyperscale-net
        restart: unless-stopped
        volumes:
         - hyperscale-controller-data:/data
         # The orchestrator VM paths(left side of colon) used here are
examples. Configure the respective mount paths.
         - /home/hyperscale_user/logs/controller_service:/opt/delphix/logs
         - /mnt/hyperscale:/etc/hyperscale

        environment:
          - API_KEY_CREATE=${API_KEY_CREATE:-false}
          - EXECUTION_STATUS_POLL_DURATION=$
{EXECUTION_STATUS_POLL_DURATION:-120000}
          - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=$
{LOG_LEVEL_CONTROLLER_SERVICE:-INFO}
          - API_VERSION_COMPATIBILITY_STRICT_CHECK=$
{API_VERSION_COMPATIBILITY_STRICT_CHECK:-false}
          - LOAD_SERVICE_REQUIREPOSTLOAD=${LOAD_SERVICE_REQUIRE_POST_LOAD:-true
}
          - SKIP_UNLOAD_SPLIT_COUNT_VALIDATION=$
{SKIP_UNLOAD_SPLIT_COUNT_VALIDATION:-false}
          - SKIP_LOAD_SPLIT_COUNT_VALIDATION=$
{SKIP_LOAD_SPLIT_COUNT_VALIDATION:-false}
          - CANCEL_STATUS_POLL_DURATION=${CANCEL_STATUS_POLL_DURATION:-60000}
          - NFS_STORAGE_HOST=${NFS_STORAGE_HOST:-}
          - NFS_STORAGE_EXPORT_PATH=${NFS_STORAGE_EXPORT_PATH:-}
          - NFS_STORAGE_MOUNT_TYPE=${NFS_STORAGE_MOUNT_TYPE:-}
          - NFS_STORAGE_MOUNT_OPTION=${NFS_STORAGE_MOUNT_OPTION:-}
          - APPLICATION_NAME=${APPLICATION_NAME:-hs-staging-area}
          - SOURCE_KEY_FIELD_NAMES=database_name,collection_name
          - VALIDATE_UNLOAD_ROW_COUNT_FOR_STATUS=$
{VALIDATE_UNLOAD_ROW_COUNT_FOR_STATUS:-false}
          - VALIDATE_MASKED_ROW_COUNT_FOR_STATUS=$
{VALIDATE_MASKED_ROW_COUNT_FOR_STATUS:-false}
          - VALIDATE_LOAD_ROW_COUNT_FOR_STATUS=$
{VALIDATE_LOAD_ROW_COUNT_FOR_STATUS:-false}
          - DISPLAY_BYTES_INFO_IN_STATUS=${DISPLAY_BYTES_INFO_IN_STATUS:-true}
          - DISPLAY_ROW_COUNT_IN_STATUS=${DISPLAY_ROW_COUNT_IN_STATUS:-false}

    unload-service:
        build:
            context: unload-service
            args:
              - VERSION=${VERSION}
        image: delphix-mongo-unload-service-app:${VERSION}
        init: true
        environment:
          - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_UNLOAD_SERVICE:-
INFO}
          - UNLOAD_FETCH_ROWS=${UNLOAD_FETCH_ROWS:-10000}
          - CONCURRENT_EXPORT_LIMIT=${CONCURRENT_EXPORT_LIMIT:-10}
          - HIKARI_MAX_LIFE_TIME=${UNLOAD_HIKARI_MAX_LIFE_TIME:-1800000}
          - HIKARI_KEEP_ALIVE_TIME=${UNLOAD_HIKARI_KEEP_ALIVE_TIME:-300000}
          - FILE_DELIMITER=${FILE_DELIMITER:-,}
          - FILE_ENCLOSURE=${FILE_ENCLOSURE:-"}
          - FILE_ESCAPE_ENCLOSURE=${FILE_ESCAPE_ENCLOSURE:-"}

        networks:
          - hyperscale-net
        restart: unless-stopped
        volumes:
          - hyperscale-unload-data:/data
          # The orchestrator VM paths(left side of colon) used here are
examples. Configure the respective mount paths.
          - /mnt/hyperscale:/etc/hyperscale
          - /home/hyperscale_user/logs/unload_service:/opt/delphix/logs

    masking-service:
        build:
            context: masking-service
            args:
              - VERSION=${VERSION}
        image: delphix-masking-service-app:${VERSION}
        init: true
        networks:
          - hyperscale-net
        restart: unless-stopped
        volumes:
          - hyperscale-masking-data:/data
          # The orchestrator VM paths(left side of colon) used here are
examples. Configure the respective mount paths.
```

```
          - /mnt/hyperscale:/etc/hyperscale
          - /home/hyperscale_user/logs/masking_service:/opt/delphix/logs
        environment:
          - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_MASKING_SERVICE:-
INFO}
          - INTELLIGENT_LOADBALANCE_ENABLED=$
{INTELLIGENT_LOADBALANCE_ENABLED:-true}

load-service:
        build:
            context: load-service
            args:
              - VERSION=${VERSION}
        image: delphix-mongo-load-service-app:${VERSION}
        init: true
        environment:
          - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_LOAD_SERVICE:-
INFO}
          - SQLLDR_BLOB_CLOB_CHAR_LENGTH=${SQLLDR_BLOB_CLOB_CHAR_LENGTH:-20000}
          - HIKARI_MAX_LIFE_TIME=${LOAD_HIKARI_MAX_LIFE_TIME:-1800000}
          - HIKARI_KEEP_ALIVE_TIME=${LOAD_HIKARI_KEEP_ALIVE_TIME:-300000}
       networks:
          - hyperscale-net
        restart: unless-stopped
        volumes:
          - hyperscale-load-data:/data
          # The orchestrator VM paths(left side of colon) used here are
examples. Configure the respective mount paths.
          - /mnt/hyperscale:/etc/hyperscale
          - /home/hyperscale_user/logs/load_service:/opt/delphix/logs

proxy:
        image: delphix-hyperscale-masking-proxy:${VERSION}
        init: true
        networks:
          - hyperscale-net
        ports:
          - "443:8443"
          - "80:8080"
        restart: unless-stopped
        depends_on:
          - controller-service
        #volumes:
          # Uncomment to bind mount /etc/config
          #- /nginx/config/path/on/host:/etc/config
networks:
    hyperscale-net:
volumes:
    hyperscale-load-data:
    hyperscale-unload-data:
    hyperscale-masking-data:
    hyperscale-controller-data:
```

5. **(Optional)** To modify the default Hyperscale configuration properties for the application, see Configuration Settings.
6. Run the application from the same location where you extracted the `docker-compose.yaml` file with the `docker-compose up -d` command.
   a. To check if the application is running, use the `docker-compose ps` command. The output of this command should show five containers up and running.
   b. To access application logs of a given container, run the `docker logs -f service_container_name>` command. The service container name can be received from the output of the `docker-compose ps` command.
   c. Run the `sudo docker-compose down` command to stop the application (if required).
7. Once the application starts, an API key will be generated that will be required to authenticate with the Hyperscale Compliance Orchestrator. This key will be found in the docker container logs of the controller service. You can either look for the key from the controller service logs location that was set as a volume binding in the `docker-compose.yaml` file or use the `docker logs -f <service_container_name>` command to retrieve the logs. The service container name can be received from the output of the `docker-compose ps` command.
   a. The above command displays an output similar to the following where the string `NEWLY GENERATED API KEY` can be received from the log:

```
2022-05-18 12:24:10.981  INFO 7 --- [           main] o.a.c.c.C.[Tomcat].
[localhost].[/]    : Initializing Spring embedded WebApplicationContext
2022-05-18 12:24:10.982  INFO 7 --- [           main]
w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext:
initialization completed in 9699 ms
NEWLY GENERATED API KEY: 1.89lPH1dHSJQwHuQvzawD99sf4SpBPXJADUmJS8v00VCF4V7
rjtRFAftGWygFfsqM
```

   b. To authenticate with the Hyperscale Compliance Orchestrator, you must use the API key and include the HTTP Authorization request header with the type apk; `apk <API Key>`.
   c. For more information, see the **Authentication** section under Accessing the Hyperscale Compliance API.

Continuous Compliance Engine Installation

Delphix Continuous Compliance Engine is a multi-user, browser-based web application that provides complete, secure, and scalable software for your sensitive data discovery, masking, and tokenization needs while meeting enterprise-class infrastructure requirements. For information about installing the Continuous Compliance Engine, see Continuous Compliance Engine Installation documentation.

Custom configuration

Docker Compose should only be used to deploy Hyperscale Compliance in an evaluation/testing capacity.

Introduction

This topic provides background information on performing custom configurations that are referenced throughout Hyperscale Compliance documentation.

Bind mounts

Configuration of Hyperscale Compliance is achieved through a combination of Configuration Settings and the use of Docker bind mounts. A bind mount is a directory or file on the host machine that will be mounted inside the container. Changes made to the files on the host machine will be reflected inside the container. It does not matter where the files live on the host machine, but the files must be mounted to specific locations inside the container so that the application can find them.

The Hyperscale Compliance and proxy containers can both be configured via separate bind-mounted directories. Each container requires all configuration files to be mounted to the relevant directory inside the container. Therefore, it is recommended to create a directory for each container on the host machine to store all of the configuration files and mount them to the relevant directory. This is done by editing the `docker-compose.yaml`.

Like, Under **proxy services**, add a **volumes** section if one does not already exist; this is used to mount the configuration directory on the host to `/etc/config`. For example, if `/my/proxy/config` is the directory on the host that contains the configuration files, then the relevant part of the compose file would look like this:

```
services:
    proxy:
        volumes:
            - /my/proxy/config:/etc/config
```

To change the configuration of the Hyperscale Compliance container, make a similar change under its service section, the only difference being the directory on the host. After making this change, the application will need to be stopped and restarted.

The structure of `/my/proxy/config` will need to match the required layout in `/etc/config`. When each container starts, it will create default versions of each file and place them in the expected location. It is highly recommended to start from the default version of these files. For example, if `/my/proxy/config` is the bind mount directory on the host, it could be populated with all the default configuration files by running the following commands.

First, create an `nginx` directory inside `/my/proxy/config` on the host.

```
cd /my/proxy/config
mkdir nginx
```

Find the **id** of the proxy container with **docker ps**. Look for the container with a **delphix-hyperscale-masking-proxy** image name. To determine the user and group ownership for any configuration files, start the containers and open a shell to the relevant one (nginx in this example), then examine the current user/group IDs associated with the files (where `x.0.0` should be changed to the version of Hyperscale Compliance being installed).

```
docker ps

CONTAINER ID   IMAGE                                 COMMAND                  CREATED
STATUS         PORTS                                       NAMES
802606779b9d   delphix-hyperscale-masking-proxy:dev   "/sbin/tini -- /boot…"   3
 hours ago   Up 3 hours          0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->4
43/tcp, :::443->443/tcp
hyperscale-masking_proxy_1
```

In the above example, `802606779b9d` is the **id**. Run the following command to copy the default files to the bind mount.

```
docker cp <container id>:/etc/config/nginx /my/proxy/config/nginx
```

One can always go back to the original configuration by removing the bind-mount and restarting the container or using docker cp as in the previous example to overwrite the custom files with the default versions.

Upgrading the Hyperscale Compliance Orchestrator (Docker Compose)

Pre-requisite

Before upgrading, ensure you have downloaded the Hyperscale Compliance x.0.0 (where `x.0.0` should be changed to the version of Hyperscale being installed) tar bundle from the Delphix Download website.

How to upgrade the Hyperscale Compliance Orchestrator

Perform the following steps to upgrade the Hyperscale Compliance Orchestrator to the x.0.0 version:

1. Run `cd /<hyperscale_installation_path>/` and `docker-compose down` to stop and remove all the running containers.
2. Run the below commands to delete all existing dangling images and hyperscale images:

   ```
   docker rmi $(docker images -f "dangling=true" -q)
   docker rmi $(docker images "delphix-hyperscale-masking-proxy" -q)
   docker rmi $(docker images "delphix-controller-service-app" -q)
   docker rmi $(docker images "delphix-masking-service-app" -q)
   docker rmi $(docker images "delphix-*load-service-app" -q)
   ```

3. Remove all files or folders from existing installation directories, except `docker-compose.yaml` (Keep its backup outside the installation directory so it is not overridden while executing the next step).
4. Take backup of .env file and untar the patch tar in your existing installation path (where `x.0.0` should be changed to the version of Hyperscale being installed). `tar -xzvf delphix-hyperscale-masking-x.0.0.tar.gz -C <existing_installation_path>`
5. Replace the `docker-compose.yaml` supplied with the bundle file as per the following:
   - **For users upgrading from 3.0.0.x:** Use the connector-specific `docker-compose-sample.yaml` file(e.g. `docker-compose-oracle.yaml` or `docker-compose-mssql.yaml`) supplied with the bundle and add the same 'volumes' and/or any other properties (if configured) for each container referencing the backed-up `docker-compose.yaml` from step 3.
   - **For users upgrading from 4.0.0.0 and above:** Replace the `docker-compose.yaml` file supplied with the bundle with the `docker-compose.yaml` file that you created as a backup at step 3.
   - Similar to other services, make sure to add the volume binding for the staging area path under controller-service as well. For example,

     ```
     - /mnt/hyperscale:/etc/hyperscale
     ```

6. Apply the backed up .env file and set the VERSION property as x.0.0 (i.e. VERSION=x.0.0).
7. Run the below commands to load the images(will configure Oracle-based unload/load setup):

   ```
   docker load --input controller-service.tar
   docker load --input unload-service.tar
   docker load --input masking-service.tar
   docker load --input load-service.tar
   ```

```
docker load --input proxy.tar
```

- If upgrading from an MSSQL connector setup(supported starting 5.0.0.0 release), instead of running the above commands for load/unload services setup(which are for Oracle), run the below commands(rest remains same for the controller, masking, and proxy services):

  ```
  docker load --input mssql-unload-service.tar
  docker load --input mssql-load-service.tar
  ```

- If upgrading from a Delimited/Parquet Files connector setup (supported starting 12.0.0 release), instead of running the above commands for load/unload services setup(which are for Oracle), run the below commands(rest remains same for the controller, masking, and proxy services) after updating new image names in docker-compose.yaml:

  ```
  docker load --input file-connector-unload-service.tar
  docker load --input file-connector-load-service.tar
  ```

- If upgrading from a MongoDB connector setup (supported starting 13.0.0 release), instead of running the above commands for load/unload services setup (which are for Oracle), run the below commands (rest remains same for the controller, masking, and proxy services):

  ```
  docker load --input mongo-unload-service.tar
  docker load --input mongo-load-service.tar
  ```

- If upgrading from a Snowflake connector setup (supported starting 28.0.0 release), instead of running the above commands for load/unload services setup (which are for Oracle), run the below commands (rest remains same for the controller, masking, and proxy services):

  ```
  docker load --input snowflake-unload-service.tar
  docker load --input snowflake-load-service.tar
  ```

8. Make sure to have the below ports configured under proxy service:

```
ports:
    - "443:8443"
    - "80:8080"
```

9. Ensure your mount are configured and accessible, before running a job.
If upgrading to version 24.0 (and onwards) , ensure that the location mounted on the Hyperscale host is the same as the one mapped to `/etc/hyperscale` in your docker-compose.yaml. If a previous mount exists on another location, unmount it and re-mount to the right directory e.g. if a mount point exists with name `staging_area` at path `/mnt/provision/staging_area` , execute the following commands and restart the containers.
   a. If NFS file server is used as staging server execute this command:

   ```
   sudo umount /mnt/provision/staging_area
   ```

```
sudo mount -t nfs4 <source_nfs_endpoint>:<source_nfs_location> /mnt/
provision
```

b. If the NFS Server installation is a Delphix Continuous Data Engine empty VDB you can either:
   i. Append the staging_area path to the volume binding for all the services in docker-compose.yaml. For example:

   ```
   volumes:
         - /mnt/provision/staging_area:/etc/hyperscale
   ```

   ii. Alternatively, you can update mount path of Environment on Continuous Data Engine:
      1. Disable the empty VDB(Data Set).
      2. Update path on Environment → Databases page. For example: change path from `/mnt/provision/staging_area` to `mnt/provision/`
      3. Enable the empty VDB(Data Set).
      4. Restart Hyperscale containers.

After re-mounting, recheck the permissions of staging area on Hyperscale host. Refer to instruction number 3 on this page: Installation and Setup for required staging area permissions.

> Upon application startup, all existing mount-filsystems will be deleted. Please ensure to take backup of the mount setup details, if needed.

> If using file connectors, any unload or load jobs in a running state at the time of a container restart are marked as failed.

10. Run `docker-compose up -d` to create containers.

Managing the storage space

There are two storage locations where the continuous increase in disk space consumption can lead to a depletion of the available disk space on the Hyperscale host system.

1. Overlay2 File System

The `/var/lib/docker/overlay2` directory stores several file system layers for images and containers, and it may also accumulate data related to unused containers and images.

To prevent excessive disk space usage caused by the Overlay2 File System, it's essential to follow the Hyperscale upgrade steps, including the deletion of dangling images as specified. If this storage becomes full, run the below command:
`docker rmi $(docker images -f "dangling=true" -q)`

2. Container Logs

Docker maintains logs printed on the console by each service in JSON format within the `/var/lib/docker/containers` directory (because the default driver is json-file). These logs keep appending to these files. If this storage becomes full, perform the below steps:

1. Create a `daemon.json` file under `/etc/docker` directory (For more information, refer to https://docs.docker.com/config/containers/logging/local/#usage%5D & Configure logging driver). The below example describes the sample content of the file.

   ```
   {
     "log-driver": "json-file",
     "log-opts": {
       "max-size": "2g",
       "max-file": "3"
     }
   }
   ```

2. Reload & restart the docker service.

   ```
   sudo systemctl daemon-reload
   sudo systemctl restart docker
   ```

3. Stop all currently running docker services & then restart these again. Execute the below commands from the HS directory.

   ```
   docker-compose down
   docker-compose up -d
   ```

Migrating to Kubernetes

1. Copy docker-compose services data to local storage

Stop Hyperscale services so that all of the product's state is flushed to persistent storage. You must run the below command from the same directory where the `docker-compose.yaml` file exists.

```
docker-compose stop
```

Create folders for data backup of all services.

```
mkdir container_data
cd container_data/
mkdir controller
mkdir unload
mkdir masking
mkdir load
mkdir proxy
cd ../
```

Copy the SQLite database file and encryption key Docker volume folder data on a local machine in the created folder.

```
docker ps -a // list out the name of containers

docker cp <controller-service-container-name>:/data ./container_data/controller/
docker cp <unload-service-container-name>:/data ./container_data/unload/
docker cp <load-service-container-name>:/data ./container_data/load/
docker cp <masking-service-container-name>:/data ./container_data/masking/

eg: docker cp hyperscale-masking_controller-service_1:/data ./container_data/
controller/
```

Copy the SSL certificates Docker volume folder data on a local machine in the created folder (This step is only required if custom certificates are used).

```
docker cp <proxy-container-name>:/etc/config/nginx/ssl ./container_data/proxy
```

All the `container_data` files should be as below.

```
├── controller
│   └── data
│       └── db.sqlite
├── load
│   └── data
│       ├── db.sqlite
│       └── encryption.key
├── masking
```

```
│   └── data
│       ├── db.sqlite
│       └── encryption.key
├── proxy
│   └── ssl
│       ├── dhparam.pem
│       ├── nginx.crt
│       ├── nginx.key
│       └── ssl.conf
└── unload
    └── data
        ├── db.sqlite
        └── encryption.key
```

Grant permission 770 to all files.

```
chmod 770 -R .
```

Please note that docker-compose log files will not be migrated so keep the backup, of log files before starting the migration process.

2. Restore docker-compose services data to Hyperscale deployed on the Kubernetes setup (in Persistent Volume)

Install the required version of the Hyperscale Compliance Engine using Kubernetes. For more information about the installation steps, refer to thehttps://hyperscalemasking.delphix.com/docs/latest/installation-and-setup-kubernetes section.

List the pods name for further reference:

```
kubectl get pods -n hyperscale-services
```

Restore Hyperscale docker-compose version volume data with Hyperscale deployed on the Kubernetes setup (in Persistent Volume).

Execute the below process for all the pods names except the proxy pod to copy each service data.

```
cd container_data/<service- folder-name>
kubectl cp data hyperscale-services/<service-pod-name>:/

eg:
cd container_data/masking
kubectl cp data hyperscale-services/masking-service-7788ccbbbb-lnzhv:/
```

3. List out deployment names and restart all the services using kubectl rollout.

```
kubectl get deployments -n hyperscale-services
kubectl rollout restart deployment <controller-service-deployment-name>  -n
hyperscale-services
eg: kubectl rollout restart deployment controller-service  -n hyperscale-services
```

4. Update the SSL certificates (This step is only required if custom certificates are used).

Go to `container_data/proxy/ssl` folder and get the base64 value of `nginx.crt`, `nginx.key,` and `dhparam.pem`. For example:

```
cat nginx.crt | base64 | awk '{print}' ORS='' | awk '{print}'
```

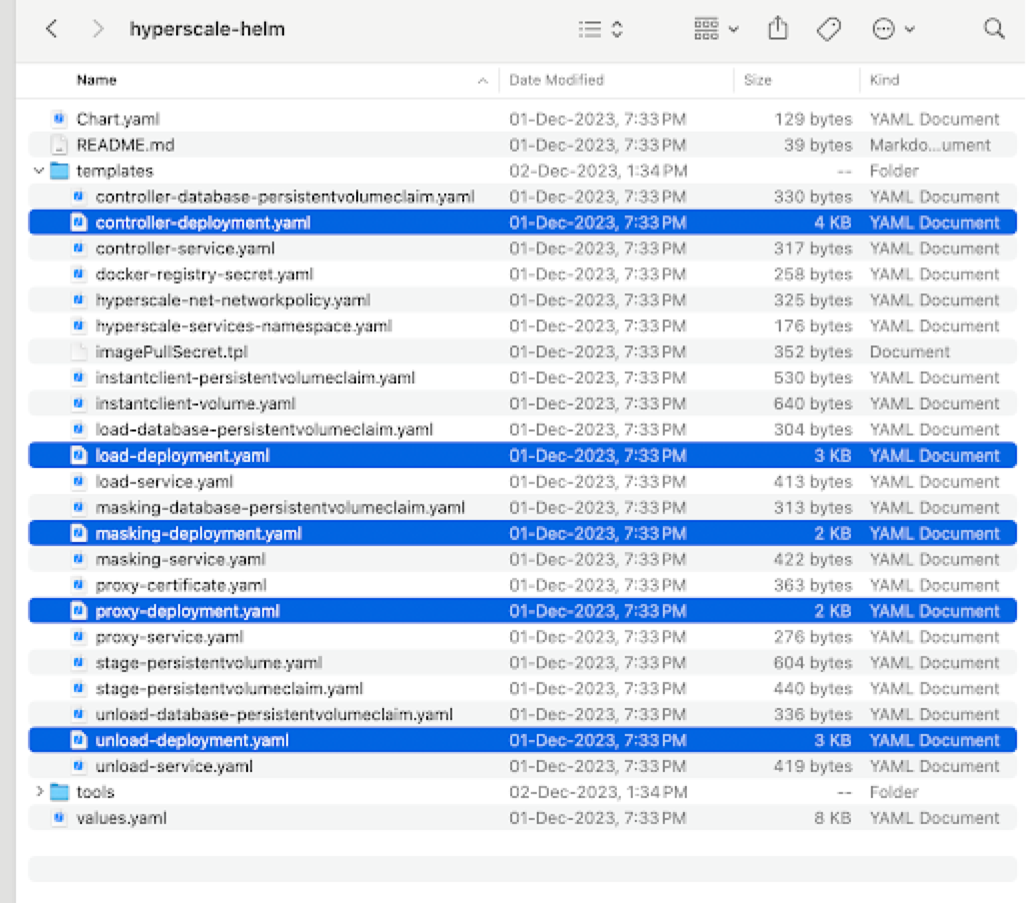
Now update the base64 value of each in k8 `values.yaml` file.

5. Update the other user-configurable properties from docker-compose .env file to values.yaml accordingly

All the available configurations can be found at https://hyperscalemasking.delphix.com/docs/latest/configuration-settings , the configuration migration is divided into two steps:

Modify existing configuration for any service:

Go to the `hyperscale-helm` chart directory. Under the templates folder, locate the corresponding `<service-name>-deployment.yaml` file. For more details, refer to the below screenshot.

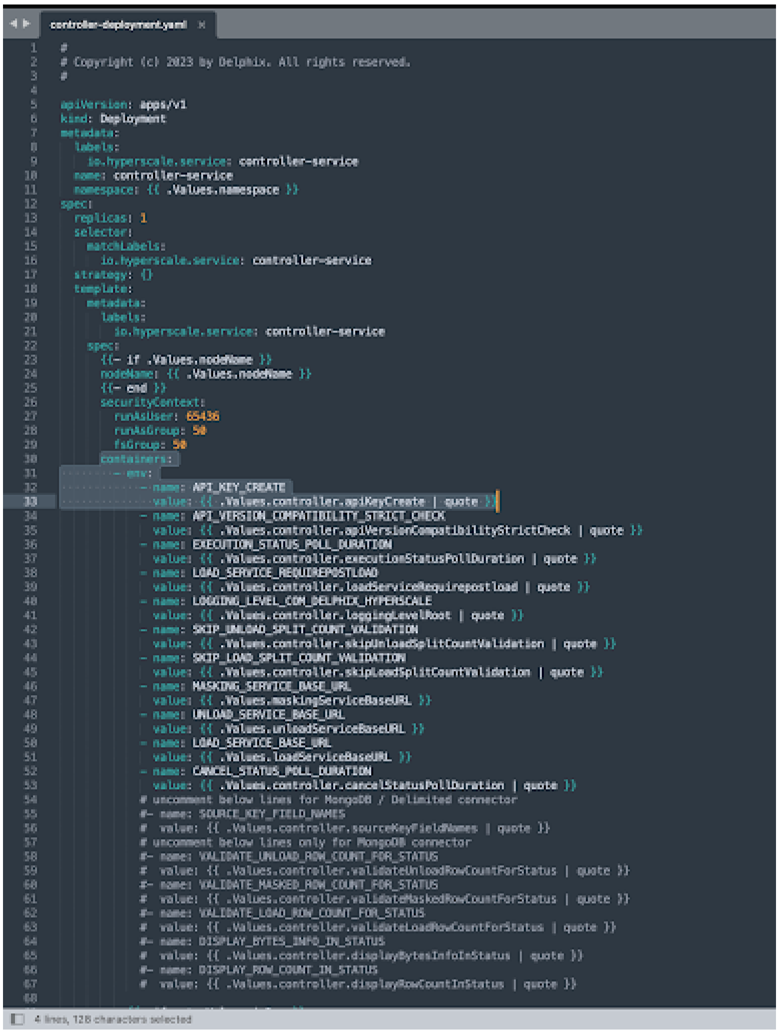


hyperscale-helm

Name	Date Modified	Size	Kind
Chart.yaml	01-Dec-2023, 7:33 PM	129 bytes	YAML Document
README.md	01-Dec-2023, 7:33 PM	39 bytes	Markdo...ument
templates	02-Dec-2023, 1:34 PM	--	Folder
controller-database-persistentvolumeclaim.yaml	01-Dec-2023, 7:33 PM	330 bytes	YAML Document
controller-deployment.yaml	01-Dec-2023, 7:33 PM	4 KB	YAML Document
controller-service.yaml	01-Dec-2023, 7:33 PM	317 bytes	YAML Document
docker-registry-secret.yaml	01-Dec-2023, 7:33 PM	258 bytes	YAML Document
hyperscale-net-networkpolicy.yaml	01-Dec-2023, 7:33 PM	325 bytes	YAML Document
hyperscale-services-namespace.yaml	01-Dec-2023, 7:33 PM	176 bytes	YAML Document
imagePullSecret.tpl	01-Dec-2023, 7:33 PM	352 bytes	Document
instantclient-persistentvolumeclaim.yaml	01-Dec-2023, 7:33 PM	530 bytes	YAML Document
instantclient-volume.yaml	01-Dec-2023, 7:33 PM	640 bytes	YAML Document
load-database-persistentvolumeclaim.yaml	01-Dec-2023, 7:33 PM	304 bytes	YAML Document
load-deployment.yaml	01-Dec-2023, 7:33 PM	3 KB	YAML Document
load-service.yaml	01-Dec-2023, 7:33 PM	413 bytes	YAML Document
masking-database-persistentvolumeclaim.yaml	01-Dec-2023, 7:33 PM	313 bytes	YAML Document
masking-deployment.yaml	01-Dec-2023, 7:33 PM	2 KB	YAML Document
masking-service.yaml	01-Dec-2023, 7:33 PM	422 bytes	YAML Document
proxy-certificate.yaml	01-Dec-2023, 7:33 PM	363 bytes	YAML Document
proxy-deployment.yaml	01-Dec-2023, 7:33 PM	2 KB	YAML Document
proxy-service.yaml	01-Dec-2023, 7:33 PM	276 bytes	YAML Document
stage-persistentvolume.yaml	01-Dec-2023, 7:33 PM	604 bytes	YAML Document
stage-persistentvolumeclaim.yaml	01-Dec-2023, 7:33 PM	440 bytes	YAML Document
unload-database-persistentvolumeclaim.yaml	01-Dec-2023, 7:33 PM	336 bytes	YAML Document
unload-deployment.yaml	01-Dec-2023, 7:33 PM	3 KB	YAML Document
unload-service.yaml	01-Dec-2023, 7:33 PM	419 bytes	YAML Document
tools	02-Dec-2023, 1:34 PM	--	Folder
values.yaml	01-Dec-2023, 7:33 PM	8 KB	YAML Document

Open the corresponding `<service-name>-deployment.yaml` file for which you want to modify the new configuration.

Example: Let's suppose you want to change the config value for `API_KEY_CREATE` in `controller-deployment.yaml` file. As you can see that `API_KEY_CREATE` config value is mapped with `Values.controller.apiKeyCreate`.



```
#
# Copyright (c) 2023 by Delphix. All rights reserved.
#

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    io.hyperscale.service: controller-service
  name: controller-service
  namespace: {{ .Values.namespace }}
spec:
  replicas: 1
  selector:
    matchLabels:
      io.hyperscale.service: controller-service
  strategy: {}
  template:
    metadata:
      labels:
        io.hyperscale.service: controller-service
    spec:
      {{- if .Values.nodeName }}
      nodeName: {{ .Values.nodeName }}
      {{- end }}
      securityContext:
        runAsUser: 65436
        runAsGroup: 50
        fsGroup: 50
      containers:
        - env:
            - name: API_KEY_CREATE
              value: {{ .Values.controller.apiKeyCreate | quote }}
            - name: API_VERSION_COMPATIBILITY_STRICT_CHECK
              value: {{ .Values.controller.apiVersionCompatibilityStrictCheck | quote }}
            - name: EXECUTION_STATUS_POLL_DURATION
              value: {{ .Values.controller.executionStatusPollDuration | quote }}
            - name: LOAD_SERVICE_REQUIREPOSTLOAD
              value: {{ .Values.controller.loadServiceRequirepostload | quote }}
            - name: LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE
              value: {{ .Values.controller.loggingLevelRoot | quote }}
            - name: SKIP_UNLOAD_SPLIT_COUNT_VALIDATION
              value: {{ .Values.controller.skipUnloadSplitCountValidation | quote }}
            - name: SKIP_LOAD_SPLIT_COUNT_VALIDATION
              value: {{ .Values.controller.skipLoadSplitCountValidation | quote }}
            - name: MASKING_SERVICE_BASE_URL
              value: {{ .Values.maskingServiceBaseURL }}
            - name: UNLOAD_SERVICE_BASE_URL
              value: {{ .Values.unloadServiceBaseURL }}
            - name: LOAD_SERVICE_BASE_URL
              value: {{ .Values.loadServiceBaseURL }}
            - name: CANCEL_STATUS_POLL_DURATION
              value: {{ .Values.controller.cancelStatusPollDuration | quote }}
            # uncomment below lines for MongoDB / Delimited connector
            #- name: SOURCE_KEY_FIELD_NAMES
            #  value: {{ .Values.controller.sourceKeyFieldNames | quote }}
            # uncomment below lines only for MongoDB connector
            #- name: VALIDATE_UNLOAD_ROW_COUNT_FOR_STATUS
            #  value: {{ .Values.controller.validateUnloadRowCountForStatus | quote }}
            #- name: VALIDATE_MASKED_ROW_COUNT_FOR_STATUS
            #  value: {{ .Values.controller.validateMaskedRowCountForStatus | quote }}
            #- name: VALIDATE_LOAD_ROW_COUNT_FOR_STATUS
            #  value: {{ .Values.controller.validateLoadRowCountForStatus | quote }}
            #- name: DISPLAY_BYTES_INFO_IN_STATUS
            #  value: {{ .Values.controller.displayBytesInfoInStatus | quote }}
            #- name: DISPLAY_ROW_COUNT_IN_STATUS
            #  value: {{ .Values.controller.displayRowCountInStatus | quote }}

```

Now under the `hyperscale-helm` chart directory, open the `values.yaml` file and search for `apiKeyCreate` property of controller attribute and change the corresponding value.

Add a new configuration for any service

In case the configuration name doesn't exist in `<service-name>-deployment.yaml` file, add your configuration as part of env in `<service-name>-deployment.yaml` file and also define config with the same name in `values.yaml` corresponding to the service environment as explained above, to get the values from `values.yaml` file.

Apply helm upgrade to reflect the new `values.yaml` file changes.

```
helm upgrade hyperscale-helm -f hyperscale-helm/values.yaml hyperscale-helm
```

Your docker-compose Hyperscale VM is now fully migrated to Kubernetes Hyperscale VM. Check the k8 running pods, like below:

```
delphix@hyperscale-k8s:~/_database$ kubectl get pods -n hyperscale-services
NAME                                 READY    STATUS     RESTARTS    AGE
masking-service-7788ccbbbb-lnzhv     1/1      Running    0           4h28m
unload-service-5f9946556f-4qg7l      1/1      Running    0           4h28m
proxy-767d6ccbdd-2vdhv               1/1      Running    0           4h28m
load-service-5bb994dbbf-c85mz        1/1      Running    0           3h43m
controller-service-6c6df56bbd-blng6  1/1      Running    0           62m
```

If `apiKeyCreate` property is defined as true in `values.yaml`, then you can get the API-Key as below to access the API. In case, `apiKeyCreate` property is defined as false then you can use your existing API-Key.

```
kubectl logs <controller-pod-name> -n hyperscale-services | grep 'NEWLY GENERATED API
KEY' | sed 's/^.*: //'
```

Kubernetes

This section covers the following topics:

- Host requirements (Kubernetes)
- Docker image registry and names for Hyperscale services
- Installation and Setup (Kubernetes)
- Ingress setup (Kubernetes)
- Bootstrapping API keys (Kubernetes)
- Hyperscale logs (Kubernetes)
- Limitations (Kubernetes)
- Upgrading the Hyperscale Compliance Orchestrator (Kubernetes)

Host requirements (Kubernetes)

Type	Host Requirement	Explanation
User	A user (example: hyperscale_os) with the following permissions is required: • Permission to run helm commands • Permission to run `kubectl` commands	These permissions are required to be able to install helm charts and manage the Kubernetes resources.
Staging area directory permissions	The staging area directory requires the following permissions based on the UID/GID of the OS user so that the Hyperscale Compliance Orchestrator and the Continuous Compliance Engine(s) can perform read/write/execute operations on the staging area: • If the Hyperscale Compliance OS user has a UID of 65436, then the staging area directory must have a UID of 65436 and 700 permission mode. • If the Hyperscale Compliance OS user has a GID of 50 and does not have a UID of 65436, then the staging area directory must have a GID of 50 and 770 permission mode.	These permissions on the staging area directory are required for the Hyperscale Compliance containers and the continuous compliance engines to be able to read/write into the staging area.
Installation Directory	A directory to download and manage helm charts.	
Hardware Requirements	**Minimum:** 8 vCPU, 64 GB of memory, 50GB OS disk. **Recommended:** 16 vCPU, 128GB of memory, 50GB OS disk.	

Docker image registry and names for Hyperscale services

The docker images for Hyperscale Compliance services fall under two categories of services:

1. **Common services**: A set of service images common across deployments with varying unload/load services.
2. **Unload and Load services**: A set of unload and load service images specific to a dataset vendor.

All images are stored under the hyperscale.download.delphix.com/delphix-hyperscale registry. Also, any image will have a name of the form `<image-name>-[VERSION]` where [VERSION] corresponds to a particular release version.

The following list provides the image URLs for the docker images in the two categories of services:

Common services

```
hyperscale.download.delphix.com/delphix-hyperscale:proxy-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:controller-service-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:masking-service-[VERSION]
```

Unload and Load services

Oracle Unload and Load

```
hyperscale.download.delphix.com/delphix-hyperscale:oracle-unload-service-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:oracle-load-service-[VERSION]
```

MSSQL Unload and Load

```
hyperscale.download.delphix.com/delphix-hyperscale:mssql-unload-service-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:mssql-load-service-[VERSION]
```

File Connector Unload and Load

```
hyperscale.download.delphix.com/delphix-hyperscale:file-connector-unload-service-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:file-connector-load-service-[VERSION]
```

Mongo Unload and Load

```
hyperscale.download.delphix.com/delphix-hyperscale:mongo-unload-service-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:mongo-load-service-[VERSION]
```

Snowflake Unload and Load

```
hyperscale.download.delphix.com/delphix-hyperscale:snowflake-unload-service-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:snowflake-load-service-[VERSION]
```

Installation and Setup (Kubernetes)

Hyperscale Compliance is designed to run and is supported on any Certified Kubernetes platform (https://www.cncf.io/certification/software-conformance) that supports Helm (https://helm.sh/docs/topics/kubernetes_distros/). Microk8s has been explicitly tested by Delphix and is recommended for use. The product is also OCI-compliant and may use any container runtime within a certified Kubernetes platform that implements the OCI Runtime Specification including CRI-O, Docker, and Podman.
Delphix regularly tests against a range of popular Kubernetes platforms to cover a representative sample of implementations. The following Kubernetes platforms have been explicitly tested by Delphix and are recommended for use: Microk8s, AWS EKS, and OpenShift on VMWare vSphere.

Installation Requirements

To deploy Hyperscale Compliance via Kubernetes, a running Kubernetes cluster is required to run, the `kubectl` c ommand line tool to interact with the Kubernetes cluster and HELM for deployment onto the cluster.

Requirement	**Recommended Version**	**Comments**
Kubernetes Cluster	1.25 or above	If you want to install MicroK8s, follow the steps mentioned in the MicroK8s on Linux (online mode).
HELM	3.9.0 or above	HELM installation should support HELM v3. More information on HELM can be found at `https://helm.sh/docs/` . To install HELM, follow the installation instructions at `https://helm.sh/docs/intro/install/` . The installation also requires access to the HELM repository from where Hyperscale charts can be downloaded. The HELM repository URL is `https://dlpx-helm-hyperscale.s3.amazonaws.com` .
kubectl	1.25.0 or above	

- If an intermediate HELM repository is to be used instead of the default Delphix HELM repository, then the repository URL, username, and password to access this repository needs to be configured in the `values.yaml` file under the **imageCredentials** section.
- HELM will internally refer to the kubeconfig file to connect to the Kubernetes cluster. The default kubeconfig file is present at location: `~/.kube/config` .
- If the kubeconfig file needs to be overridden while running HELM commands, set the ***KUBECONFIG*** environment variable to the location of the kubeconfig file.
- Oracle Load doesn’t support Object Identifiers(OIDs).

Installation

Download the HELM charts

The latest version of the chart can be pulled locally with the following command (where `x.x.x` should be changed to the version of Hyperscale being installed):

```
curl -XGET https://dlpx-helm-hyperscale.s3.amazonaws.com/hyperscale-helm-x.x.x.tgz -o hyperscale-helm-x.x.x.tgz
```

This command will download a file with the name `hyperscale-helm-x.x.x.tgz` in the current working directory. The downloaded file can be extracted using the following command (where `x.x.x` should be changed to the version of Hyperscale being installed):

```
tar -xvf hyperscale-helm-x.x.x.tgz
```

This will extract into the following directory structure:

```
hyperscale-helm
    ├── Chart.yaml
    ├── README.md
    ├── templates
        │-<all templates files>
    ├── tools
        │-<all tool files>
    ├── values-file-connector.yaml
    ├── values-mongo.yaml
    ├── values-snowflake.yaml
    ├── values-mssql.yaml
    ├── values-oracle.yaml
    └── values.yaml
```

Verify the authenticity of the downloaded HELM charts

The SHA-256 hash sum of the downloaded helm chart tarball file can be verified as follows:

1. Execute the below command and note the digest value for version x.x.x (where `x.x.x` should be changed to the version of Hyperscale being installed)
   curl `https://dlpx-helm-hyperscale.s3.amazonaws.com/index.yaml`
2. Execute the sha256sum command (or equivalent) on the downloaded file (where `x.x.x` should be changed to the version of Hyperscale being installed) (hyperscale-helm-x.x.x.tgz)
   `sha256sum hyperscale-helm-x.x.x.tgz`

The value generated by the sha256sum utility in step 2 must match the digest value noted in step 1.

Configure Registry Credentials for Docker Images

For pulling the Docker images from the registry, permanent credentials associated with your Delphix account would need to be configured in the `values.yaml` file. To get these permanent credentials, visit the Hyperscale Compliance Download page and log in with your credentials. Once logged in, select the Hyperscale HELM Repository link and accept the Terms and Conditions. Once accepted, credentials for the docker image registry will be presented. Note them down and edit the `imageCredentials.username` and `imageCredentials.password` properties in the `values.yaml` file as shown below:

```
# Credentials to fetch Docker images from Delphix internal repository
      imageCredentials:
# Username to login to docker registry
      username: <username>
# Password to login to docker registry
      password: <password>
```

> Delphix will delete unused credentials after 30 days and inactive (but previously used) credentials after 90 days.

Helm chart configuration files

`hyperscale-helm` is the name of the folder that was extracted in the previous step. In the above directory structure, there are essentially two files that come into play while attempting to install the helm chart:

1. A `values.yaml` configuration file that contains configurable properties, common to all the services, with their default values.
2. A `values-[connector-type].yaml` configuration file that contains configurable properties, applicable to the services of the specific connector, with their default values.

The following sections talk about some of the important properties that will need to be configured correctly for a successful deployment. A full list of the configurable properties can be found on the Configuration Settings page.

(Mandatory) Configure the staging area volume

A volume will need to be mounted, via persistent volume claims, inside the pods that will provide access to the staging area for the hyperscale compliance services. In Kubernetes deployment of Hyperscale, we can use NFS server or AWS S3 buckets for mounting staging area volume. This can be configured in the following ways that involves setting/overriding some properties in the `values.yaml` configuration file:

1. Set the value for `stagingstorageType` as MOUNT(for NFS Server) or AWS_S3(for S3 bucket).
2. If `stagingstorageType` is MOUNT, set values for the following properties if the cluster needs to mount an NFS shared path from an NFS server. For information about setting up and configuring an NFS server for the staging area volume, refer to NFS Server Installation.

- nfsStorageHost
- nfsStorageExportPath
- nfsStorageMountOption
- nfsStorageMountType

3. If `stagingstorageType` is AWS_S3, set values for the following properties as required if the cluster needs to bind the pods to a persistent volume using the S3 CSI driver. Refer to Configuring AWS S3 bucket as staging area for more details.

- authMechanism
- awsBucketName
- awsBucketRegion
- awsBucketPrefix

- awsBucketDelimiter
- awsAccessKey
- awsSecretKey
- stagingAwsS3SecretName

Installing the helm chart with these properties set will create a persistent volume on the cluster. As such, the user installing the helm chart should either be a cluster-admin or should have the privileges to be able to create persistent volume on the cluster. Otherwise, either of the following set of properties must also be set:

1. **stagePvcName:** Set this property if the cluster needs to bind the pods to a persistent volume claim. Note that until this PVC is bound to a backing PV, the pods will not start getting created and as such, the cluster admin should ensure that the backing PV is either statically provisioned or dynamically provisioned based on the storage class associated with PVC.
2. **stagePvName and stageStorageClass:** Set these properties if the cluster needs to bind the pods to a persistent volume with the associated storage class name. Once the helm chart installation starts, a PVC will be created that is managed by the helm. ***Note:*** *stageStorageClass is not required for AWS S3.*

The following properties are supporting/optional properties that can be overridden along with the above properties:

1. **nfsStorageMountOption:** If ***nfsStorageHost*** and ***nfsStorageExportPath*** have been set, set the appropriate mount option if you would like the cluster to mount with an option other than the default option of `nfsvers=4.2.`
2. **stageAccessMode and stageStorageSize:** Persistent Volume claims can request specific storage capacity size and access modes.

(Mandatory for Oracle) Configure the instantclient volume

A volume will need to be mounted, via persistent volume claims, inside the Oracle load service that will provide access to Oracle's ***instantclient*** binaries. This can be configured by one of the following ways that involves setting/overriding some properties in the `values-oracle.yaml` configuration file:

1. ***nfsInstantClientHost*** and ***nfsInstantClientExportPath:*** Set values for these properties if the cluster needs to mount an NFS shared path from an NFS server.

Note: Installing the helm chart with these properties set will create a persistent volume on the cluster. As such, the user installing the helm chart should either be a cluster-admin or should have the privileges to be able to create persistent volume on the cluster.

1. **instantClientPvcName:** Set this property if the cluster needs to bind the pods to a persistent volume claim. Note that until this PVC is bound to a backing PV, the pods will not start getting created and as such, the cluster admin should ensure that the backing PV is either manually provisioned or dynamically provisioned based on the storage class associated with PVC.
2. **instantClientPvName and instantClientStorageClass:** Set these properties if the cluster needs to bind the pods to a persistent volume with the associated storage class name. Once the helm chart installation starts, a PVC will be created that is managed by the helm.

The following properties are supporting/optional properties that can be overridden along with the above properties:

1. **instantClientMountOption:** If ***nfsInstantClientHost*** and ***nfsInstantClientExportPath*** have been set, set the appropriate mount option if you would like the cluster to mount with an option other than the default option of `nfsvers=4.2.`
2. **instantClientAccessMode and instantClientStorageSize:** Persistent Volume claims can request specific storage capacity size and access modes.

(Mandatory for File Connector) Configure the source and target connector type and (optionally) the source and target volumes

UnloadFSMount and loadFSMount (earlier unloadStorageType= FS and loadStorageType **= FS):**

To use the filesystem source and target connector types, you will need to configure persistent volumes using the nfsUnloadStorage options, then uncomment and set the values of unloadFSMount and loadFSMount to true. If these values are set to true, a volume will need to be mounted, via persistent volume claims, inside the file-connector unload service that will provide access to the source file location and inside the load service that will provide access to the target file location.

UnloadHadoopMount and loadHadoopMount (earlier unloadStorageType= Hadoop and loadStorageType **= Hadoop):**

To use the Hadoop source and target connector types and configure persistent volumes using the nfsHadoopStorage options, you will need to uncomment and set the values of unloadHadoopMount and loadHadoopMount to true. If these values are set to true, a volume will need to be mounted to add the Hadoop configuration files, via persistent volume claims, inside the file-connector unload and load service that will provide access to the Hadoop configuration files.

These can be configured in one of the following ways that involves setting/overriding some properties in the `values-file-connector.yaml` configuration file:

1. **nfsUnloadStorageHost, nfsUnloadStorageExportPath, nfsLoadStorageHost, and nfsLoadStorageExportPath*:*** Set values for these properties if the cluster needs to mount an NFS shared path from an NFS server.

> **Note**: Installing the helm chart with these properties set will create a persistent volume on the cluster. As such, the user installing the helm chart should either be a cluster admin or should have the privileges to be able to create persistent volume on the cluster.

1. **unloadStoragePvcName and loadStoragePvcName:** Set these properties if the cluster needs to bind the pods to a persistent volume claim. Note that until this PVC is bound to a backing PV, the pods will not start getting created and as such, the cluster admin should ensure that the backing PV is either manually provisioned or dynamically provisioned based on the storage class associated with PVC.
2. **unloadStoragePvName, unloadStorageClass, loadStoragePvName, and loadStorageClass:** Set these properties if the cluster needs to bind the pods to a persistent volume with the associated storage class name. Once the helm chart installation starts, a PVC will be created that is managed by the helm.

The following properties are supporting/optional properties that can be overridden along with the above properties:

1. **unloadStorageMountOption and loadStorageMountOption:** If `nfsUnloadStorageHost`, `nfsUnloadStorageExportPath`, `nfsLoadStorageHost`, and `nfsLoadStorageExportPath` are configured, *s*et the appropriate mount option that you would like the cluster to use to mount the storage option. Uncomment the line for `nfsvers=4.2.`
2. **unloadStorageSize, unloadStorageAccessMode, loadStorageSize, and loadStorageAccessMode:** Persistent Volume claims can request specific storage capacity size and access modes.

Optionally, if you would like to use PySpark as the data writer type, you may configure it under the unload and load service property values by uncommenting the line and setting `dataWriterType: pyspark`.

To enable the staging push feature for unload and load services, set `skipUnloadWriters` and `skipLoadWriters` to true. Alternatively, you can provide the format file instead of using the staging push feature. To enable this option, set the `userProvidedFormatFile` to true.

> **Note:** Configurations such as dataWriterType, skipLoadWriters and userProvidedFormatFile can now be configured independently for each job using the source_configs and target_configs in job configuration.

(Optional) Configure the service database volumes

A volume will need to be mounted, via persistent volume claims, inside the pods that will provide the storage for the service databases for each hyperscale compliance service. By default, a persistent volume claim, using the default storage class, will be requested on the cluster. This can be configured, for some or all services, in one of the following ways that involves setting/overriding properties in the `values.yaml` configuration file:

1. **[service-name].dbPvcName:** Set this property if the cluster needs to bind the pods to a persistent volume claim. Note that until this PVC is bound to a backing PV, the pods will not get created and as such, the cluster admin should ensure that the backing PV is either manually provisioned or dynamically provisioned based on the storage class associated with PVC. The service database names default to `controller-db` , `unload-db` , `masking-db` and `load-db` for the controller, unload, masking and load services respectively.
2. **[service-name].databaseStorageSize:** Set this property if the cluster should request a PVC with a storage size to something other than the pre-configured size.
3. **storageClassName:** Set this property if the cluster should request a PVC using a specific storage class.

(Optional) Configure the cluster node for each service

By default, pods will be scheduled on the node(s) determined by the cluster. Set a node name under the **[service-name].nodeName** property for the service(s) if you would like to request the cluster to schedule pods on particular node(s).

(Optional) Set resource requests and limits

Some users may have default container settings as part of their Kubernetes or OpenShift infrastructure management. Sometimes, it is important to alter those settings for Hyperscale containers. You can configure resource requests and limits for each Hyperscale container like the following:

```
controller:
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
    limit:
      memory: "512Mi"
      cpu: "500m"
```

The above example is only for controller service. You can can configure properties for other services (load, unload and masking) in the same way. Note, the example above includes sample values, and you may need to contact your infrastructure team to determine these values.

Install the Helm Chart

Once the desired properties have been set/overridden, proceed to install the helm chart by running:

```
helm install hyperscale-helm <directory path of the extracted chart> -f values-
[connector-type].yaml
```

Check for the Successful Installation

After installing the helm chart, check the status of the helm chart and the pods using the following commands:

```
$ helm list
NAME              NAMESPACE    REVISION    UPDATED
STATUS      CHART                    APP VERSION
hyperscale-helm      default      1           2023-04-17 05:38:17.639357049 +0000 UTC
deployed    hyperscale-helm-18.0.0
```

```
$ kubectl get pods --namespace=hyperscale-services

NAME                                  READY   STATUS    RESTARTS   AGE

controller-service-65575b6458-2q9b4   1/1     Running   0          125m

load-service-5c644b9cc8-g9fs8         1/1     Running   0          125m

masking-service-7ddfd49c8f-5j2q5      1/1     Running   0          125m

proxy-5bd8d8f589-gkx8g                1/1     Running   0          125m

unload-service-55b5bd8cc8-7z95b       1/1     Running   0          125m
```

Creating ingress controller and ingress resource

After successfully deploying the Hyperscale Compliance services on the Kubernetes cluster, the final step involves creating an ingress route to manage external traffic to the services efficiently. For instructions, follow the steps as documented in the Ingress Setup.

MicroK8s on Linux

Installation and setup for MicroK8s on Linux distributions.

Ubuntu

1. **Install MicroK8s:**
   Open a terminal and execute the following command to install MicroK8s:

   ```
   sudo snap install microk8s --classic --channel=1.25
   ```

   a. If Snap is not installed on your system, follow the instructions here to install it.
2. **Join the MicroK8s group:**
   MicroK8s creates a user group to enable seamless command execution. Add your user to this group and set the correct permissions for the `.kube` caching directory with:

   ```
   sudo usermod -a -G microk8s $USER
   sudo chown -f -R $USER ~/.kube
   ```

   a. Re-login or restart your session to apply the group changes:

      ```
      su - $USER
      ```
3. **Check MicroK8s status:**
   Ensure MicroK8s is correctly installed and ready by checking its status:

   ```
   microk8s status --wait-ready
   ```
4. **Enable add-ons:**
   Enable essential add-ons for Hyperscale deployment:

   ```
   microk8s enable hostpath-storage
   microk8s enable helm
   microk8s enable dns
   microk8s enable ingress
   ```
5. **Create an alias for `kubectl` and `helm`**
   Facilitate command usage with an alias for `microk8s kubectl` and `microk8s helm`

   ```
   echo "alias kubectl='microk8s kubectl'" >> ~/.bash_aliases
   echo "alias helm='microk8s helm'" >> ~/.bash_aliases
   source ~/.bash_aliases
   ```

CentOS and Red Hat

For CentOS and Red Hat, the installation process diverges primarily due to the absence of `snap` by default.

1. **Enable `snapd`:**
   First, enable the EPEL repository and install `snapd`:

   ```
   sudo yum install epel-release
   sudo yum install snapd
   ```

   a. Then, start and enable `snapd`:

      ```
      sudo systemctl enable --now snapd.socket
      ```

   b. For CentOS, you may also need to enable classic `snap` support by creating a symbolic link:

      ```
      sudo ln -s /var/lib/snapd/snap /snap
      ```

2. **Install MicroK8s:**
   With `snapd` enabled, you can now install MicroK8s using `snap`:

   ```
   sudo snap install microk8s --classic --channel=1.25
   ```

3. **Group and permissions:**
   Follow the same steps as for Ubuntu to add your user to the MicroK8s group and adjust permissions for the `.kube` directory.
4. **Check status and enable add-ons:**
   Verify MicroK8s installation and enable the necessary add-ons as outlined in the Ubuntu section.
5. **Alias for `kubectl` and `helm`**
   Create and source the alias for kubectl and helm as described for Ubuntu.

Ingress setup (Kubernetes)

Ingress exposes `HTTP` and `HTTPS` routes from outside the cluster to the Hyperscale Compliance services running within the cluster. For more information, refer to the Ingress official documentation.

> The exact steps to set up an Ingress vary by Kubernetes vendor and company policies. This section provides non-exhaustive instructions for a basic setup, but you must ask your Kubernetes cluster administrator for guidance.

The proxy pod runs an Nginx HTTP server which must be the only target of the Ingress rules, redirecting all external traffic to it. Out of the box, the pod accepts requests over HTTPs on port 443, using a self-signed certificate.

After setting up an Ingress, TLS/SSL will be terminated by the HTTP server/load balancer/proxy implementing the Ingress, and not Hyperscale Compliance.

Ingress controller installation and route creation

An Ingress controller is required to continue. Expand a section below based on your Kubernetes environment to show the corresponding **Ingress controller installation** and **Ingress route creation** instructions.

Microk8s

Ingress controller installation

An ingress controller can be installed by enabling the ingress addon on the Microk8s cluster. It is enabled by running the command:

```
microk8s enable ingress
```

This addon adds an NGINX Ingress Controller for MicroK8s.

Ingress route creation

Next, define the ingress rules for routing traffic to the Hyperscale Compliance services. Create a file named `ingress.yaml` with the following configuration:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hyperscale-ingress
  namespace: hyperscale-services
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
    nginx.ingress.kubernetes.io/proxy-body-size: "50m"
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "600"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "600"
spec:
  ingressClassName: nginx
  rules:
    - http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: proxy
              port:
                 number: 443
```

This ingress configuration directs all HTTP traffic arriving at the root path (`/`) to the `proxy` service on port 443, using HTTPS as the backend protocol.

Applying the ingress configuration

With both the `ingress.yaml` files created, apply these configurations to your MicroK8s cluster using the following commands:

```
kubectl apply -f ingress.yaml
```

Alternatively, you can apply the above ingress configuration with the following single `kubectl` command:

```
kubectl create ingress hyperscale-ingress --namespace=hyperscale-services --rule="/
*=proxy:443" --annotation=nginx.ingress.kubernetes.io/backend-protocol=HTTPS --
annotation=nginx.ingress.kubernetes.io/proxy-body-size=50m --
annotation=nginx.ingress.kubernetes.io/proxy-connect-timeout=600 --
annotation=nginx.ingress.kubernetes.io/proxy-read-timeout=600 --
annotation=nginx.ingress.kubernetes.io/proxy-send-timeout=600
```

These commands register the ingress class and resource with your Kubernetes cluster, enabling the Nginx Ingress Controller to start routing external traffic to your Hyperscale Compliance services.

Amazon AWS EKS

Ingress controller installation

Please follow these instructions to install an AWS load balancer controller (An Ingress controller that configures AWS application load balancers).

Ingress route creation

Create a file named `ingress.yaml`, replacing the value of `certificate-arn` in the example below with the ARN of the certificate you want to use for the `HTTPs` endpoint.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hyperscale-ingress
  namespace: hyperscale-services
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internal
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}, {"HTTP":80}]'
    alb.ingress.kubernetes.io/ssl-redirect: '443'
    alb.ingress.kubernetes.io/backend-protocol: HTTPS
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-west-2:xxxxx:certificat
e/xxxxxxx
spec:
  rules:
    - http:
       paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: proxy
              port:
                 number: 443
```

Alternatively, you may use certificate discovery to have the ALB select a matching certificate from AWS Certificate manager based on the hostname.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hyperscale-ingress
  namespace: hyperscale-services
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internal
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}, {"HTTP":80}]'
    alb.ingress.kubernetes.io/ssl-redirect: '443'
    alb.ingress.kubernetes.io/backend-protocol: HTTPS

spec:
    tls:
  - hosts:
    - www.example.com
  rules:
    - http:
       paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: proxy
              port:
                 number: 443
```

Applying the ingress configuration

Apply the Ingress resource with `kubectl apply`:

```
kubectl apply -f ingress.yaml
```

This creates an application load balancer, which forwards all traffic to Hyperscale Compliance.

Bootstrapping API keys (Kubernetes)

Once the application starts, an API key will be generated that will be required to authenticate with the Hyperscale Compliance Orchestrator. This key will be found in the logs of the controller service pod. You can use the following command to get the API key:

```
kubectl logs <controll_service_pod_name> -n <namespace> | grep 'NEWLY GENERATED API KEY'
```

The above command displays an output similar to the following:

```
NEWLY GENERATED API KEY:
1.bTYUvuzXgnhS8U7WwYZKyF27egO1B73pJUxyw2fHAxhgVLweMIfB6LOisfA3ZGNI
```

To authenticate with the Hyperscale Compliance Orchestrator, you must use the API key and include the HTTP Authorization request header with type apk; `apk <API Key>` ;.

For more information, see the **Authentication** section under Accessing the Hyperscale Compliance API.

After acquiring the bootstrap API key, you can create a new API key for your convenience. After generating the new API key, you should override the `apiKeyCreate` property in the installed helm chart by running the following command.

```
helm upgrade <release_name> <directory path of the extracted chart> --reuse-values --set apiKeyCreate=false
```

Hyperscale logs (Kubernetes)

All Hyperscale Compliance containers log to ***stdout*** and ***stderr*** so that their logs are processed by Kubernetes. To view container-level logs running on the Kubernetes cluster, run the following command:

```
kubectl logs <pod_name> -n <namespace>
```

Limitations (Kubernetes)

This release of the Hyperscale Compliance does not support scaling up the services in a Kubernetes cluster. This support will be added in a future release.

Upgrading the Hyperscale Compliance Orchestrator (Kubernetes)

Perform the following steps to upgrade a Hyperscale Compliance Orchestrator (Kubernetes).

1. Create a new folder called `hyperscale-helm-[version]`, where [version] is the latest version to which the platform is being upgraded.
   `$ mkdir hyperscale-helm-[version]`
2. Download the new version of the chart using the following command in tandem with the newly created folder. **Note:** This command will download a file named `hyperscale-helm-[version].tgz` in the folder `hyperscale-helm-[version]`.

   ```
   $ cd hyperscale-helm-[version]
   $ curl -XGET https://dlpx-helm-hyperscale.s3.amazonaws.com/hyperscale-helm-
   [version].tgz -o hyperscale-helm-[version].tgz
   ```
3. The downloaded file is then extracted using the following command.
   `$ tar -xvf hyperscale-helm-[version].tgz`
4. This will extract into the following directory structure.

   ```
   hyperscale-helm
        ├── Chart.yaml
        ├── README.md
        ├── templates
            |-<all templates files>
        ├── tools
            |-<all tool files>
        ├── values-file-connector.yaml
        ├── values-mongo.yaml
        ├── values-mssql.yaml
        ├── values-oracle.yaml
        └── values.yaml
   ```
5. Copy the `values.yaml` and `values-<connector>.yaml` files from the previous version parallel to the `hyperscale-helm-[version]` folder.
   a. If upgrading from a Delimited/Parquet Files connector, then the image name has been changed to file-connector. As Parquet/Delimited File connectors have been merged into the File Connector, the new connector name will be `file-connector`.
6. After copying the `values.yaml` and `values-<connector>.yaml` files, there are updates that need to be made to the file under the *imageCredentials* section:
   - Bumping up the version, specified against the **tag** property**,** to the desired higher version.
   - If the credentials configured in your `values.yaml` have expired, which will be the case if the credentials were unused for 30 days or were inactive (but previously used) for 90 days, please retrieve a new set of credentials by visiting the `Hyperscale Compliance` page on http://download.delphix.com and selecting the `Hyperscale Helm Repository` link. Configure your `values.yaml` with the new set of credentials.

- This password update in `values.yaml` is only required if the user is using a Delphix-provided Docker Registry (hyperscale.download.delphix.com/delphix-hyperscale) directly in the deployment (i.e. `values.yaml`).
- If you are using your internal Docker Registry, you should first pull the next version of the Docker images from the Delphix-provided registry with the credentials associated with your Delphix account.
- The following are the 'docker' commands that can be used to pull Docker images into your internal Docker Registry:
    - Docker login command (username and password are your permanent credentials retrieved from Delphix Download site).

      ```
      $ docker login --username [USER] --password [PASSWORD]
      ```

    - Pull Docker images of the Hyperscale Compliance services.

      ```
      $ docker pull <image-url>
      ```

   The image URLs for the Hyperscale Compliance services can be referenced from this page of the documentation.

7. Provide the values for NFS/AWS S3 related parameters to setup the mount-filesystem.

> ⚠ Upon application startup, all existing mount-filsystems will be **deleted**. Please ensure to take backup of the mount setup details, if needed.

> If upgrading from a version <= 23.0.0 and the staging area volume has been configured via NFS storage, ensure that all of the following parameters have been set in the values.yaml file of the latest version's helm chart bundle:
>
> - nfsStorageHost - Set this to the same value as set in the previous version's helm chart values.
> - nfsStorageExportPath - Set this to the same value as set in the previous version's helm chart values.
> - nfsStorageMountType - Set this to one of applicable values(NFS4, NFS3, CIFS).

8. Run the helm upgrade command.
   `$ helm upgrade -f values-<connector>.yaml hyperscale-helm hyperscale-helm`

OpenShift

This section covers the following topics:

- Host requirements (OpenShift)
- Docker image registry and names for Hyperscale services (OpenShift)
- HTTPS certificate for Hyperscale (OpenShift)
- Installation and setup (OpenShift)
- Bootstrapping API keys (OpenShift)
- Hyperscale logs (OpenShift)
- Limitations (OpenShift)
- Upgrading the Hyperscale Compliance Orchestrator (OpenShift)

Host requirements (OpenShift)

Type	Host Requirement	Explanation
User	A user (example: hyperscale_os) with the following permissions is required: • Permission to run helm commands • Permission to run `oc` commands	These permissions are required to be able to install helm charts and manage the OpenShift resources.
Staging area directory permissions	The staging area directory requires the following permissions based on the UID/GID of the OS user so that the Hyperscale Compliance Orchestrator and the Continuous Compliance Engine(s) can perform read/write/execute operations on the staging area: • If the Hyperscale Compliance OS user has a UID of 65436, then the staging area directory must have a UID of 65436 and 700 permission mode. • If the Hyperscale Compliance OS user has a GID of 50 and does not have a UID of 65436, then the staging area directory must have a GID of 50 and 770 permission mode.	These permissions on the staging area directory are required for the Hyperscale Compliance containers and the continuous compliance engines to be able to read/write into the staging area.
Installation Directory	A directory to download and manage helm charts.	
Hardware Requirements	**Minimum:** 8 vCPU, 64 GB of memory, 50GB OS disk. **Recommended:** 16 vCPU, 128GB of memory, 50GB OS disk.	

Docker image registry and names for Hyperscale services (OpenShift)

The docker images for Hyperscale Compliance services fall under two categories of services:

1. **Common services**: A set of service images common across deployments with varying unload/load services.
2. **Unload and Load services**: A set of unload and load service images specific to a dataset vendor.

All images are stored under the hyperscale.download.delphix.com/delphix-hyperscale registry. Also, any image will have a name of the form `<image-name>-[VERSION]` where [VERSION] corresponds to a particular release version.

The following list provides the image URLs for the docker images in the two categories of services:

Common services

```
hyperscale.download.delphix.com/delphix-hyperscale:proxy-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:controller-service-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:masking-service-[VERSION]
```

Unload and Load services

Oracle Unload and Load

```
hyperscale.download.delphix.com/delphix-hyperscale:oracle-unload-service-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:oracle-load-service-[VERSION]
```

MSSQL Unload and Load

```
hyperscale.download.delphix.com/delphix-hyperscale:mssql-unload-service-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:mssql-load-service-[VERSION]
```

File Connector Unload and Load

```
hyperscale.download.delphix.com/delphix-hyperscale:file-connector-unload-service-
[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:file-connector-load-service-
[VERSION]
```

Mongo Unload and Load

```
hyperscale.download.delphix.com/delphix-hyperscale:mongo-unload-service-[VERSION]
hyperscale.download.delphix.com/delphix-hyperscale:mongo-load-service-[VERSION]
```

HTTPS certificate for Hyperscale (OpenShift)

By default, Hyperscale creates a unique self-signed certificate to enable HTTPS when starting for the first time. This certificate and private key are configured in the `values.yaml` file under:

```
proxy:
 crt:<certificate_value>
 key:<private_key_value>
```

To use your own certificates, these default values need to be replaced. They are Base64 encoded values of the certificate and key respectively.

> You may need to contact your IT team to get the corresponding certificate files.

- To generate the Base64 encoded value of the certificate, run the following command:

  ```
  cat my_cert_file.cert | base64 -w 0
  ```

- To generate the Base64 encoded value of the key, run the following command:

  ```
  cat my_private_key.key | base64 -w 0
  ```

After changing the ***crt*** and ***key*** values in `values.yaml` file, run the HELM upgrade and restart the proxy service. Run the following command to restart proxy service:

```
kubectl rollout restart deployment proxy -n <hyperscale-namespace>
```

Installation and setup (OpenShift)

Installation requirements

To deploy Hyperscale Compliance via Openshift, you will require a running OpenShift cluster, the `oc` command line tool to interact with the OpenShift cluster, and HELM for deployment onto the cluster.

Requirement	Recommended Version	Comments
oc	4.11.3 or above	
HELM	3.9.0 or above	HELM installation should support HELM v3. More information on HELM can be found at `https://helm.sh/docs/`. To install HELM, follow the installation instructions at `https://helm.sh/docs/intro/install/`. The installation also requires access to the HELM repository from where Hyperscale charts can be downloaded. The HELM repository URL is `https://dlpx-helm-hyperscale.s3.amazonaws.com`.
OpenShift Cluster	4.12 or above	

- If an intermediate HELM repository is to be used instead of the default Delphix HELM repository, then the repository URL, username, and password to access this repository needs to be configured in the `values.yaml` file under the **imageCredentials** section.
- Oracle Load doesn’t support Object Identifiers(OIDs).

Installation process

OC login

Run the OC login command to authenticate OpenShift CLI with the server:

```
oc login https://openshift1.example.com -u=<<user_name>> -p=<<password>>
```

Verify KubeConfig

HELM will use the configuration file inside the `$HOME/.kube/` folder to deploy artifacts on an OpenShift cluster. Be sure the config file has the cluster context added, and the current context is set to use this cluster. To verify the context, run this command:

```
oc config current-context
```

Create a new project

Create a new project named hyperscale-services using the command below:

```
oc new-project hyperscale-services --description="Hyperscale Deployment project" --
display-name="hyperscale-services"
```

Define SecurityContextConstraints

Hyperscale Compliance services by default run with UID: 65436 GID: 50, so that the files created by Hyperscale Compliance can be read by the Hyperscale Compliance Engine (and vise-versa) which runs UID: 65436 GID: 50.

The default SecurityContextConstraints (SCC) in Openshift makes all hyperscale services run as random UID/GID that breaks the arrangement that Hyperscale has with the Compliance Engine. To make Hyperscale work with a Compliance Engine, you need to create custom SecurityContextConstraints (SCC).

Here are the (sample) steps to achieve the same. You must perform these tasks only once for a deployment setup. If these steps have already been executed, use the ServiceAccount in `values.yaml` file.

OC login

Run the OC login command (by providing the administrator username and password) to authenticate OpenShift CLI with the server:

```
oc login https://openshift1.example.com -u=<<user_name>> -p=<<password>>
```

Create SecurityContextConstraints

Create a file (e.g `hs-scc.yaml`) with the following content:

```
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: <SecurityContextConstraints-Name>
allowPrivilegedContainer: false
runAsUser:
  type: MustRunAs
  uid: 65436
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: MustRunAs
  ranges:
    - min: 50
      max: 50
```

Apply the configuration to create SecurityContextConstraint.

```
oc apply -f hs-ssc.yaml
```

Create ServiceAccount

Run the following command to create a service account.

```
oc create sa <service-account-name>
```

Create Role and Role Binding

Create a file (e.g. `hs-role.yaml`) with the following content.

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: hs-role-scc
rules:
  - apiGroups: ["security.openshift.io"]
    resources: ["securitycontextconstraints"]
    resourceNames: ["<SecurityContextConstraints-Name>"]
    verbs: ["use"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: hs-rb-scc
subjects:
  - kind: ServiceAccount
    name: <service-account-name>
roleRef:
  kind: Role
  name: hs-role-scc
  apiGroup: rbac.authorization.k8s.io
```

Replace <SecurityContextConstraints-Name> and <service-account-name> in the file with the names used in the previous step. Use the following command to apply the configuration to create role and role binding.

```
oc apply -f hs-role.yaml
```

Installation

Download the HELM charts

The latest version of the chart can be pulled locally with the following command (where `x.x.x` should be changed to the version of Hyperscale being installed):

```
curl -XGET https://dlpx-helm-hyperscale.s3.amazonaws.com/hyperscale-helm-x.x.x.tgz -o hyperscale-helm-x.x.x.tgz
```

This command will download a file with the name `hyperscale-helm-x.x.x.tgz` in the current working directory. The downloaded file can be extracted using the following command (where `x.x.x` should be changed to the version of Hyperscale being installed):

`tar -xvf hyperscale-helm-x.x.x.tgz`

This will extract into the following directory structure:

```
hyperscale-helm
    ├── Chart.yaml
    ├── README.md
    ├── templates
        |-<all templates files>
    ├── tools
        |-<all tool files>
    ├── values-file-connector.yaml
    ├── values-mongo.yaml
    ├── values-mssql.yaml
    ├── values-oracle.yaml
    └── values.yaml
```

Verify the authenticity of the downloaded HELM charts

The SHA-256 hash sum of the downloaded helm chart tarball file can be verified as follows:

1. Execute the below command and note the digest value for version x.x.x (where `x.x.x` should be changed to the version of Hyperscale being installed)
   curl `https://dlpx-helm-hyperscale.s3.amazonaws.com/index.yaml`
2. Execute the sha256sum command (or equivalent) on the downloaded file (where `x.x.x` should be changed to the version of Hyperscale being installed) (hyperscale-helm-x.x.x.tgz)
   `sha256sum hyperscale-helm-x.x.x.tgz`

The value generated by the sha256sum utility in step 2 must match the digest value noted in step 1.

Configure Registry Credentials for Docker Images

For pulling the Docker images from the registry, permanent credentials associated with your Delphix account would need to be configured in the `values.yaml` file. To get these permanent credentials, visit the Hyperscale Compliance Download page and log in with your credentials. Once logged in, select the Hyperscale HELM Repository link and accept the Terms and Conditions. Once accepted, credentials for the docker image registry will be presented. Note them down and edit the `imageCredentials.username` and `imageCredentials.password` properties in the `values.yaml` file as shown below:

```
# Credentials to fetch Docker images from Delphix internal repository
      imageCredentials:
# Username to login to docker registry
      username: <username>
# Password to login to docker registry
      password: <password>
```

> Delphix will delete unused credentials after 30 days and inactive (but previously used) credentials after 90 days.

Helm chart configuration files

`hyperscale-helm` is the name of the folder that was extracted in the previous step. In the above directory structure, there are essentially two files that come into play while attempting to install the helm chart:

1. A `values.yaml` configuration file that contains configurable properties, common to all the services, with their default values.
2. A `values-[connector-type].yaml` configuration file that contains configurable properties, applicable to the services of the specific connector, with their default values.

The following sections talk about some of the important properties that will need to be configured correctly for a successful deployment. A full list of the configurable properties can be found on the Configuration Settings page.

(Mandatory) Configure the staging area volume

A volume will need to be mounted, via persistent volume claims, inside the pods that will provide access to the staging area for the hyperscale compliance services. This can be configured in one of the following ways that involves setting/overriding some properties in the `values.yaml` configuration file:

1. ***nfsStorageHost*** and ***nfsStorageExportPath:*** Set values for these properties if the cluster needs to mount an NFS shared path from an NFS server. For information about setting up and configuring an NFS server for the staging area volume, refer to NFS Server Installation.

> - Installing the helm chart with these properties set will create a persistent volume on the cluster. As such, the user installing the helm chart should either be a cluster-admin or should have the privileges to be able to create persistent volume on the cluster.
> - The above parameters are also used to auto-configure the mount-filesystem. Hence, the value for `nfsStorageMountType` property must also be defined.

2. **stagePvcName:** Set this property if the cluster needs to bind the pods to a persistent volume claim. Note that until this PVC is bound to a backing PV, the pods will not start getting created and as such, the cluster admin should ensure that the backing PV is either statically provisioned or dynamically provisioned based on the storage class associated with PVC.
3. **stagePvName and stageStorageClass:** Set these properties if the cluster needs to bind the pods to a persistent volume with the associated storage class name. Once the helm chart installation starts, a PVC will be created that is managed by the helm.

The following properties are supporting/optional properties that can be overridden along with the above properties:

1. **nfsStorageMountOption:** If ***nfsStorageHost*** and ***nfsStorageExportPath*** have been set, set the appropriate mount option if you would like the cluster to mount with an option other than the default option of `nfsvers=4.2.`
2. **stageAccessMode and stageStorageSize:** Persistent Volume claims can request specific storage capacity size and access modes.

(Mandatory for Oracle) Configure the instantclient volume

A volume will need to be mounted, via persistent volume claims, inside the Oracle load service that will provide access to Oracle's ***instantclient*** binaries. This can be configured by one of the following ways that involves setting/overriding some properties in the `values-oracle.yaml` configuration file:

1. ***nfsInstantClientHost*** and ***nfsInstantClientExportPath:*** Set values for these properties if the cluster needs to mount an NFS shared path from an NFS server.

> **Note**: Installing the helm chart with these properties set will create a persistent volume on the cluster. As such, the user installing the helm chart should either be a cluster-admin or should have the privileges to be able to create persistent volume on the cluster.

1. **instantClientPvcName:** Set this property if the cluster needs to bind the pods to a persistent volume claim. Note that until this PVC is bound to a backing PV, the pods will not start getting created and as such, the cluster admin should ensure that the backing PV is either manually provisioned or dynamically provisioned based on the storage class associated with PVC.
2. **instantClientPvName and instantClientStorageClass:** Set these properties if the cluster needs to bind the pods to a persistent volume with the associated storage class name. Once the helm chart installation starts, a PVC will be created that is managed by the helm.

The following properties are supporting/optional properties that can be overridden along with the above properties:

1. **instantClientMountOption:** If ***nfsInstantClientHost*** and ***nfsInstantClientExportPath*** have been set, set the appropriate mount option if you would like the cluster to mount with an option other than the default option of `nfsvers=4.2.`
2. **instantClientAccessMode and instantClientStorageSize:** Persistent Volume claims can request specific storage capacity size and access modes.

(Mandatory for File Connector) Configure the source and target connector type and (optionally) the source and target volumes

UnloadFSMount and loadFSMount (earlier unloadStorageType= FS and loadStorageType **= FS):**

To use the filesystem source and target connector types, you will need to configure persistent volumes using the nfsUnloadStorage options, then uncomment and set the values of unloadFSMount and loadFSMount to true. If these values are set to true, a volume will need to be mounted, via persistent volume claims, inside the file-connector unload service that will provide access to the source file location and inside the load service that will provide access to the target file location.

UnloadHadoopMount and loadHadoopMount (earlier unloadStorageType= Hadoop and loadStorageType **= Hadoop):**

To use the Hadoop source and target connector types and configure persistent volumes using the nfsHadoopStorage options, you will need to uncomment and set the values of unloadHadoopMount and loadHadoopMount to true. If these values are set to true, a volume will need to be mounted to add the Hadoop configuration files, via persistent volume claims, inside the file-connector unload and load service that will provide access to the Hadoop configuration files.

These can be configured in one of the following ways that involves setting/overriding some properties in the `values-file-connector.yaml` configuration file:

1. **nfsUnloadStorageHost, nfsUnloadStorageExportPath, nfsLoadStorageHost, and nfsLoadStorageExportPath*:*** Set values for these properties if the cluster needs to mount an NFS shared path from an NFS server.

Note: Installing the helm chart with these properties set will create a persistent volume on the cluster. As such, the user installing the helm chart should either be a cluster admin or should have the privileges to be able to create persistent volume on the cluster.

1. **unloadStoragePvcName and loadStoragePvcName:** Set these properties if the cluster needs to bind the pods to a persistent volume claim. Note that until this PVC is bound to a backing PV, the pods will not start getting created and as such, the cluster admin should ensure that the backing PV is either manually provisioned or dynamically provisioned based on the storage class associated with PVC.
2. **unloadStoragePvName, unloadStorageClass, loadStoragePvName, and loadStorageClass:** Set these properties if the cluster needs to bind the pods to a persistent volume with the associated storage class name. Once the helm chart installation starts, a PVC will be created that is managed by the helm.

The following properties are supporting/optional properties that can be overridden along with the above properties:

1. **unloadStorageMountOption and loadStorageMountOption:** If `nfsUnloadStorageHost`, `nfsUnloadStorageExportPath`, `nfsLoadStorageHost`, and `nfsLoadStorageExportPath` are configured, *s*et the appropriate mount option that you would like the cluster to use to mount the storage option. Uncomment the line for `nfsvers=4.2.`
2. **unloadStorageSize, unloadStorageAccessMode, loadStorageSize, and loadStorageAccessMode:** Persistent Volume claims can request specific storage capacity size and access modes.

Optionally, if you would like to use PySpark as the data writer type, you may configure it under the unload and load service property values by uncommenting the line and setting `dataWriterType: pyspark`.

To enable the staging push feature for unload and load services, set `skipUnloadWriters` and `skipLoadWriters` to true. Alternatively, you can provide the format file instead of using the staging push feature. To enable this option, set the `userProvidedFormatFile` to true.

Note: Configurations such as dataWriterType, skipLoadWriters and userProvidedFormatFile can now be configured independently for each job using the source_configs and target_configs in job configuration.

(Optional) Configure the service database volumes

A volume will need to be mounted, via persistent volume claims, inside the pods that will provide the storage for the service databases for each hyperscale compliance service. By default, a persistent volume claim, using the default storage class, will be requested on the cluster. This can be configured, for some or all services, in one of the following ways that involves setting/overriding properties in the `values.yaml` configuration file:

1. **[service-name].dbPvcName:** Set this property if the cluster needs to bind the pods to a persistent volume claim. Note that until this PVC is bound to a backing PV, the pods will not get created and as such, the cluster admin should ensure that the backing PV is either manually provisioned or dynamically provisioned based on the storage class associated with PVC. The service database names default to `controller-db`, `unload-db`, `masking-db` and `load-db` for the controller, unload, masking and load services respectively.
2. **[service-name].databaseStorageSize:** Set this property if the cluster should request a PVC with a storage size to something other than the pre-configured size.
3. **storageClassName:** Set this property if the cluster should request a PVC using a specific storage class.

(Optional) Configure the cluster node for each service

By default, pods will be scheduled on the node(s) determined by the cluster. Set a node name under the **[service-name].nodeName** property for the service(s) if you would like to request the cluster to schedule pods on particular node(s).

Enable Openshift specific properties

- Update **isOpenShift: true** for Openshift
- **serviceAccountName:** <service-account-name>. This ServiceAccount is created as described in the above section **Define SecurityContextConstraints**.

(Optional) Set resource requests and limits

Some users may have default container settings as part of their Kubernetes or OpenShift infrastructure management. Sometimes, it is important to alter those settings for Hyperscale containers. You can configure resource requests and limits for each Hyperscale container like the following:

```
controller:
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
    limit:
      memory: "512Mi"
      cpu: "500m"
```

The above example is only for controller service. You can configure properties for other services (load, unload and masking) in the same way. Please note that the example above includes sample values, user may need to connect with their infrastructure team to decide these values.

Install the Helm Chart

Once the desired properties have been set/overridden, proceed to install the helm chart by running:

```
helm install hyperscale-helm <directory path of the extracted chart> -f values-[connector-type].yaml
```

Check for the Successful Installation

After installing the helm chart and setting up the ingress controller, check the status of the helm chart and the pods using the following commands:

```
$ helm list
NAME              NAMESPACE    REVISION    UPDATED
STATUS      CHART                    APP VERSION
hyperscale-helm      default      1           2023-04-17 05:38:17.639357049 +0000 UTC
deployed    hyperscale-helm-18.0.0
```

```
$ kubectl get pods --namespace=hyperscale-services

NAME                                  READY   STATUS    RESTARTS   AGE

controller-service-65575b6458-2q9b4   1/1     Running   0          125m

load-service-5c644b9cc8-g9fs8         1/1     Running   0          125m

masking-service-7ddfd49c8f-5j2q5      1/1     Running   0          125m

proxy-5bd8d8f589-gkx8g                1/1     Running   0          125m

unload-service-55b5bd8cc8-7z95b       1/1     Running   0          125m
```

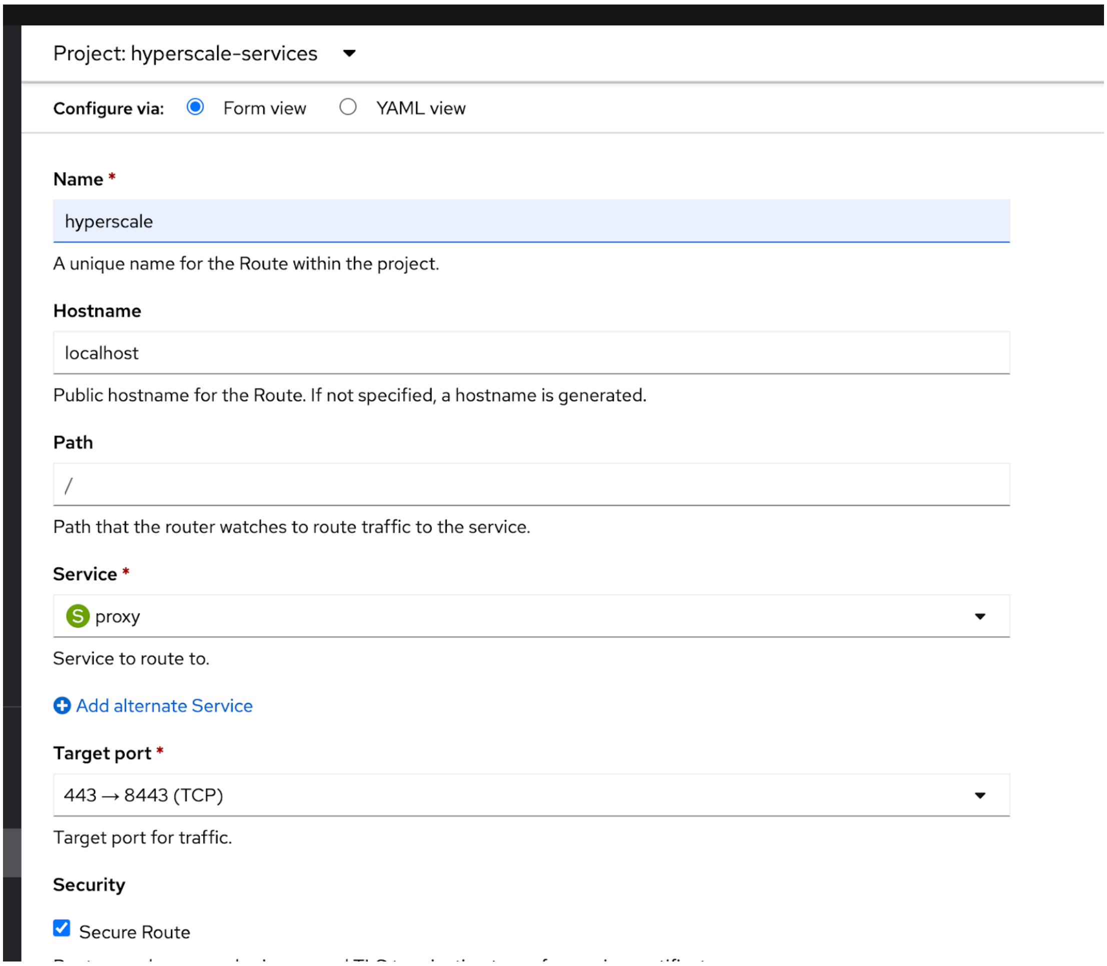
Configure Ingress

Hyperscale Compliance only works with HTTPS Ingress. It does not support HTTP.

Creating route

To create a route, you can use the OpenShift console and create a new one for the Hyperscale service.

1. Go to **Network** > **Route** > **Create Route**.
2. Provide the details as shown in the following screenshot.



Project: hyperscale-services ▾

Configure via: ● Form view ○ YAML view

Name *

hyperscale

A unique name for the Route within the project.

Hostname

localhost

Public hostname for the Route. If not specified, a hostname is generated.

Path

/

Path that the router watches to route traffic to the service.

Service *

S proxy ▾

Service to route to.

⊕ Add alternate Service

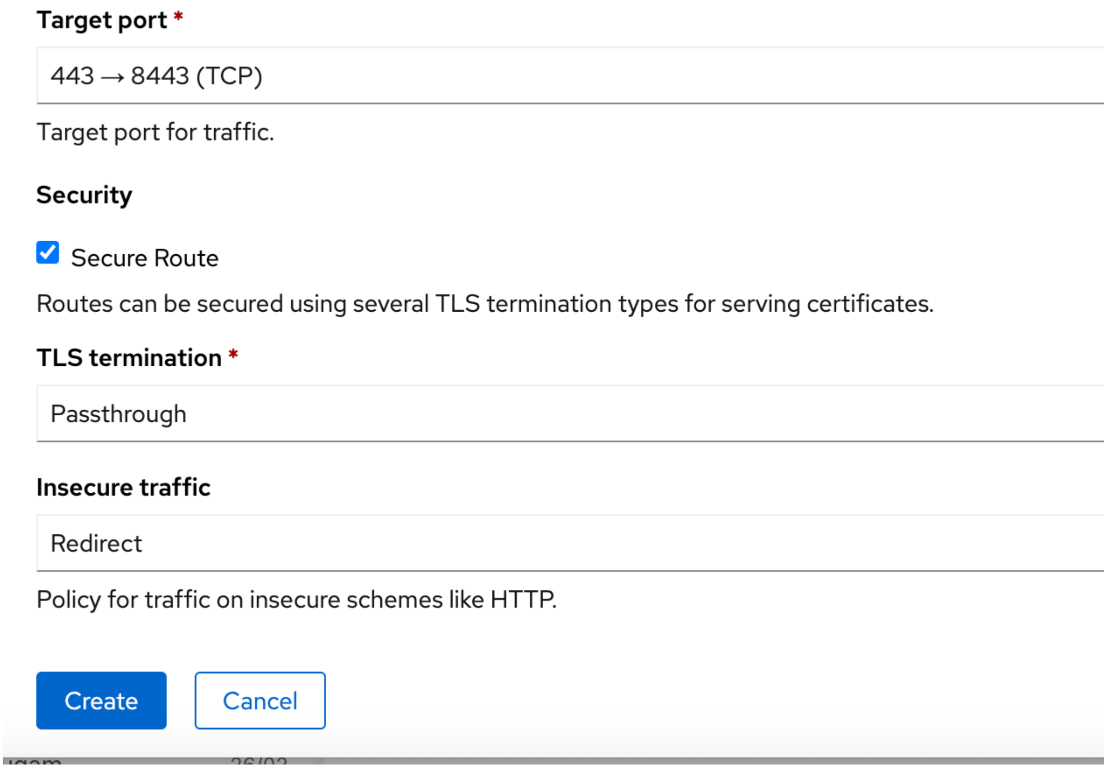
Target port *

443 → 8443 (TCP) ▾

Target port for traffic.

Security

☑ Secure Route



Target port *

443 → 8443 (TCP)

Target port for traffic.

Security

☑ Secure Route

Routes can be secured using several TLS termination types for serving certificates.

TLS termination *

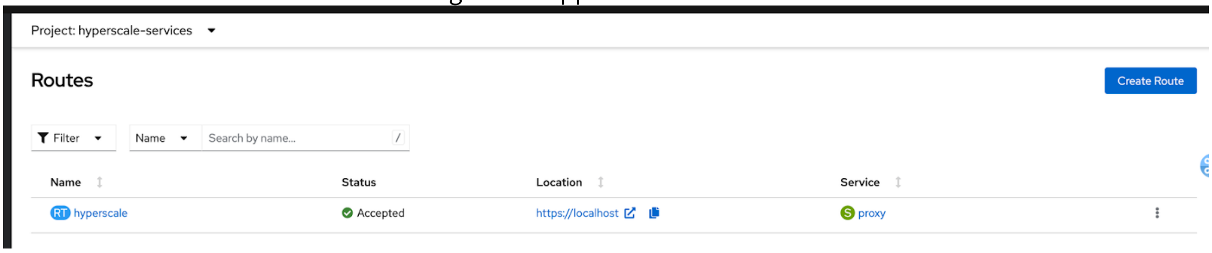
Passthrough

Insecure traffic

Redirect

Policy for traffic on insecure schemes like HTTP.

3. Click on the **Create** button. The following screen appears.



Project: hyperscale-services

Routes

Name	Status	Location	Service
RT hyperscale	✓ Accepted	https://localhost	S proxy

4. Click on the URL in the location column to access hyperscale.

Bootstrapping API keys (OpenShift)

Once the application starts, an API key will be generated that will be required to authenticate with the Hyperscale Compliance Orchestrator. This key will be found in the logs of the controller service pod. You can use the following command to get the API key:

```
oc logs <controll_service_pod_name> -n <namespace> | grep 'NEWLY GENERATED API KEY'
```

The above command displays an output similar to the following:

```
NEWLY GENERATED API KEY:
1.bTYUvuzXgnhS8U7WwYZKyF27egO1B73pJUxyw2fHAxhgVLweMIfB6LOisfA3ZGNI
```

To authenticate with the Hyperscale Compliance Orchestrator, you must use the API key and include the HTTP Authorization request header with type apk; `apk <API Key>` ;.

For more information, see the **Authentication** section under Accessing the Hyperscale Compliance API.

After acquiring the bootstrap API key, you can create a new API key for your convenience. After generating the new API key, you should override the `apiKeyCreate` property in the installed helm chart by running the following command.

```
helm upgrade <release_name> <directory path of the extracted chart> --reuse-values --set apiKeyCreate=false
```

Hyperscale logs (OpenShift)

All Hyperscale Compliance containers log to ***stdout*** and ***stderr*** so that their logs are processed by OpenShift. To view container-level logs running on the OpenShift cluster, run the following command:

```
oc logs <pod_name>
```

Limitations (OpenShift)

This release of the Hyperscale Compliance does not support scaling up the services in a Kubernetes/OpenShift cluster. This support will be added in a future release.

Upgrading the Hyperscale Compliance Orchestrator (OpenShift)

Perform the following steps to upgrade a Hyperscale Compliance Orchestrator (OpenShift).

1. Create a new folder called `hyperscale-helm-[version]` , where [version] is the latest version to which the platform is being upgraded.
   `$ mkdir hyperscale-helm-[version]`
2. Download the new version of the chart using the following command in tandem with the newly created folder. **Note:** This command will download a file named `hyperscale-helm-[version].tgz` in the folder `hyperscale-helm-[version]` .

   ```
   $ cd hyperscale-helm-[version]
   $ curl -XGET https://dlpx-helm-hyperscale.s3.amazonaws.com/hyperscale-helm-
   [version].tgz -o hyperscale-helm-[version].tgz
   ```
3. The downloaded file is then extracted using the following command.
   `$ tar -xvf hyperscale-helm-[version].tgz`
4. This will extract into the following directory structure.

   ```
   hyperscale-helm
        ├── Chart.yaml
        ├── README.md
        ├── templates
            |-<all templates files>
        ├── tools
            |-<all tool files>
        ├── values-file-connector.yaml
        ├── values-mongo.yaml
        ├── values-mssql.yaml
        ├── values-oracle.yaml
        └── values.yaml
   ```
5. Copy the `values.yaml` and `values-<connector>.yaml` files from the previous version parallel to the `hyperscale-helm-[version]` folder.
   a. If upgrading from a Delimited/Parquet Files connector, then the image name has been changed to file-connector. As Parquet/Delimited File connectors have been merged into the File Connector, the new connector name will be `file-connector` .
6. After copying the `values.yaml` and `values-<connector>.yaml` files, there are updates that need to be made to the file under the *imageCredentials* section:
   - Bumping up the version, specified against the **tag** property**,** to the desired higher version.
   - If the credentials configured in your `values.yaml` have expired, which will be the case if the credentials were unused for 30 days or were inactive (but previously used) for 90 days, please retrieve a new set of credentials by visiting the `Hyperscale Compliance` page on http://download.delphix.com and selecting the `Hyperscale Helm Repository` link. Configure your `values.yaml` with the new set of credentials.

- This password update in `values.yaml` is only required if the user is using a Delphix-provided Docker Registry (hyperscale.download.delphix.com/delphix-hyperscale) directly in the deployment (i.e. `values.yaml`).
- If you are using your internal Docker Registry, you should first pull the next version of the Docker images from the Delphix-provided registry with the credentials associated with your Delphix account.
- The following are the ‘docker’ commands that can be used to pull Docker images into your internal Docker Registry:
    - Docker login command (username and password are your permanent credentials retrieved from Delphix Download site).

      ```
      $ docker login --username [USER] --password [PASSWORD]
      ```

    - Pull Docker images of the Hyperscale Compliance services.

      ```
      $ docker pull <image-url>
      ```

   The image URLs for the Hyperscale Compliance services can be referenced from this page of the documentation.

7. Provide the values for NFS/AWS S3 related parameters to setup the mount-filesystem.

> Upon application startup, all existing mount-filsystems will be **deleted**. Please ensure to take backup of the mount setup details, if needed.

> If upgrading from a version <= 23.0.0 and the staging area volume has been configured via NFS storage, ensure that all of the following parameters have been set in the values.yaml file of the latest version's helm chart bundle:
>
> - nfsStorageHost - Set this to the same value as set in the previous version's helm chart values.
> - nfsStorageExportPath - Set this to the same value as set in the previous version's helm chart values.
> - nfsStorageMountType - Set this to one of applicable values(NFS4, NFS3, CIFS).

8. Run the helm upgrade command.
   `$ helm upgrade -f values-<connector>.yaml hyperscale-helm hyperscale-helm`

Podman Compose

⚠ Delphix highly recommends new installations be performed on Kubernetes.

This section covers the following topics:

- Host requirements (Podman Compose)
- Installation and Setup (Podman Compose)
- Upgrading the Hyperscale Compliance Orchestrator (Podman Compose)

Host requirements (Podman Compose)

Type	Host Requirement	Explanation
User	A user (hyperscale_os) with the following permissions are required: • Should have permissions to install `podman` and `podman-compose`. • Permission to run mount, unmount, mkdir and rmdir as a super-user with NOPASSWD. • sudo permission to run `sudo sysctl net.ipv4.ip_unprivileged_port_start=80` • Should have either GID=50 and/or UID=65436.	This will be a primary user responsible to install and operate the Hyperscale Compliance.
Installation Directory	There must be a directory on the Hyperscale Compliance Orchestrator host where the Hyperscale Compliance can be installed.	This is a directory where the Hyperscale Compliance tar archive file will be placed and extracted. The extracted artifacts will include docker images(tar archive files) and a configuration file(podman-compose.yaml) that will be used to install the Hyperscale Compliance.
Log File Directory	An optional directory to place log files.	This directory (can be configured via podman-compose.yaml configuration file) will host the runtime/log files of the Hyperscale Compliance Orchestrator.
NFS Client Services	NFS client services must be enabled on the host.	NFS client service is required to be able to mount an NFS shared storage from where the Hyperscale Compliance Orchestrator will be able to read the source files and write the target files. For more information, see NFS Server Installation.
Hardware Requirements	• Minimum: 8 vCPU, 64 GB of memory, 100GB data disk. • Recommended: 16 vCPU, 128GB of memory, 500GB data disk.	OS disk space: 50 GB

Installation and setup (Podman Compose)

⚠ Delphix highly recommends new installations be performed on Kubernetes.

This section describes the steps you must perform to install the Hyperscale Compliance Orchestrator.

Hyperscale Compliance installation

Pre-requisites

Ensure that you meet the following requirements before you install the Hyperscale Compliance Orchestrator.

- Download the Hyperscale tar file (`delphix-hyperscale-masking-x.0.0.tar.gz`) from download.delphix.com (where `x.0.0` should be changed to the version of Hyperscale being installed).
- You must create a user that has permission to install Podman and Podman Compose.
- Install Podman on VM. The minimum supported podman version is 4.4.1.

Note: By default, the Podman 4.4.1 version is not available for a few debian-based Linux distributions. For example, you cannot install Podman 4.4.1 on Ubuntu 20.04 and above. The workaround is to use a RHEL machine to host the hyperscale deployment with Podman.

- Install Podman Compose on the VM. The minimum supported podman-compose version is 1.0.6
- Check if podman and podman-compose are installed by running the following command:
  - `podman-compose -v` The above command displays an output similar to the following: `podman-compose version 1.0.6`
  - `podman -v` The above command displays an output similar to the following: `podman version 4.4.1`
- Podman can not create containers that bind to ports < 1024 (here). Hyperscale's proxy container binds port 80, 443. Run following command to enable binding or port : `sudo sysctl net.ipv4.ip_unprivileged_port_start=80`
- [Only Required for Oracle Load Service] Download and install Linux-based Oracle's instant client on the machine where the Hyperscale Compliance Orchestrator will be installed. The client should essentially include `instantclient-basic` (Oracle shared libraries) along with `instantclient-tools` containing `Oracle's SQL*Loader` client. Both the packages instantclient-basic and instantclient-tools should be unzipped in the same directory. A group ownership id of 50 with a permission mode of 550 or a user id of 65436 with a permission mode of 500 must be set recursively on the directory where Oracle's instant client `binaries/libraries` will be installed. This is required by the Hyperscale Compliance Orchestrator to be able to read or execute from the directory.

ℹ Oracle Load doesn't support Object Identifiers(OIDs).

Procedure

Perform the following procedure to install the Hyperscale Compliance Orchestrator.

1. Unpack the Hyperscale tar file (where `x.0.0` should be changed to the version of Hyperscale being installed).

```
tar -xzf delphix-hyperscale-masking-x.0.0.tar.gz
```

2. Upon unpacking, you will find the podman image tar files which are categorized as below:

Universal images common for all connectors.

- `controller-service.tar`
- `masking-service.tar`
- `proxy.tar`

Oracle (required only for Oracle data source masking)

- `unload-service.tar`
- `load-service.tar`

MSSQL (required only for MS SQL data source masking)

- `mssql-unload-service.tar`
- `mssql-load-service.tar`

File Connector (required only for Delimited and Parquet Files masking)

- `file-connector-unload-service.tar`
- `file-connector-load-service.tar`

MongoDB (required only for MongoDB database masking)

- `mongo-unload-service.tar`
- `mongo-load-service.tar`

Each deployment set consists of 5 images (3 Universal images and 2 images related to each dataset type). Proceed to load the required images into podman as below:

For Oracle data source masking:

```
podman load --input unload-service.tar
podman load --input load-service.tar
podman load --input controller-service.tar
podman load --input masking-service.tar
podman load --input proxy.tar
```

For MS SQL data source masking:

```
podman load --input mssql-unload-service.tar
podman load --input mssql-load-service.tar
podman load --input controller-service.tar
podman load --input masking-service.tar
podman load --input proxy.tar
```

For Delimited and Parquet Files masking using File connector:

```
podman load --input file-connector-unload-service.tar
podman load --input file-connector-load-service.tar
podman load --input controller-service.tar
podman load --input masking-service.tar
podman load --input proxy.tar
```

For MongoDB data source masking:

```
podman load --input mongo-unload-service.tar
podman load --input mongo-load-service.tar
podman load --input controller-service.tar
podman load --input masking-service.tar
podman load --input proxy.tar
```

3. Create an NFS shared mount, that will act as a **Staging Area**, on the Hyperscale Compliance Orchestrator host where the Hyperscale Compliance Orchestrator will perform read/write/execute operations.
    a. Create a ‘Staging Area’ directory. For example: `/mnt/provision`. Note that there is no more need to explicitly create a `mount-name` directory as required in previous releases. The shared path from the NFS server shall be the same as the value defined for config `NFS_STORAGE_EXPORT_PATH`. The user(s) within each of the docker containers part of the Hyperscale Compliance Orchestrator and the appliance OS user(s) in the Continuous Compliance Engine(s), all have the user id as 65436 and/or group ownership id as 50. As such, the directory `provision`, require the following permissions, based on the UID/GID of the OS user, so that the Hyperscale Compliance Orchestrator and the Continuous Compliance Engine(s) can perform read/write/execute operations on the staging area:
        i. If the Hyperscale Compliance OS user has a UID of 65436, then the directory `provision`, must have a UID of 65436 and 700 permission mode.
        ii. If the Hyperscale Compliance OS user has a GID of 50 and does not have a UID of 65436, then the directory `provision`, must have a GID of 50 and 770 permission mode.
    b. Mount the NFS shared directory on the staging area directory(`/mnt/provision`). This NFS shared storage can be created and mounted in two ways as detailed in the NFS Server Installation section. Based on the umask value for the user which is used to mount, the permissions for the staging area directory could get altered after the NFS share has been mounted. In such cases, the permissions(i.e. 770 or 700 whichever applies based on point 3a) must be applied again on the staging area directory.
4. Configure the following container volume bindings for the containers by editing the `podman-compose.yaml` file from tar:
    a. For each of the containers, except the ‘proxy’ container, add a volume entry binding the staging area path (from 3(a), `/mnt/hyperscale`) to the Hyperscale Compliance Orchestrator container path(`/etc/hyperscale`) as a volume binding under the ‘volumes’ section.
    b. [Only Required for Oracle Load Service] For the **load-service** container, add a volume entry that binds the path of the directory on the host where both the Oracle instant Client packages were unzipped to the path on the container (`/usr/lib/instantclient`) under the ‘volumes’ section.
    c. [Only Required for File Connector Unload Service] For File connector unload-service, the source NFS location has to be mounted to the container as volume in order for it to access the source files. The path mounted on the container is passed during the creation of the source connector-info.

```
# Mount your source NFS location onto your Hyperscale Engine server
sudo mount [-t nfs4] <source_nfs_endpoint>:<source_nfs_location>
<nfs_mount_on_host>
```

```
# Later mount <nfs_mount_on_host> as a podman volume to the file-connector
unload-service container (in podman-compose.yaml, created using podman-
compose-delimitedfiles-sample.yaml)
unload-service:
  image: delphix-file-connector-unload-service-app:<HYPERSCALE VERSION>
     ...
     volumes:
          ...
     # Source files should be made available within the unload-service
container file system
     # The paths within the container should be configured in the source
section of connector-info [with type=FS]
     -
<nfs_mount_on_host>:<source_files_mount_passed_during_connector_info_creat
ion_path_in_contianer>
```

d. [Only Required for File Connector load Service] For File Connector load-service, the target NFS location has to be mounted to the container as volume in order for it to access the target location where masked files will be placed. The path mounted on the container is passed during the creation of the target connector-info.

```
# Mount your target NFS location onto your Hyperscale Engine server
sudo mount [-t nfs4] <target_nfs_endpoint>:<target_nfs_location>
<target_nfs_mount_on_host>
```

```
# Later mount <nfs_mount_on_host> as a podman volume to the file-connector
load-service container (in podman-compose.yaml, created using podman-
compose-delimitedfiles-sample.yaml)
load-service:
  image: delphix-file-connector-load-service-app:${VERSION}
     ...
     volumes:
          ...
     # Target location should be made available within the load-service
container file system
     # The paths within the container should be configured in the target
section of connector-info [with type=FS]
     -
<target_nfs_mount_on_host>:<target_location_passed_during_connector_in
```

e. **(Only Required for File Connector load Service)** If using the staging push feature, you will need to provide source files on the NFS mount point which is mapped to the staging area. Alternatively, you can also provide format files with the source files. If providing format files, set the environment variable `USER_PROVIDED_FORMAT_FILE` to true, and by default, pyarrow writer will create the format file. In the below example `/source/files/path` and staging area must share the same mount point.

```
unload-service:
  image: delphix-file-connector-unload-service-app:${VERSION}
     ...
     volumes:
          ...
     # Source location should be made available within the unload-service
container file system
     # The paths within the container should be configured in the source
section of connector-info [with type=FS]
     - /source/files/path:/etc/hyperscale/source
load-service:
  image: delphix-file-connector-load-service-app:${VERSION}
     ...
     volumes:
          ...
     # Target location should be made available within the load-service
container file system
     # The paths within the container should be configured in the target
section of connector-info [with type=FS]
     - /target/files/path:/etc/hyperscale/target
```

5. [Optional] Some data (for example, logs, configuration files, etc.) that is generated inside containers may be useful to debug possible errors or exceptions while running the hyperscale jobs, and as such it may be beneficial to persist these logs outside containers. The following data can be persisted outside the containers:
    a. The logs generated for each service i.e. unload, controller, masking, and load services.
    b. The sqlldr utility logs and control files at opt/sqlldr location in the load-service container.
    c. The file-upload folder at `/etc/hyperscale/uploads` in the controller-service container

If you would like to persist the above data on your host, then you have the option to do the same by setting up volume bindings in the respective service as indicated below, that map locations inside the containers to locations on the host in the `podman-compose.yaml` file. The host locations again must have a group ownership id of 50 with a permission mode of 770 or a user id of 65436 with a permission of 700, due to the same reasons as highlighted in step 3a.

Here are examples of the podman-compose.yaml file for Oracle, MS SQL, MongoDB data sources, Delimited file, and Parquet file masking:

For Oracle data source masking:

```
version: "4"
services:
  controller-service:
    image: delphix-controller-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    healthcheck:
      test: 'curl --fail --silent http://localhost:8080/actuator/health | grep UP ||
exit 1'
      interval: 30s
      timeout: 25s
      retries: 3
      start_period: 30s
    depends_on:
      - unload-service
      - masking-service
      - load-service
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-controller-data:/data
      - /home/hyperscale_user/logs/controller_service:/opt/delphix/logs
      - /mnt/hyperscale:/etc/hyperscale
    environment:
      - API_KEY_CREATE=${API_KEY_CREATE:-false}
      - EXECUTION_STATUS_POLL_DURATION=${EXECUTION_STATUS_POLL_DURATION:-12000}
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_CONTROLLER_SERVICE:-INFO}
      - API_VERSION_COMPATIBILITY_STRICT_CHECK=$
{API_VERSION_COMPATIBILITY_STRICT_CHECK:-false}
      - LOAD_SERVICE_REQUIREPOSTLOAD=${LOAD_SERVICE_REQUIRE_POST_LOAD:-true}
      - SKIP_UNLOAD_SPLIT_COUNT_VALIDATION=$
{SKIP_UNLOAD_SPLIT_COUNT_VALIDATION:-false}
      - SKIP_LOAD_SPLIT_COUNT_VALIDATION=${SKIP_LOAD_SPLIT_COUNT_VALIDATION:-false}
      - CANCEL_STATUS_POLL_DURATION=${CANCEL_STATUS_POLL_DURATION:-60000}
      - NFS_STORAGE_HOST=${NFS_STORAGE_HOST:-}
      - NFS_STORAGE_EXPORT_PATH=${NFS_STORAGE_EXPORT_PATH:-}
      - NFS_STORAGE_MOUNT_TYPE=${NFS_STORAGE_MOUNT_TYPE:-}
      - NFS_STORAGE_MOUNT_OPTION=${NFS_STORAGE_MOUNT_OPTION:-}
      - APPLICATION_NAME=${APPLICATION_NAME:-hs-staging-area}
    unload-service:
    image: delphix-unload-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_UNLOAD_SERVICE:-INFO}
      - UNLOAD_FETCH_ROWS=${UNLOAD_FETCH_ROWS:-10000}
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-unload-data:/data
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/unload_service:/opt/delphix/logs
  masking-service:
    image: delphix-masking-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-masking-data:/data
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/masking_service:/opt/delphix/logs
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_MASKING_SERVICE:-INFO}
      - INTELLIGENT_LOADBALANCE_ENABLED=${INTELLIGENT_LOADBALANCE_ENABLED:-true}
  load-service:
    image: delphix-load-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_LOAD_SERVICE:-INFO}
      - SQLLDR_BLOB_CLOB_CHAR_LENGTH=${SQLLDR_BLOB_CLOB_CHAR_LENGTH:-20000}
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-load-data:/data
      - /mnt/hyperscale:/etc/hyperscale
      - /opt/oracle/instantclient_21_5:/usr/lib/instantclient
      - /home/hyperscale_user/logs/load_service:/opt/delphix/logs
      - /home/hyperscale_user/logs/load_service/sqlldr:/opt/sqlldr/
  proxy:
     image: delphix-hyperscale-masking-proxy:${VERSION}
    init: true
    networks:
      - hyperscale-net
    ports:
      - "443:8443"
      - "80:8080"
    restart: unless-stopped
    depends_on:
      - controller-service
      #volumes:
      # Uncomment to bind mount /etc/config
      #- /nginx/config/path/on/host:/etc/config
networks:
  hyperscale-net:
volumes:
  hyperscale-load-data:
  hyperscale-unload-data:
  hyperscale-masking-data:
  hyperscale-controller-data:
```

For MS SQL data source masking: A sample file specific to MS SQL connector is should look like following.

```
version: "4"
services:
  controller-service:
    image: delphix-controller-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    healthcheck:
      test: 'curl --fail --silent http://localhost:8080/actuator/health | grep UP ||
exit 1'
      interval: 30s
      timeout: 25s
      retries: 3
      start_period: 30s
    depends_on:
      - unload-service
      - masking-service
      - load-service
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-controller-data:/data
      - /home/hyperscale_user/logs/controller_service:/opt/delphix/logs
      - /mnt/hyperscale:/etc/hyperscale
    environment:
      - API_KEY_CREATE=${API_KEY_CREATE:-false}
      - EXECUTION_STATUS_POLL_DURATION=${EXECUTION_STATUS_POLL_DURATION:-12000}
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_CONTROLLER_SERVICE:-INFO}
      - API_VERSION_COMPATIBILITY_STRICT_CHECK=$
{API_VERSION_COMPATIBILITY_STRICT_CHECK:-false}
      - LOAD_SERVICE_REQUIREPOSTLOAD=${LOAD_SERVICE_REQUIRE_POST_LOAD:-true}
      - SKIP_UNLOAD_SPLIT_COUNT_VALIDATION=$
{SKIP_UNLOAD_SPLIT_COUNT_VALIDATION:-false}
      - SKIP_LOAD_SPLIT_COUNT_VALIDATION=${SKIP_LOAD_SPLIT_COUNT_VALIDATION:-false}
      - CANCEL_STATUS_POLL_DURATION=${CANCEL_STATUS_POLL_DURATION:-60000}
      - NFS_STORAGE_HOST=${NFS_STORAGE_HOST:-}
      - NFS_STORAGE_EXPORT_PATH=${NFS_STORAGE_EXPORT_PATH:-}
      - NFS_STORAGE_MOUNT_TYPE=${NFS_STORAGE_MOUNT_TYPE:-}
      - NFS_STORAGE_MOUNT_OPTION=${NFS_STORAGE_MOUNT_OPTION:-}
      - APPLICATION_NAME=${APPLICATION_NAME:-hs-staging-area}
  unload-service:
    image: delphix-mssql-unload-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_UNLOAD_SERVICE:-INFO}
      - UNLOAD_FETCH_ROWS=${UNLOAD_FETCH_ROWS:-10000}
      - SPARK_DATE_TIMESTAMP_FORMAT=${DATE_TIMESTAMP_FORMAT:-yyyy-MM-dd 
HH:mm:ss.SSSS}
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-unload-data:/data
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/unload_service:/opt/delphix/logs
  masking-service:
    image: delphix-masking-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-masking-data:/data
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/masking_service:/opt/delphix/logs
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_MASKING_SERVICE:-INFO}
      - INTELLIGENT_LOADBALANCE_ENABLED=${INTELLIGENT_LOADBALANCE_ENABLED:-true}
  load-service:
    image: delphix-mssql-load-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_LOAD_SERVICE:-INFO}
      - SQLLDR_BLOB_CLOB_CHAR_LENGTH=${SQLLDR_BLOB_CLOB_CHAR_LENGTH:-20000}
      - SPARK_DATE_TIMESTAMP_FORMAT=${DATE_TIMESTAMP_FORMAT:-yyyy-MM-dd 
HH:mm:ss.SSSS}
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-load-data:/data
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/load_service:/opt/delphix/logs
  proxy:
      image: delphix-hyperscale-masking-proxy:${VERSION}
    init: true
    networks:
      - hyperscale-net
    ports:
      - "443:8443"
      - "80:8080"
    restart: unless-stopped
    depends_on:
      - controller-service
      #volumes:
      # Uncomment to bind mount /etc/config
      #- /nginx/config/path/on/host:/etc/config
networks:
  hyperscale-net:
volumes:
  hyperscale-load-data:
  hyperscale-unload-data:
  hyperscale-masking-data:
  hyperscale-controller-data:
```

For Delimted and Parquet Files masking: A sample file specific to the File connector should look like the following.

```
version: "4"
services:
  controller-service:
    image: delphix-controller-service-app:<HYPERSCALE VERSION>
    security_opt:
      - label:disable
    userns_mode: keep-id
    healthcheck:
      test: 'curl --fail --silent http://localhost:8080/actuator/health | grep UP ||
exit 1'
      interval: 30s
      timeout: 25s
      retries: 3
      start_period: 30s
    depends_on:
      - unload-service
      - masking-service
      - load-service
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-controller-data:/data
      - /mnt/parent_staging_area:/etc/hyperscale
    environment:
      - API_KEY_CREATE=true
      - EXECUTION_STATUS_POLL_DURATION=120000
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=INFO
      - API_VERSION_COMPATIBILITY_STRICT_CHECK=false
      - LOAD_SERVICE_REQUIREPOSTLOAD=false
      - SKIP_UNLOAD_SPLIT_COUNT_VALIDATION=false
      - SKIP_LOAD_SPLIT_COUNT_VALIDATION=false
      - CANCEL_STATUS_POLL_DURATION=60000
      - SOURCE_KEY_FIELD_NAMES=unique_source_files_identifier
      - NFS_STORAGE_HOST=${NFS_STORAGE_HOST:-}
      - NFS_STORAGE_EXPORT_PATH=${NFS_STORAGE_EXPORT_PATH:-}
      - NFS_STORAGE_MOUNT_TYPE=${NFS_STORAGE_MOUNT_TYPE:-}
      - NFS_STORAGE_MOUNT_OPTION=${NFS_STORAGE_MOUNT_OPTION:-}
      - APPLICATION_NAME=${APPLICATION_NAME:-hs-staging-area}
  unload-service:
    image: delphix-file-connector-unload-service-app:<HYPERSCALE VERSION>
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-unload-data:/data
      # Staging area volume mount, here /mnt/parent_staging_area is used as an example
      - /mnt/parent_staging_area:/etc/hyperscale
      # Source files should be made available within the unload-service container file system
      # The paths within the container should be configured in the source section of connector-info [with type=FS]
      - /mnt/source_files:/mnt/source
      #- /mnt/source_files2:/mnt/source2
      # In case source is hadoop
      #- /path/to/keytab_file/hadoop.keytab:/app/hadoop.keytab
      #- /path/to/hadoop/core-site.xml:/app/hadoop/etc/hadoop/core-site.xml
      #- /path/to/etc/krb5.conf:/etc/krb5.conf
      # In case user want to provide external Hadoop client for pyarrow writer then they can provide
      #- /path/to/hadoop/client/hadoop:/app/hadoop
  masking-service:
    image: delphix-masking-service-app:<HYPERSCALE VERSION>
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-masking-data:/data
      # Staging area volume mount, here /mnt/parent_staging_area is used as an example
      - /mnt/parent_staging_area:/etc/hyperscale
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=INFO
      - INTELLIGENT_LOADBALANCE_ENABLED=true
  load-service:
    image: delphix-file-connector-load-service-app:<HYPERSCALE VERSION>
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-load-data:/data
      # Staging area volume mount, here /mnt/parent_staging_area is used as an
example
      - /mnt/parent_staging_area:/etc/hyperscale
      # Target location should be made available within the load-service container
file system
      # The paths within the container should be configured in the target section of
connector-info [with type=FS]
      - /mnt/target_files:/mnt/target
      #- /mnt/target_files2:/mnt/target2
      # In case target is hadoop
      #- /path/to/keytab_file/hadoop.keytab:/app/hadoop.keytab
      #- /path/to/hadoop/core-site.xml:/app/hadoop/etc/hadoop/core-site.xml
      #- /path/to/etc/krb5.conf:/etc/krb5.conf
      # In case user want to provide external Hadoop client for pyarrow writer then
they can provide
      #- /path/to/hadoop/client/hadoop:/app/hadoop
  proxy:
     image: delphix-hyperscale-masking-proxy:${VERSION}
    init: true
    networks:
      - hyperscale-net
    ports:
      - "443:8443"
      - "80:8080"
    restart: unless-stopped
    depends_on:
      - controller-service
networks:
  hyperscale-net:
volumes:
  hyperscale-load-data:
  hyperscale-unload-data:
  hyperscale-masking-data:
  hyperscale-controller-data:
```

For MongoDB data source masking:

```
# Copyright (c) 2023, 2024 by Delphix. All rights reserved.
version: "4"
services:
  controller-service:
    build:
      context: controller-service
      args:
        - VERSION=${VERSION}
    image: delphix-controller-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    healthcheck:
      test: 'curl --fail --silent http://localhost:8080/actuator/health | grep UP ||
exit 1'
      interval: 30s
      timeout: 25s
      retries: 3
      start_period: 30s
    depends_on:
      - unload-service
      - masking-service
      - load-service
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-controller-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /mnt/hyperscale:/etc/hyperscale
    environment:
      - API_KEY_CREATE=${API_KEY_CREATE:-true}
      - EXECUTION_STATUS_POLL_DURATION=${EXECUTION_STATUS_POLL_DURATION:-120000}
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_CONTROLLER_SERVICE:-INFO}
      - API_VERSION_COMPATIBILITY_STRICT_CHECK=$
{API_VERSION_COMPATIBILITY_STRICT_CHECK:-false}
      - LOAD_SERVICE_REQUIREPOSTLOAD=${LOAD_SERVICE_REQUIRE_POST_LOAD:-true}
      - SKIP_UNLOAD_SPLIT_COUNT_VALIDATION=$
{SKIP_UNLOAD_SPLIT_COUNT_VALIDATION:-false}
      - SKIP_LOAD_SPLIT_COUNT_VALIDATION=${SKIP_LOAD_SPLIT_COUNT_VALIDATION:-false}
      - CANCEL_STATUS_POLL_DURATION=${CANCEL_STATUS_POLL_DURATION:-60000}
      # update below parameters for creation/updation of mount-filesystem
      - NFS_STORAGE_HOST=${NFS_STORAGE_HOST:-}
      - NFS_STORAGE_EXPORT_PATH=${NFS_STORAGE_EXPORT_PATH:-}
      - NFS_STORAGE_MOUNT_TYPE=${NFS_STORAGE_MOUNT_TYPE:-}
      - NFS_STORAGE_MOUNT_OPTION=${NFS_STORAGE_MOUNT_OPTION:-}
      - APPLICATION_NAME=${APPLICATION_NAME:-hs-staging-area}

      # uncomment below for Delimited files connector
      #- SOURCE_KEY_FIELD_NAMES=unique_source_files_identifier
      # uncomment below for MongoDB connector
      - SOURCE_KEY_FIELD_NAMES=database_name,collection_name
      - VALIDATE_UNLOAD_ROW_COUNT_FOR_STATUS=$
{VALIDATE_UNLOAD_ROW_COUNT_FOR_STATUS:-false}
      - VALIDATE_MASKED_ROW_COUNT_FOR_STATUS=$
{VALIDATE_MASKED_ROW_COUNT_FOR_STATUS:-false}
      - VALIDATE_LOAD_ROW_COUNT_FOR_STATUS=$
{VALIDATE_LOAD_ROW_COUNT_FOR_STATUS:-false}
      - DISPLAY_BYTES_INFO_IN_STATUS=${DISPLAY_BYTES_INFO_IN_STATUS:-true}
      - DISPLAY_ROW_COUNT_IN_STATUS=${DISPLAY_ROW_COUNT_IN_STATUS:-true}
```

```
  unload-service:
    build:
      context: unload-service
      args:
        - VERSION=${VERSION}
    image: delphix-mongo-unload-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    environment:
      - CONCURRENT_EXPORT_LIMIT=${CONCURRENT_EXPORT_LIMIT:-10}
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_UNLOAD_SERVICE:-INFO}
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-unload-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /mnt/hyperscale:/etc/hyperscale

  masking-service:
    build:
      context: masking-service
      args:
        - VERSION=${VERSION}
    image: delphix-masking-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-masking-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /mnt/hyperscale:/etc/hyperscale
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_MASKING_SERVICE:-INFO}
      - INTELLIGENT_LOADBALANCE_ENABLED=${INTELLIGENT_LOADBALANCE_ENABLED:-true}

  load-service:
    build:
      context: load-service
      args:
        - VERSION=${VERSION}
    image: delphix-mongo-load-service-app:${VERSION}
    security_opt:
      - label:disable
    userns_mode: keep-id
    init: true
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${LOG_LEVEL_LOAD_SERVICE:-INFO}
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-load-data:/data
      # The orchestrator VM paths(left side of colon) used here are examples.
Configure the respective mount paths.
      - /mnt/hyperscale:/etc/hyperscale
  proxy:
      image: delphix-hyperscale-masking-proxy:${VERSION}
    init: true
    networks:
      - hyperscale-net
    ports:
      - "443:8443"
      - "80:8080"
    restart: unless-stopped
    depends_on:
      - controller-service
      #volumes:
      # Uncomment to bind mount /etc/config
      #- /nginx/config/path/on/host:/etc/config
networks:
  hyperscale-net:
volumes:
  hyperscale-load-data:
  hyperscale-unload-data:
  hyperscale-masking-data:
  hyperscale-controller-data:
```

6. (OPTIONAL) To modify the default Hyperscale configuration properties for the application, see Configuration Settings.

7. Run the application from the same location where you extracted the `podman-compose.yaml` file.

```
podman-compose up -d
```

- Run the following command to check if the application is running. The output of this command should shows five containers up and running.

```
podman-compose ps
```

- Run the following command to access application logs of a given container.

```
podman logs -f service_container_name>
```

Service container name can be accessed by output of the command `podman-compose ps`.

- Run the following command to stop the application (if required).

`podman-compose down`

8. Once the application starts, an API key will be generated that will be required to authenticate with the Hyperscale Compliance Orchestrator. This key will be found in the podman container logs of the controller service. You can either look for the key from the controller service logs location that was set as a volume binding in the `podman-compose.yaml` file or you could use the following 'podman' command to retrieve the logs.

   `podman logs -f <service_container_name>`

> Service container name can be accessed by output of the command `podman-compose ps`.

The above command displays an output similar to the following where the string `NEWLY GENERATED API KEY` can be grepped from the log::

```
2022-05-18 12:24:10.981  INFO 7 --- [           main] o.a.c.c.C.[Tomcat].[localhost].
[/]    : Initializing Spring embedded WebApplicationContext
2022-05-18 12:24:10.982  INFO 7 --- [           main]
w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization
completed in 9699 ms
NEWLY GENERATED API KEY: 1.89lPH1dHSJQwHuQvzawD99sf4SpBPXJADUmJS8v00VCF4V7rjtRFAftGWy
gFfsqM
```

To authenticate with the Hyperscale Compliance Orchestrator, you must use the API key and include the HTTP Authorization request header with the type apk; `apk <API Key>`.

For more information, see the **Authentication** section under Accessing the Hyperscale Compliance API.

Continuous Compliance Engine Installation

Delphix Continuous Compliance Engine is a multi-user, browser-based web application that provides complete, secure, and scalable software for your sensitive data discovery, masking, and tokenization needs while meeting enterprise-class infrastructure requirements. For information about installing the Continuous Compliance Engine, see Continuous Compliance Engine Installation documentation.

Upgrading the Hyperscale Compliance Orchestrator (Podman Compose)

Pre-requisite

Before upgrading, ensure you have downloaded the Hyperscale Compliance x.0.0 (where `x.0.0` should be changed to the version of Hyperscale being installed) tar bundle from the Delphix Download website.

How to upgrade the Hyperscale Compliance Orchestrator

Perform the following steps to upgrade the Hyperscale Compliance Orchestrator to the x.0.0 version:

1. Run `cd /<hyperscale_installation_path>/` and `podman-compose down` to stop and remove all the running containers.
2. Run the below commands to delete all existing dangling images and Hyperscale images:

   ```
   podman rmi $(podman images -f "dangling=true" -q)
   podman rmi $(podman images "delphix-hyperscale-masking-proxy" -q)
   podman rmi $(podman images "delphix-controller-service-app" -q)
   podman rmi $(podman images "delphix-masking-service-app" -q)
   podman rmi $(podman images "delphix-*load-service-app" -q)
   ```

3. Remove all files or folders from existing installation directories, except `podman-compose.yaml` (Keep its backup outside the installation directory so it is not overridden while executing the next step) and the `.env` file.
4. Take backup of .env file and untar the patch tar in your existing installation path (where `x.0.0` should be changed to the version of Hyperscale being installed). `tar -xzvf delphix-hyperscale-masking-x.0.0.tar.gz -C <existing_installation_path>`
5. Replace the `podman-compose.yaml` supplied with the bundle file with your backed up `podman-compose.yaml` file from step 3. Alternatively, copy over the differing bits(essentially the volumes and properties for each service) from the backed-up `podman-compose.yaml` file to the new `podman-compose.yaml` file supplied in the bundle.
6. Similarly, either replace the new .env file with the backed up .env file from step 4 and set the VERSION property as x.0.0 (i.e. VERSION=x.0.0) or use the new .env file from the installation bundle and set any properties in it as set in the old .env file.
7. Run the below commands to load the images(this will configure Oracle-based unload/load setup):

   ```
   podman load --input controller-service.tar
   podman load --input unload-service.tar
   podman load --input masking-service.tar
   podman load --input load-service.tar
   podman load --input proxy.tar
   ```

   - If upgrading from an MSSQL connector setup(supported starting 5.0.0.0 release), instead of running the above commands for load/unload services setup(which are for Oracle), run the below commands(rest remains same for the controller, masking, and proxy services):

```
podman load --input mssql-unload-service.tar
podman load --input mssql-load-service.tar
```

- If upgrading from a Delimited/Parquet Files connector setup (supported starting 12.0.0 release), instead of running the above commands for load/unload services setup(which are for Oracle), run the below commands(rest remains same for the controller, masking, and proxy services) after updating new image names in podman-compose.yaml:

```
podman load --input file-connector-unload-service.tar
podman load --input file-connector-load-service.tar
```

- If upgrading from a MongoDB connector setup (supported starting 13.0.0 release), instead of running the above commands for load/unload services setup (which are for Oracle), run the below commands (rest remains same for the controller, masking, and proxy services):

```
podman load --input mongo-unload-service.tar
podman load --input mongo-load-service.tar
```

8. Make sure to have the below ports configured under proxy service:

```
ports:
    - "443:8443"
    - "80:8080"
```

9. Ensure your mounts are configured and accessible, before running a job.
If upgrading to version 24.0 (and onwards) , ensure that the location mounted on the Hyperscale host is the same as the one mapped to `/etc/hyperscale` in your podman-compose.yaml. If a previous mount exists on another location, unmount it and re-mount to the right directory e.g. if a mount point exists with name `staging_area` at path `/mnt/provision/staging_area` , execute the following commands and restart the containers.
    a. If NFS file server is used as staging server execute this command:

```
sudo umount /mnt/provision/staging_area
sudo mount -t nfs4 <source_nfs_endpoint>:<source_nfs_location> /mnt/
provision
```

    b. If the NFS Server installation is a Delphix Continuous Data Engine empty VDB you can either:
        i. Append the staging_area path to the volume binding for all the services in podman-compose.yaml. For example:

```
volumes:
      - /mnt/provision/staging_area:/etc/hyperscale
```

        ii. Alternatively, you can update mount path of Environment on Continuous Data Engine:
            1. Disable the empty VDB(Data Set).
            2. Update path on Environment → Databases page. For example: change path from `/mnt/provision/staging_area` to `mnt/provision/`

3. Enable the empty VDB(Data Set).
4. Restart Hyperscale containers.

After re-mounting, recheck the permissions of staging area on Hyperscale host. Refer to instruction number 3 on this page: Installation and Setup for required staging area permissions.

> ⚠ Upon application startup, all existing mount-filsystems will be deleted. Please ensure to take backup of the mount setup details, if needed.

> ℹ If using file connectors, any unload or load jobs in a running state at the time of a container restart are marked as failed.

10. Run `podman-compose up -d` to create containers.

NFS server installation

The Hyperscale Compliance Orchestrator requires a Staging Area to read from the source file(s) and write to the target file(s). The Staging Area must be an NFS-shared filesystem accessible to the Hyperscale Compliance Orchestrator and the Continuous Compliance Engines. The following are the supported ways by which the filesystem can be shared over NFS(NFSv3/NFSv4):

Delphix Continuous Data Engine empty VDB

To create a Delphix Virtualization Engine empty VDB, follow the below procedure.

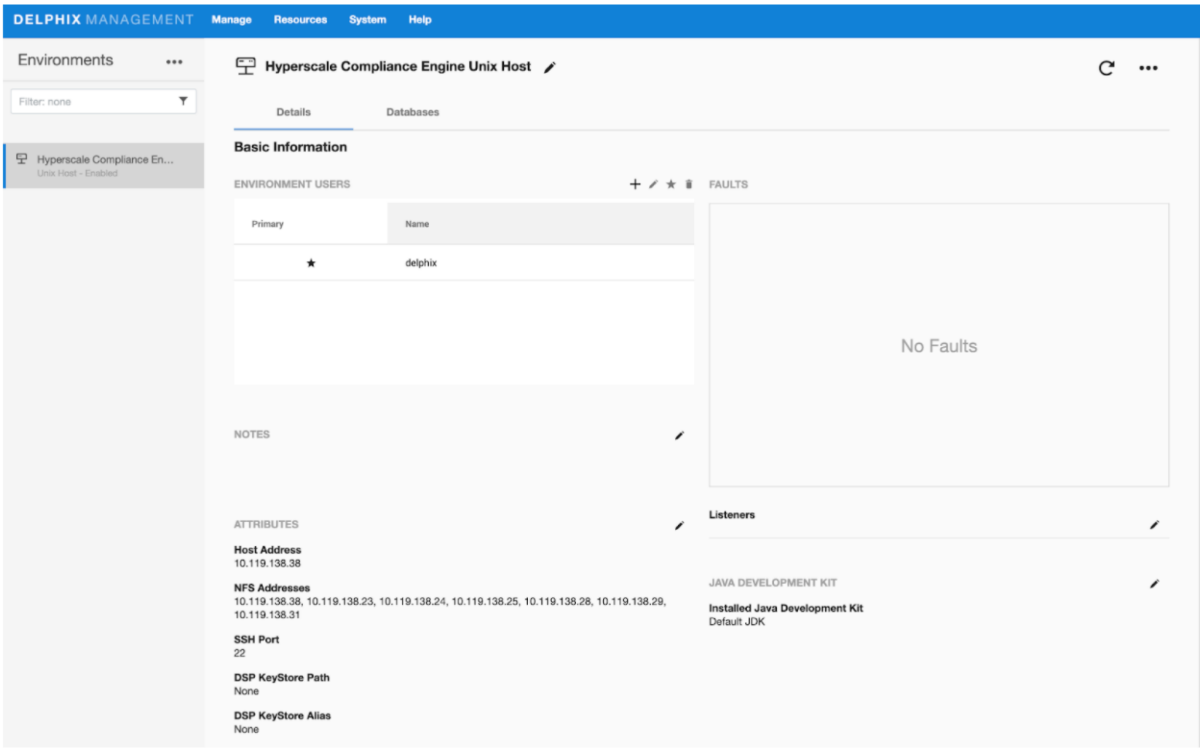
Continuous Data Engine installation

Delphix Virtualization Engine is a data management platform that provides the ability to securely copy and share datasets. Using virtualization, you will ingest your data sources and create virtual data copies, which are full read-write capable database instances that use a small fraction of the resources a normal database copy would require.

For information about installing the Virtualization Engine, see Virtualization Engine Installation documentation.

Discover and configure Hyperscale Compliance Orchestrator environment

1. After installing and configuring the Virtualization Engine, make sure that the Network and Connectivity Requirements for using Empty VDB on Unix environments are met.
2. Discover the Hyperscale Compliance Orchestrator Unix host on the Virtulization’s Engine Management application. For more information, see Adding a Unix Environment.
3. Navigate to **Manage > Environments** to view the discovered Hyperscale Compliance Orchestrator Unix host.
4. After the discovery is completed, configure the same Unix host on the Environments screen such that the IP addresses of the Hyperscale Compliance Orchestrator Unix host along with the Continuous Compliance Engines part of the Continuous Compliance Engine cluster are populated in the NFS Addresses field. This is done to ensure that the empty VDB is shared with both Hyperscale Compliance Orchestrator and the Continuous Compliance Engines part of the Continuous Compliance Engine cluster.



Primary	Name
★	delphix

Host Address
10.119.138.38

NFS Addresses
10.119.138.38, 10.119.138.23, 10.119.138.24, 10.119.138.25, 10.119.138.28, 10.119.138.29, 10.119.138.31

SSH Port
22

DSP KeyStore Path
None

DSP KeyStore Alias
None

Installed Java Development Kit
Default JDK

Provision an empty VDB

1. Follow the steps listed under Create an Empty VDB for Unstructured Files in the Delphix Engine to provision an empty VDB on the discovered Hyperscale Compliance Orchestrator Unix host.

2. Note the mount path provided while provisioning the empty VDB as that is the path which will be used to fill the empty VDB with the source file(s) that the Hyperscale Compliance Orchestrator needs to mask and where the target masked file(s) will be placed.

> Hyperscale Compliance OS user should have read/write permissions on the mount point path where the empty VDB will be provisioned.Hyperscale Compliance OS user should have read/write permissions on the mount point path where the empty VDB will be provisioned.

The location of the mounted empty VDB on the Hyperscale Compliance Orchestrator Unix host can be found with a simple ‘grep’ of the mount path, provided while provisioning the empty VDB, using the ‘mount’ utility:

```
hyperscale-engine:~$ df -h | grep /mnt/provision/hyperscale_data

10.119.138.34:/domain0/group-2/appdata_container-3/appdata_timeflow-4/datafile 20T 3.
5T

16T 18% /mnt/provision/hyperscale_data
```

3. Copy the source file(s) to the location where the empty VDB has been mounted.

NFS file server

1. An NFS shared filesystem can also be provided by a typical NFS server. Export a filesystem from the NFS file server such that the Hyperscale Compliance Orchestrator and Continuous Compliance Engines part of the Continuous Compliance Engine Cluster have read and write permission on it. As such, the export entry should be of the following form based on the UID/GID corresponding to the owner of the shared path:

   ```
   <mount_path> <ip1,ip2,ip3,ipn>(rw,all_squash,anonuid=<uid>,anongid=<gid>)
   ```

2. Export the NFS share using the below command:

   ```
   sudo exportfs -rav
   ```

3. Once the NFS share is exported from the NFS server, proceed to mount the same share on the Hyperscale Compliance Orchestrator host.

```
sudo mount -t nfs -o vers=4 <nfs-server-host-ip>:<mount_path> <user.home>/hyperscale
```

Storage requirements for the NFS file server

Considering a single Hyperscale Compliance job execution, the Hyperscale Compliance Orchestrator will store unloaded files (unloaded from source) and masked files. As such, the required storage will amount to 2X the size of the source data.

Configuring AWS S3 bucket as staging area

The Hyperscale Compliance Orchestrator requires a staging area to read from the source file(s) and write to the target file(s). In Kubernetes deployment of Hyperscale Compliance, we can now leverage AWS S3 buckets accessible to the Hyperscale Compliance Orchestrator and the Continuous Compliance engine as the staging area. This feature is supported on microk8s and AWS EKS distributions.

Pre-requisites:

AWS provides Delphix with a Container Storage Interface (CSI) driver, and using this driver, a given Kubernetes application can access S3 objects through a file system interface. To configure the mount point through this CSI driver, a Kubernetes cluster administrator/AWS account admin will need to perform the following tasks:

1. Creation of S3 bucket
2. Creation of IAM policy
3. Creation of IAM Role
4. Installing the mount point for S3 CSI driver (version v1.11.0 or later)

⚠ **Troubleshooting tip:**
If installing the S3 CSI driver on microk8s, you may encounter the following error:

```
Could not check if "/var/snap/microk8s/common/var/lib/kubelet/pods/
4a5d6dfb-646a-462d-a223-d510c059f8b7/volumes/kubernetes.io~csi/s3-pv/mount" is
a mount point, Failed to read /host/proc/mounts: open /host/proc/mounts:
invalid argument
```

because microK8s uses /var/snap/microk8s/common/var/lib/kubelet as the kubelet path which is different from other standard distributions. We need to inform this path to driver while installing it using helm.
The helm chart installation command should be:

```
helm upgrade --install aws-mountpoint-s3-csi-driver --namespace kube-system
aws-mountpoint-s3-csi-driver/aws-mountpoint-s3-csi-driver --set
node.kubeletPath=/var/snap/microk8s/common/var/lib/kubelet
```

Setup:

Once the pre-requisites are met, a persistent volume needs to be created. It can be achieved in either of the following ways:

- Cluster administration creates a PV using the template below and shares the PV name which is configured against the parameter ***stagePvName*** in values.yaml.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <pv-name>
spec:
  capacity:
    storage: 5Gi # ignored, required
  accessModes:
    - ReadWriteMany
  mountOptions:
    - uid=65436
    - gid=50
    - allow-other
    - allow-delete
    - region <bucket-region>
  csi:
    driver: s3.csi.aws.com  #required
    volumeHandle: s3-csi-driver-volume
    volumeAttributes:
      bucketName: <bucket-name>
```

- Hyperscale Orchestrator creates the PV automatically during deployment using the AWS S3 related properties defined in values.yaml.

The following properties shall be specified in values.yaml (even if PV is created by cluster admin).

- stagingStorageType
- authMechanism
- awsBucketName
- awsBucketRegion
- awsBucketPrefix
- awsBucketDelimiter
- awsAccessKey
- awsSecretKey
- stagingAwsS3SecretName

For more information on these parameters, see Commonly Used Properties

Notes:

- This feature is compatible only with Continuous Compliance engines v20.0.0 and above.
- The stagingStorageType must be set as AWS_S3
- The following two authentication mechanisms to connect to S3 bucket from the Continuous Compliance engine are supported:
  - Instance profile based authentication (AWS_ROLE)
  - Access and secret key based authentication (AWS_SECRET)
- If authMechanism is set as AWS_ROLE, an IAM instance profile, granting access to the S3 bucket, must be attached to the EC2 instances of the Hyperscale Compliance engine (or to the EKS cluster in case of EKS deployment) and the Continuous Compliance engines. For further details, refer to this AWS Knowledge Center article.
- If authMechanism is set as AWS_SECRET, either of the following information shall be provided:
  - **stagingAwsS3SecretName** : Name of the Kubernetes secret that holds awsAccessKey and awsSecretKey information. It can be created by cluster admin using command line as follows:

```
kubectl create secret generic <secret-name> --from-
literal=accessKeyId=<bucketAccessKeyId> --from-
literal=secretAccessKey=<bucketSecretAccessKey>
```

- **awsAccessKey** and **awsSecretKey** : Hyperscale Orchestrator will use these values and automatically create a generic secret.

Configuring Azure Blob containers as staging area

The Hyperscale Compliance Orchestrator requires a staging area to read from the source files and write to the target files. In Kubernetes deployment of Hyperscale Compliance, you can now use Azure Blob Containers as a staging area to access the Hyperscale Compliance Orchestrator and the Continuous Compliance engine's source and target files.

This feature is supported on MicroK8s and Microsoft's Azure Kubernetes Service (AKS) distributions and it is only available for Continuous Compliance engine version 28.0.0.0 onwards.

Prerequisites

For Hyperscale Compliance containers to access Blob objects through a file system interface, a CSI driver (Container Storage Interface) must be configured on the Kubernetes cluster.

To configure the mount point through the CSI driver, a Kubernetes cluster administrator or an Azure account administrator will need to:

1. Create an Azure Storage account.
2. Open the Azure storage account and create a Blob Container.
3. Create and assign Azure's built-in roles such as Owner or Reader roles to users. For instructions, see Azure RBAC documentation.
4. Create a managed identity by following the instructions provided in Managed identities for Azure resources
5. Install the mount point for Blob CSI driver version v1.25.1 or later. For installation instructions, see Azure Kubernetes Service and MicroK8s.

Troubleshooting

As MicroK8s uses a different path to other standard distributions, when you are installing the Blob CSI driver on MicroK8s using: `/var/snap/microk8s/common/var/lib/kubelet` you may receive this error:

Could not check if "/var/snap/microk8s/common/var/lib/kubelet/pods/4a5d6dfb-646a-462d-a223-d510c059f8b7/volumes/kubernetes.io~csi/blob-pv/mount" is a mount point, Failed to read /host/proc/mounts: open /host/proc/mounts: invalid argument

If you encounter the above error, you will need to inform the `/var/snap/microk8s/common/var/lib/kubelet` path to driver while installing it using helm.

This is the helm chart installation command:

```
helm install blob-csi-driver blob-csi-driver/blob-csi-driver --set
node.enableBlobfuseProxy=true --set linux.kubelet="/var/snap/microk8s/common/var/lib/
kubelet" --namespace kube-system --version v1.25.1
```

You may also need to make the mount location shared:

```
/bin/mount --make-shared /var/snap/microk8s/common/var/lib/kubelet
```

Setup

When you have completed the steps in the prerequisites section, you will need to create a PV (persistent volume). A PV can be created via cluster administration or by using the Azure Blob related properties defined in values.yaml. Both options are discussed below:

- Cluster administration creates a PV using the template below and shares the PV name which is configured against the parameter ***stagePvName*** in values.yaml.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: blob.csi.azure.com
  name: blob-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  mountOptions:
    - -o allow_other
    - -o direct_io
    - --file-cache-timeout-in-seconds=0
    - --use-attr-cache=false
    - --invalidate-on-sync=true
    - --attr-timeout=0
  csi:
    driver: blob.csi.azure.com
    volumeHandle: azure-csi-driver-volume-handle # Should be unique for a container
    volumeAttributes:
      storageAccount: <STORAGE_ACCOUNT>
      containerName: <CONTAINER_NAME>
      AzureStorageAuthType: MSI # key, MSI
      protocol: fuse2
```

- Hyperscale Orchestrator creates the PV automatically during deployment using the Azure Blob related properties defined in values.yaml.

The following properties need to be specified in values.yaml even if a PV is created by cluster admin.

- stagingStorageType
- blobAccountName
- blobContainerName
- blobAuthType
- blobContainerPrefix
- blobContainerDelimiter
- blobSecretKey
- stagingAzureBlobSecretName

For more information on these parameters, see Commonly Used Properties

Additional information

- Set the `stagingStorageType` as `AZURE_BLOB_STORAGE`
- To connect to the Blob Container from the Continuous Compliance engine, you can use either of the following authentication mechanisms:
    - Instance profile-based authentication (`AZURE_MANAGED_IDENTITY`)
    - Access and secret key based authentication (`AZURE_SECRET`)
- If authMechanism is set as AZURE_MANAGED_IDENTITY, a Managed Identity that grants access to the Blob Container must be attached to the Azure instances of either the Hyperscale Compliance engine, Continuous Compliance engine, or the AKS cluster. Refer to the Azure Kubernetes Services (AKS) knowledge guide for further information.
- If authMechanism is set as `AZURE_SECRET` , you will need to provide either of the following:
    - **stagingAzureBlobSecretName:** Name of the Kubernetes secret that holds `blobSecretKey` information. The `stagingAzureBlobSecretName` can be created by a cluster administrator using this command line:

      ```
      kubectl create secret generic <secret-name> --from-
      literal=azurestorageaccountname=<storage-account> --from-literal
      azurestorageaccountkey=<blob-secret--key> --type=Opaque
      ```
    - **blobSecretKey**: Hyperscale Orchestrator will use these values and automatically create a generic secret.

Accessing the Hyperscale Compliance API

To access the Hyperscale Compliance API, open a web browser and type the following in the address bar: `https://<hyperscale-compliance-host-address>/hyperscale-compliance`. **Before** navigating to the address, replace `<hyperscale-compliance-host-address>` (remove the angle brackets too) with the IP address of the Hyperscale Compliance Orchestrator VM.

Here is a sample command for Linux to retrieve the IP address.

```
kubectl describe service proxy -n hyperscale-services |grep "IP:" | tr -s " " | cut
-d " " -f 2
```

Authentication

To authenticate with the Hyperscale Compliance Orchestrator, you must use an API key. This is done by including the key in the HTTP authorization request header with the type `apk`.

An example cURL command with the API key would appear as:

```
curl --header 'Authorization: apk
1.t8YTjLyPiMatdtnhAw9RD0gRVZr2hFsrfikp3YxVl8URdB9zuaVHcMuhXkLd1TLj'
```

As described in the HTTP Authorization request header documentation, this is the typical syntax for the authorization header – `Authorization: <auth-scheme> <authorisation-parameters>`.

- For Basic Authentication, include the following header parameters – `Authorization: Basic <credentials>`.
- For the Bearer Authentication scheme, use the following – `Authorization: Bearer <JWT Bearer Token>`.

Creating an API key

An API key is a simple encrypted string that you can use when calling Hyperscale Compliance APIs.

- You must use the initial created API key to create a new secure key. This is done by creating a new API Client entity. The `"name"` attribute must be the desired name to uniquely identify the user of this key.
- The API key created will never expire.
- It is recommended that you regularly rotate API key (by deleting the old one and creating a new one)

For more information on the initial created API key, refer to step 7 in these installation instructions.

Run the following command to create a new API key.

```
curl -X 'POST' \
  'https://<host-name>/api/<api_version>/api-keys' \
  -H 'accept: application/json' \
  -H 'Authorization: apk 
1.t8YTjLyPiMatdtnhAw9RD0gRVZr2hFsrfikp3YxVl8URdB9zuaVHcMuhXkLd1TLj' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "<name-of-key>"
}'
```

A response message similar to the one shown below should appear. Copy or save the newly created token from the response, this token value will **not** be accessible later.

```
{
  "api_key_id": 2,
  "token": "2.ExZtmf6EN1xvFMsXpXlOyhHVYlTuFzCm2yGhpUOQQ5ID8N8oGz79d4yn8ZsPhF46"
}
```

Now that a new and secure API key has been created, the old one must be deleted for security purposes. Run the following command to delete the old key.

```
curl -X 'DELETE' \
  'https://<host-name>/api/<api_version>/api-keys/1' \
  -H 'accept: */*' \
  -H 'Authorization: apk 
2.ExZtmf6EN1xvFMsXpXlOyhHVYlTuFzCm2yGhpUOQQ5ID8N8oGz79d4yn8ZsPhF46'
```

Using the newly generated key

After you delete the old key, revert the changes performed in step 4 of the Hyperscale Compliance Installation and restart docker-compose. You must be able to use the new key for authorization as follows:

```
curl --header 'Authorization: apk 
2.ExZtmf6EN1xvFMsXpXlOyhHVYlTuFzCm2yGhpUOQQ5ID8N8oGz79d4yn8ZsPhF46'
```

Default API Version

If the version is omitted from the base path of the request's URL, a default API version i.e. the latest API version of that Hyperscale Engine is used.

How to setup a Hyperscale Compliance job

Pre-checks

You must check the following before starting a job:

- Storage space must be twice the size of the source data for staging area.
- You must have sufficient storage in the target DB for loading the masked data.
- You must check and increase the size of the temporary tablespace in Oracle. For example, if you have 4 billion rows, then you must use 100G.
- You must check and provide the required permission(i.e. 770 or 700) after creating an empty VDB(or mounting an NFS share) on the mounted folder on the Hyperscale Compliance host.
- Based on the unmask value for the user that is used to mount, the permissions for the staging area directory could get altered after the empty VDB or NFS share has been mounted. In such cases, you must re-apply the permissions (i.e. 770 or 700) on the staging area directory.
- You must restart the services after changing the permission on VDB mounted folder in case you already have created the containers.
- Continuous Compliance Engine should be cleaned up before use and should only be used with Hyperscale Job. Any other masking job on Continuous Compliance Engine apart from Hyperscale Compliance Orchestrator will impact the performance of Hyperscale Compliance jobs.
- Currently, the Hyperscale Compliance Orchestrator doesn’t provide the ability to allow you to configure the masking job behavior in case of non-conformant data and does not process non-conformant data warnings from the Delphix Continuous Compliance Engine. Therefore, it is recommended to verify the value of `DefaultNonConformantDataHandling` algorithm group setting on all the Hyperscale Compliance Orchestrator. For more information, refer to the Algorithm Group Settings section. It is recommended to set the value to FAIL so that Hyperscale Job will also fail instead of leaving the data unmasked.
- If the table that you are masking has a column type of BLOB/CLOB, then you must have a minimum of 2GB memory per CLOB/BLOB column. Depending upon the unload-split you are using, you may need to increase this memory in multiple of that. For example, if you have 4 tables (each with 1 column as BLOB/CLOB type) and unload-split is 3, then your memory requirement on the Hyperscale Compliance host will be: `(4(no. of tables) x 2(memory required per CLOB/BLOB column) x 3(unload-split used)GB + 16 GB (minimum required memory for running Hyperscale Compliance Orchestrator) = 40 GB approx`.

API Flow to Setup a Hyperscale Compliance Job

The following is the API flow for setting up and executing a Hyperscale Compliance job.

1. Register Continuous Compliance Engine(s)
2. Create Connector Info
3. Upload format file using `POST /file-upload` endpoint. For more information, refer to https://delphixdocs.atlassian.net/wiki/spaces/~5ceab98e6f51c10fbd18ae6d/pages/230229360. [Required only for those who need to mask embedded XML/JSON data]
4. Create Structured Data Format [Required only for those who need to mask embedded XML/JSON data]
5. Create a Dataset
6. Create a Job
7. Create Execution

The following are the sample API requests/responses for a typical Hyperscale Compliance job execution workflow. The APIs can be accessed using a swagger-based API client by accessing the following URL; `https://<hyperscale-compliance-host-address>/hyperscale-compliance`.

> APIs must be called only in the below order.

Engines API

POST /engines (register an engine):

Request:

```
{
"name": "Delphix Continuous Compliance Engine 6.0.14.0 on AWS",
"type": "MASKING",
"protocol": "http",
"hostname": "de-6014-continuous-compliance.delphix.com",
"username": "hyperscale_compliance_user",
"password": "password123"
}
```

Response:

```
{
"id": 1,
"name": "Delphix Continuous Compliance Engine 6.0.14.0 on AWS",
"type": "MASKING",
"protocol": "http",
"hostname": "de-6014-continuous-compliance.delphix.com",
"username": "hyperscale_compliance_user",
"ssl": true,
"ssl_hostname_check": true
}
```

ConnectorInfo API

POST /connector-info (create connector info for Hyperscale Compliance)

Oracle Request:

```
{
"source": {
"jdbc_url": "jdbc:oracle:thin:@oracle-19-src.dlpxdc.co:1521/VDBOMSRDC20SRC",
"user": "oracle_db_user",
"password": "password123"
},
"target": {
"jdbc_url": "jdbc:oracle:thin:@rh79-ora-19-tgt.dlpxdc.co:1521/VDBOMSRDC200B_TGT",
"user": "oracle_db_user",
"password": "password123"
}
}
```

Oracle Response:

```
{
"id": 1,
"source": {
"jdbc_url": "jdbc:oracle:thin:@oracle-19-src.dlpxdc.co:1521/VDBOMSRDC20SRC",
"user": "oracle_db_user"
},
"target": {
"jdbc_url": "jdbc:oracle:thin:@rh79-ora-19-tgt.dlpxdc.co:1521/VDBOMSRDC200B_TGT",
"user": "oracle_db_user"
}
}
```

Example 2: This example is for the cases where either username or password needs to be in either uppercase or camel case

Oracle Request:

```
{
"source": {
"user": "\"y2ijf0oj2\"",
"password": "\"xyz\"",
"jdbc_url": "jdbc:oracle:thin:@xyz.com:1521/DBOMSRDC200B",
"connection_properties": {}
},
"target": {
"jdbc_url": "jdbc:oracle:thin:@xyx.com:1521/DBOMSRDC200B",
"user": "\"y2ijf0oj2\"",
"password": "\"xyz\"",
"connection_properties": {}
}
}
```

Oracle Response:

```
{
"id": 1,
"source": {
"user": "\"y2ijf0oj2\"",
"jdbc_url": "jdbc:oracle:thin:@xyz.com:1521/DBOMSRDC200B",
"connection_properties": {}
},
"target": {
"jdbc_url": "jdbc:oracle:thin:@xyx.com:1521/DBOMSRDC200B",
"user": "\"y2ijf0oj2\"",
"connection_properties": {}
}
}
```

MSSQL Request:

```
{
  "source": {
        "jdbc_url": "jdbc:sqlserver://hyperscale-
mssql.dlpxdc.co;database=SourceDB2019;instanceName=SQL2019",
     "user": "sa",
     "password": "password123"
  },
  "target": {
    "jdbc_url": "jdbc:sqlserver://hyperscale-
mssql.dlpxdc.co;database=SourceDB2019;instanceName=SQL2019;",
    "user": "sa",
     "password": "password123"
  }
}
```

MSSQL Response:

```
{
 "id": 1,
 "source": {
   "user": "sa",
   "jdbc_url": "jdbc:sqlserver://hyperscale-
mssql.dlpxdc.co;database=SourceDB2019;instanceName=SQL2019"
 },
 "target": {
   "jdbc_url": "jdbc:sqlserver://hyperscale-
mssql.dlpxdc.co;database=SourceDB2019;instanceName=SQL2019;",
   "user": "sa",
  }
}
```

MongoDB Connector Request:

Passwords with special characters must be percent encoded.

```
{
  "connectorName": "MongoDB_Connector",
  "source": {
    "mongo_url": "mongodb://<hostname>:<port>/?
authSource=<authSource>[&<other_options>]",
    "user": "mongo_user",
    "password": "mongo_password"
  },
  "target": {
    "mongo_url": "mongodb://<hostname>:<port>/?
authSource=<authSource>[&replicaSet=<mongo_rs>&tls=true&tlsCertificateKeyFile=<cert_p
ath>]",
    "user": "mongo_user",
    "password": "mongo_password"
  }
}
```

MongoDB Connector Response:

```
{
  "source": {
    "mongo_url": "mongodb://mongodb-src.example:27017",
    "user": "mongo_user",
  },
  "target": {
    "mongo_url": "mongodb://mongodb-tgt.example.com:27017",
    "user": "mongo_user",
  }
}
```

A failure in the load may leave the target datasource in an inconsistent state since the load step truncates the target when it begins. If the source and target data source are configured to be the same datasource and a failure occurs in the load step, it is recommended that the single datasource be restored from a backup(or use the continuous data engine's rewind feature if you have a VDB as the single datasource) after the failure in the load step as the datasource may be in an inconsistent state. After the datasource is restored, you may kick off another hyperscale job. If the source and target data source are configured to be different, you may use the Hyperscale Compliance Orchestrator restartability feature to restart the job from the point of failure in the load/post-load step.

Snowflake Connector Request:

```
{
  "connectorName": "Snowflake_Connector",
  "source": {
    "account": "<orgname>-<account_name>",
    "user": "snowflake_user"
  },
  "target": {
    "account": "<orgname>-<account_name>",
    "user": "snowflake_user"
  }
}
```

Snowflake Connector Response:

```
{
"id": 1,
"connectorName": "Snowflake_Connector",
  "source": {
    "account": "<orgname>-<account_name>",
    "user": "snowflake_user"
  },
  "target": {
    "account": "<orgname>-<account_name>",
    "user": "snowflake_user"
  }
}
```

Delimited/Parquet Request:

Example1 - Without AWS S3 bucket credentials:

```
{
 "source": {
  "type": "AWS",
  "properties": {
      "server": "S3",
      "path": "s3_bucket_source/sub_folder"
  }
 },
 "target": {
  "type": "AWS",
  "properties": {
      "server": "S3",
      "path": "s3_bucket_target/sub_folder"
  }
 }
}
```

Example2 - With AWS S3 bucket credentials:

```
{
 "source": {
  "type": "AWS",
  "properties": {
    "server": "S3",
    "path": "s3_bucket_source/sub_folder",
    "aws_region": "us-east-1",
    "aws_access_key_id": "AKIAYHUJKLDHMB",
    "aws_secret_access_key": "x2IXoHDYHhdydmmm&h12563kaka",
    "aws_role_arn": "56436882398"
  }
 },
 "target": {
  "type": "AWS",
  "properties": {
    "server": "S3",
    "path": "s3_bucket_target/sub_folder",
    "aws_region": "us-east-1",
    "aws_access_key_id": "AKIAYHUJKLDHMB",
    "aws_secret_access_key": "x2IXoHDYHhdydmmm&h12563kaka",
    "aws_role_arn": "56436882398"
  }
 }
}
```

Overview of the parameters:

- **type**: Refers to the type of connector you are creating. Delphix currently supports “AWS” which refers to “Cloud vendor”, indicating that all the source and target files are available within the container through AWS credentials.
- **properties**: Holds the server and path values.
- **server**: Points to the S3 location. Currently, this is ignored as ‘AWS’ is the only supported type.
- **path**: The path to the AWS source/target S3 bucket location. This path needs not be the exact path to the files, but a parent directory. In case you are planning to use a profiler, then this path must be the exact path to the parquet files.
- Additional parameters (in case AWS credentials need to be passed separately):
  - **aws_region**: The AWS region the S3 bucket is part of.
  - **aws_access_key_id**: The AWS access key ID generated for the AWS Role.
  - **aws_secret_access_key**: The AWS secret access key generated for the AWS Role.
  - **aws_role_arn**: The AWS Role Identifier

Example 3 - With Mounted Filesystem:

```
{
  "id": 1,
  "connectorName": "Parquet_Connector_FS",
  "source": {
    "type": "FS",
    "properties": {
      "server": "local",
      "path": "/mnt/source"
    }
  },
  "target": {
    "type": "FS",
    "properties": {
      "server": "local",
      "path": "/mnt/target"
    }
  }
}
```

Here `**type=FS**` means “**File System**”. Currently, the Parquet connector only supports files mounted directly onto the docker container, i.e. available on the container file system. In the above example `/mnt/source` & `/mnt/target` are paths inside the container that denote the source and target location respectively.

> In case of staging push, the source and target locations must share a mount point with the staging area. The user needs to specify the volume path as the parent directory. For instance, if the parent path of the staging area is `/etc/hyperscale`, then the source and target paths would be something like `/etc/hyperscale/source` and `/etc/hyperscale/target`. The user should place their source files in `/etc/hyperscale/source`, and they may optionally include format files following the naming convention `<unique_source_identifier>.fmt`. The output files will be available in `/etc/hyperscale/target`.

Overview of the parameters:

- **type**: Refers to the type of connector you are creating. In the above example, we have provided the type as "FS" which refers to Filesystem.
- **properties**: Holds the server and path values.
- **server**: Currently for "FS" we only allow "local" as the server.
- **path**: This is the path for source and target locations.

Example 4 - With HADOOP-FS using Kerberos:

```
{
  "connectorName": "Hadoop_Hdfs_connector",
  "source": {
    "type": "HADOOP-FS",
    "properties": {
      "server": "hadoop-server-hostname.co",
      "path": "/path/to/source/hdfs/files",
      "protocol": "hdfs",
      "port": 8020,
      "principal_name": "principal_name/somedomain@REALM.COM"
    }
  },
  "target": {
    "type": "HADOOP-FS",
    "properties": {
      "server": "hadoop-server-hostname.co",
      "path": "/path/to/target/hdfs/files",
      "protocol": "hdfs",
      "port": 8020,
      "principal_name": "principal_name/somedomain@REALM.COM"
    }
  }
}
```

Overview of the parameters:

- **type**: Refers to the type of connector you are creating. In the above example, we are using the type as "HADOOP-FS" which refers to "HDFS", indicating that all the source and target files are available within the HDFS.
- **properties**: Holds the server and path values.
- **server**: Points to the Hadoop server.
- **path**: The path to the HDFS source/target location.
- Additional parameters:

    - **principal_name**: This is required for the Kerberos authentication, the principal name must have access assigned in the user provided keytab entry.
    - **protocol**: User can set this protocol as per their Hadoop configurations.
    - **port**: This is HDFS port, by default it will be 8020.

Example 5 - With HADOOP-DB using Kerberos:

```
{
  "connectorName": "Hadoop_Hive_connector",
  "source": {
    "type": "HADOOP-DB",
    "properties": {
      "server": "hadoop-server-hostname.co",
      "database_name": "hive",
      "database_type": "hive",
      "database_port": "10000",
      "principal_name": "principal_name/somedomain@REALM.COM",
      "protocol": "hdfs"
    }
  },
  "target": {
    "type": "HADOOP-FS",
    "properties": {
      "server": "hadoop-server-hostname.co",
      "path": "/path/to/hdfs/files",
      "protocol": "hdfs",
      "port": 8020,
      "principal_name": "principal_name/somedomain@REALM.COM"
    }
  }
}
```

Overview of the parameters:

- **type**: Refers to the type of connector you are creating. In the above example, we are using the type as “HADOOP-DB” which refers to “HDFS via Hive database” for source and “HADOOP-FS” for target.
- **properties**: Holds the server and path values.
- **server**: Points to the Hadoop server.
- **path**: The path to the HDFS source/target location.
- Additional parameters (in case AWS credentials need to be passed for separately):
    - **principal_name**: This is required for the kerberos authentication, the principal name must have access assigned in the user provided keytab entry.
    - **database_type**: Currently we only support Hive as a supported database.
    - **database_name**: Hive database name.
    - **database_port**: This is a thrift port for accessing the database.
    - **protocol**: User can set this protocol as per their hadoop configurations.
    - **port**: This is HDFS port, by default it will be 8020.

File connector different Responses:

Example 1

```
{
 "source": {
  "type": "AWS",
  "properties": {
      "server": "S3",
      "path": "s3_bucket_source/sub_folder"
  }
 },
 "target": {
  "type": "AWS",
  "properties": {
      "server": "S3",
      "path": "s3_bucket_target/sub_folder"
  }
 }
}
```

Example 2

```
{
 "source": {
  "type": "AWS",
  "properties": {
    "server": "S3",
    "path": "s3_bucket_source/sub_folder",
    "aws_region": "us-east-1",
    "aws_access_key_id": "AKIA********",
    "aws_secret_access_key": "x2IX********",
    "aws_role_arn": "56436882398"
  }
 },
 "target": {
  "type": "AWS",
  "properties": {
    "server": "S3",
    "path": "s3_bucket_target/sub_folder",
    "aws_region": "us-east-1",
    "aws_access_key_id": "AKIA********",
    "aws_secret_access_key": "x2IX********",
    "aws_role_arn": "56436882398"
  }
 }
}
```

Example 3

```
{
  "id": 1,
  "connectorName": "Parquet_Connector_FS",
  "source": {
    "type": "FS",
    "properties": {
      "server": "local",
      "path": "/mnt/source"
    }
  },
  "target": {
    "type": "FS",
    "properties": {
      "server": "local",
      "path": "/mnt/target"
    }
  }
}
```

Example 4

```
{
  "connectorName": "Hadoop_Hdfs_connector_2",
  "source": {
    "type": "HADOOP-FS",
    "properties": {
      "server": "hadoop-server-hostname.co",
      "port": "8020",
      "path": "/source/file/location",
      "principal_name": "PRINCIPAL_NAME@DOMAIN",
      "protocol": "hdfs"
    }
  },
  "target": {
    "type": "HADOOP-FS",
    "properties": {
      "server": "hadoop-server-hostname.co",
      "path": "/target/file/location",
      "protocol": "hdfs",
      "port": "8020",
      "principal_name": "PRINCIPAL_NAME@DOMAIN"
    }
  }
}
```

Example 5

```
{
  "connectorName": "Hadoop_Hive_connector_2",
  "source": {
    "type": "HADOOP-DB",
    "properties": {
      "server": "hadoop-server-hostname.co",,
      "database_name": "default",
      "database_type": "hive",
      "database_port": "10000",
      "principal_name": "PRINCIPAL_NAME@DOMAIN",
      "protocol": "hdfs"
    }
  },
  "target": {
    "type": "HADOOP-FS",
    "properties": {
      "server": "hadoop-server-hostname.co",,
      "path": "/target/file/location",
      "protocol": "hdfs",
      "port": "8020",
      "principal_name": "PRINCIPAL_NAME@DOMAIN"
    }
  }
}
```

StructuredDataFormat APIs

This functionality is specifically needed when there is a requirement to mask XML/JSON data contained within database columns. These endpoints require a `file_upload_ref` that can be generated via the `POST /file-upload` endpoint wherein you must upload the format of the XML/JSON data to be masked. For more information, refer to the Hyperscale Compliance API documentation.

POST /structured-data-format (create structured-data-format for a column)

Request:

```
{
  "file_upload_ref": "delphix-file://upload/f_XXXX/XXX.xml",
  "doc_type": "XML",
  "masking_inventory_paths": [
    {
      "path": "/catalog/book/author",
      "domain_name": "NULL_SL",
      "algorithm_name": "dlpx-core:FullName"
    },
    {
      "path": "/catalog/book/isbn",
      "domain_name": "ADDRESS",
      "algorithm_name": "dlpx-core:CM Alpha-Numeric"
    }
  ]
}
```

Response:

```
{
  "structured_data_format_id":1,
  "file_upload_ref": "delphix-file://upload/f_XXXX/XXX.xml",
  "doc_type": "XML",
  "masking_inventory_paths": [
    {
      "path": "/catalog/book/author",
      "domain_name": "NULL_SL",
      "algorithm_name": "dlpx-core:FullName"
    },
    {
      "path": "/catalog/book/isbn",
      "domain_name": "ADDRESS",
      "algorithm_name": "dlpx-core:CM Alpha-Numeric"
    }
  ]
}
```

DataSets API

- Table and schema names are case-sensitive.
- For the MSSQL connector, it's recommended to provide filter_key if the unload_split count is more than 1, and the table does not contain a primary/unique key, else data will be unloaded through a sequential approach which may be slower. If filter_key is not provided and the table includes a primary/unique key then Hyperscale will scan the table to fetch the key automatically.
- The dataset date format should be the same as the environment variable date format value.
- In one dataset, all the inventories should use the same date format for all date formats.

POST /data-sets (create dataset for Hyperscale Compliance)

Alternatively, you can create or update the dataset using payload in a file with the below endpoints:

- `POST /data-sets/file-upload`
- `PUT /data-sets/file-upload/{dataSetId}`

The above endpoints require a `file_upload_ref` that can be generated via the `POST /file-upload` endpoint. For more information, refer to the Hyperscale Compliance API documentation.

Request (with single table):

```
{
"connector_id": 1,
"data_info": [
{
"source": {
    "schema_name": "SCHEMA_1",
    "table_name": "TABLE_1",
    "unload_split": 4
},
"target": {
    "schema_name": "SCHEMA_1_TARGET",
    "table_name": "TABLE_1_TARGET",
    "stream_size": 65536
},
"masking_inventory": [
    {
    "field_name": "FIRST_NAME",
    "domain_name": "FIRST_NAME",
    "algorithm_name": "FirstNameLookup"
    },
    {
    "field_name": "LAST_NAME",
    "domain_name": "LAST_NAME",
    "algorithm_name": "LastNameLookup"
    },
    {
    "field_name": "XmlData",
    "structured_data_format_id": 1
    }
]
}
]
}
```

Response (with single table):

```
{
"id": 1,
"connector_id": 1,
"data_info": [
{
"source": {
    "schema_name": "SCHEMA_1",
    "table_name": "TABLE_1",
    "unload_split": 4
},
"target": {
    "schema_name": "SCHEMA_1",
    "table_name": "TABLE_1",
    "stream_size": 65536
},
"masking_inventory": [
    {
    "field_name": "FIRST_NAME",
    "domain_name": "FIRST_NAME",
    "algorithm_name": "FirstNameLookup"
    },
    {
    "field_name": "LAST_NAME",
    "domain_name": "LAST_NAME",
    "algorithm_name": "LastNameLookup"
    },
    {
    "field_name": "XmlData",
    "structured_data_format_id": 1
    }
]
}
]
}
```

Request (with single table & filter_key in the source):

```
{
"connector_id": 1,
"data_info": [
{
"source": {
    "schema_name": "SCHEMA_1",
    "table_name": "TABLE_1",
    "unload_split": 4,
    "filter_key": "PKID"
},
"target": {
    "schema_name": "SCHEMA_1_TARGET",
    "table_name": "TABLE_1_TARGET",
    "stream_size": 65536
},
"masking_inventory": [
    {
    "field_name": "FIRST_NAME",
    "domain_name": "FIRST_NAME",
    "algorithm_name": "FirstNameLookup"
    },
    {
    "field_name": "LAST_NAME",
    "domain_name": "LAST_NAME",
    "algorithm_name": "LastNameLookup"
    },
    {
    "field_name": "XmlData",
    "structured_data_format_id": 1
    }
]
}
]
}
```

Response (with single table & filter_key in the source):

```
{
"id": 1,
"connector_id": 1,
"data_info": [
{
"source": {
    "schema_name": "SCHEMA_1",
    "table_name": "TABLE_1",
    "unload_split": 4,
    "filter_key": "PKID"
},
"target": {
    "schema_name": "SCHEMA_1",
    "table_name": "TABLE_1",
    "stream_size": 65536
},
"masking_inventory": [
    {
    "field_name": "FIRST_NAME",
    "domain_name": "FIRST_NAME",
    "algorithm_name": "FirstNameLookup"
    },
    {
    "field_name": "LAST_NAME",
    "domain_name": "LAST_NAME",
    "algorithm_name": "LastNameLookup"
    },
    {
    "field_name": "XmlData",
    "structured_data_format_id": 1
    }
]
}
]
}
```

Request (with multiple tables):

```
{
"connector_id": 1,

"data_info": [
{
"source": {
"unload_split": 2,
"schema_name": "DLPXDBORA",
"table_name": "test_multi_0"
},
"target": {
"stream_size": 65536,
"schema_name": "DLPXDBORA",
"table_name": "test_multi_0"
},
"masking_inventory": [
{
"field_name": "col_VARCHAR",
"domain_name": "FIRST_NAME",
"algorithm_name": "FirstNameLookup"
},
{
"field_name": "XmlData",
"structured_data_format_id": 1
}
]
},
{
"source": {
"unload_split": 2,
"schema_name": "DLPXDBORA",
"table_name": "test_multi_1"
},
"target": {
"stream_size": 65536,
"schema_name": "DLPXDBORA",
"table_name": "test_multi_1"
},
"masking_inventory": [
{
"field_name": "COL_TIMESTAMP",
"domain_name": "DOB",
"algorithm_name": "DateShiftVariable",
"date_format": "yyyy-MM-dd HH:mm:ss.SSS" -->(optional field, this needs to be added
only while working with date/time masking)
}
]
}
]
}
```

Response (with multiple tables):

```
{
"id": 1,
"connector_id": 1,
"data_info": [
{
"source": {
"unload_split": 2,
"schema_name": "DLPXDBORA",
"table_name": "test_multi_0"
},
"target": {
"stream_size": 65536,
"schema_name": "DLPXDBORA",
"table_name": "test_multi_0"
},
"masking_inventory": [
{
"field_name": "col_VARCHAR",
"domain_name": "FIRST_NAME",
"algorithm_name": "FirstNameLookup"
},
{
"field_name": "XmlData",
"structured_data_format_id": 1
}
]
},
{
"source": {
"unload_split": 2,
"schema_name": "DLPXDBORA",
"table_name": "test_multi_1"
},
"target": {
"stream_size": 65536,
"schema_name": "DLPXDBORA",
"table_name": "test_multi_1"
},
"masking_inventory": [
{
"field_name": "COL_TIMESTAMP",
"domain_name": "DOB",
"algorithm_name": "DateShiftVariable",
"date_format": "yyyy-MM-dd HH:mm:ss.SSS"
}
]
}
]
}
```

File Connector with Delimited Files DataSet request:

```
{
  "connector_id": 1,
  "data_info": [
    {
      "source": {
        "delimiter": "|",
        "endOfRecord": "\n",
        "enclosure": "\"",
        "enclosureEscapeCharacter": "\\",
        "escapeEnclosureEscapeCharacter": false,
        "unique_source_files_identifier": "file_identifier1",
        "has_headers": false,
        "unload_split": 100,
        "source_files": [
          "file1.txt",
          "file2.txt"
        ]
      },
      "target": {
        "perform_join": true
      },
      "masking_inventory": [
        {
          "field_name": "f3",
          "domain_name": "FIRST_NAME",
          "algorithm_name": "FirstNameLookup"
        },
        {
          "field_name": "f5",
          "domain_name": "LAST_NAME",
          "algorithm_name": "LastNameLookup"
        }
      ]
    }
  ]
}
```

In case of staging push, the split count must always be 1. The source files should not have headers (or columns).

The source DataSet consists of the following parameters:

1. **delimiter:** The single character length delimiter used in source files.
2. **endOfRecord** : The end of line character, currently we only support `` `\n` `` , `` `\r` `` & `` `\r\n` `` .
3. **enclosure**: The single character length quote character used in the source files.
4. **enclosureEscapeCharacter:** The escape character used to escape quote characters.
5. **unique_source_files_identifier**: This is the source key that maps the load-service and masking-service data sets with the unload-service data set. Please ensure that this value is different for each item in the DataSet data_info list.
6. **has_headers:** A flag that indicates if the character source files have header column names or not.
    a. If set to **true**, format files with the same column names are created and the same can be used for the masking inventory.
    b. If set to **false**, the column names of pattern **f0, f1, f2**, and so on are used to create the format files for delimited file masking. When adding the masking inventory please make sure to use **field_name** with values **f0, f1, f2,** and so on.
7. **unload_split**: The number of splits that the files in the source_files list have to be split into. Please ensure that the split number is not too small nor too big for better overall performance.
8. **source_files:** List of all source files that need to be masked and adhere to the following rules:
    a. All files should have the same delimiter character and other helper characters.
    b. All files should have the same number of columns and the same column names if it has a header line.
    c. Supported input formats:
        i. Relative path to the file - relative to the configured connector source path. Examples (considering that the source path within the container is /mnt/source):
            1. If the absolute path of the file was `` `/mnt/source/file1.txt` `` , then the source_files list should only contain `` `files1.txt` `` .
            2. If the absolute path of the file was `` `/mnt/source/some_dir/file1.txt` `` , then the source_files list should contain the value `` `some_dir/file1.txt` `` .
        ii. Blob pattern - the path to all files matching a blob. Examples:
            1. If we want to mask files `` `/mnt/source/file1.txt` `` and `` `/mnt/source/file2.txt` `` , then the source_files list can contain the value `` `file*.txt` `` .
            2. If we want to mask all files within a directory, we can add `` `directory/*` `` to the source_files list.

9. **Optional - null_values_as_string:** Is a list of all null values that are represented as a string in data. For example, “null”, “NULL”, “None”, ““ (empty string), etc. The default value is a empty list ([]), i.e. there exists no null-string in data.

```
{
  "source": {
    ...
    "null_values_as_string": [""]
  }
}
```

The target DataSet consists of the following parameters:

1. perform_join: A flag to check if the load-service joins back all masked files or not.
    a. If set to true, the load-service will join back masked files and keep the same relative file location structure in the target path (considering that the target path within the container is /mnt/target).
        i. If the input was `file1.txt` , then the output file will be `/mnt/target/file1.txt`
        ii. If the input was `some_dir/file1.txt` , then the output file will be `/mnt/target/some_dir/file1.txt` .
    b. If set to false, the split masked files will be placed in the same relative file location structure in the target path with a split number appended to the file name.
        i. If the input was `file1.txt` , then the output files will be `/mnt/target/file1_0.txt` , `/mnt/target/file1_1.txt` , and so on.
        ii. If the input was `some_dir/file1.txt`, then the output files will be `/mnt/target/some_dir/file1_0.txt` , `/mnt/target/some_dir/file1_1.txt` , and so on.

The masking inventory remains the same, except when there are no column names in the delimited files and the connector assumes that the column names would be `f0`, `f1`, `f2` , and so on. These are **field names** that have to be used to fill up the masking inventory (as shown in the example above).

File Connector Delimited Files DataSet response:

```
{
 "id": 1,
  "connector_id": 1,
  "data_info": [
    {
      "source": {
        "delimiter": "|",
        "endOfRecord": "\n",
        "enclosure": "\"",
        "enclosureEscapeCharacter": "\\",
        "escapeEnclosureEscapeCharacter": false,
        "unique_source_files_identifier": "file_identifier1",
        "has_headers": false,
        "unload_split": 100,
        "source_files": [
          "file1.txt",
          "file2.txt"
        ]
      },
      "target": {
        "perform_join": true
      },
      "masking_inventory": [
        {
          "field_name": "f3",
          "domain_name": "FIRST_NAME",
          "algorithm_name": "FirstNameLookup"
        },
        {
          "field_name": "f5",
          "domain_name": "LAST_NAME",
          "algorithm_name": "LastNameLookup"
        }
      ]
    }
  ]
}
```

File Connector with Parquet DataSet request:

```
{
 "connector_id": 1,
 "data_info": [
  {
   "source": {
     "unique_source_files_identifier": "file_identifier1",
     "unload_split": 100,
     "file_type": "parquet",
     "source_files": [
       "relative_folder1/file1.parquet",
       "relative_folder2/file2.parquet"
     ]
   },
   "target": {
     "perform_join": true
   },
   "masking_inventory": [
     {
       "field_name": "colume_name1",
       "domain_name": "FIRST_NAME",
       "algorithm_name": "FirstNameLookup"
     },
     {
       "field_name": "colume_name2",
       "domain_name": "LAST_NAME",
       "algorithm_name": "LastNameLookup"
     }
```

```
      ]
    }
  ]
}
```

> The **Hyperscale Parquet Profiler** can be used to analyze the source parquet files and help generate the DataSet with the masking inventory. To know more, visit the Parquet Profiler documentation.

The source dataset consists of the following parameters:

1. **unique_source_files_identifier**: This is the source key that maps the load-service and masking-service data sets with the unload-service data set. Please ensure that this value is different for each item in the DataSet data_info list.
2. **unload_split**: The number of splits that the files in the source_files list have to be split into. Please ensure that the split number is not too small nor too big for better overall performance.
3. **file_type**: The file type should be set to "parquet".
4. **source_files**: List of all source files that need to be masked and adhere to the following rules:
    a. All files should have the same delimiter character and other helper characters.
    b. All files should have the same number of columns and the same column names if it has a header line.
    c. Supported input formats:
        i. Relative path to the file, relative to the configured connector source path. Examples (considering that the source path within the container is s3_bucket/source):
            1. If the file's absolute path was `s3_bucket/source/file1.parquet`, then the source_files list should only contain `files1.parquet`.
            2. If the absolute path of the file was `s3_bucket/source/some_dir/file1.parquet`, then the source_files list should contain the value `some_dir/file1.parquet`.
        ii. Blob pattern, the path to all files matching a blob. Examples:
            1. If we want to mask files `s3_bucket/source/file1.parquet` and `s3_bucket/source/file2.parquet`, then the source_files list can contain the value `file*.parquet`.
            2. If we want to mask all files within a directory, we can add `directory/*` to the source_files list.

The target dataset consists of the following parameters:

1. **perform_join**: A flag to check if the load-service joins back all masked files or not.
    a. If set to true, the load-service will join back masked files and keep the same relative file location structure in the target path (considering that the target path within the container is s3_bucket/target).
        i. If the input was `file1.parquet`, then the output file will be `s3_bucket/target/file1.parquet`
        ii. If the input was `some_dir/file1.parquet`, then the output file will be `s3_bucket/target/some_dir/file1.parquet`.
    b. If set to false, the split masked files will be placed in the same relative file location structure in the target path with a split number appended to the file name.

i. If the input was `file1.parquet`, then the output files will be `s3_bucket/target/file1_0.parquet`, `s3_bucket/target/file1_1.parquet`, and so on.

ii. If the input was `some_dir/file1.txt`, then the output files will be `s3_bucket/target/some_dir/file1_0.parquet`, `s3_bucket/target/some_dir/file1_1.parquet`, and so on.

File Connector with Parquet DataSet response:

```
{
 "connector_id": 1,
 "data_info": [
  {
   "source": {
     "unique_source_files_identifier": "file_identifier1",
     "unload_split": 100,
     "file_type": "parquet",
     "source_files": [
       "<relative_path>/file1.parquet",
       "<relative_path>/file2.parquet"
     ]
   },
   "target": {
     "perform_join": true
   },
   "masking_inventory": [
     {
       "field_name": "colume_name1",
       "domain_name": "FIRST_NAME",
       "algorithm_name": "FirstNameLookup"
     },
     {
       "field_name": "colume_name2",
       "domain_name": "LAST_NAME",
       "algorithm_name": "LastNameLookup"
     }
   ]
  }
 ]
}
```

File Connector with Hive tables DataSet request:

```
{
  "connector_id": 1,
  "data_info": [
    {
      "source": {
        "delimiter": "|",
        "endOfRecord": "\n",
        "enclosure": "\"",
        "enclosureEscapeCharacter": "\\",
        "escapeEnclosureEscapeCharacter": false,
        "unique_source_files_identifier": "file_identifier1",
        "has_headers": false,
        "unload_split": 100,
        "source_files": [
          "table1",
          "table2:part1"
        ]
      },
      "target": {
        "perform_join": true
      },
      "masking_inventory": [
        {
          "field_name": "f3",
          "domain_name": "FIRST_NAME",
          "algorithm_name": "FirstNameLookup"
        },
        {
          "field_name": "f5",
          "domain_name": "LAST_NAME",
          "algorithm_name": "LastNameLookup"
        }
      ]
    }
  ]
}
```

The source dataset consists of the following parameters:

1. **unique_source_files_identifier**: This is the source key that maps the load-service and masking-service data sets with the unload-service data set. Please ensure that this value is different for each item in the DataSet data_info list.
2. **unload_split**: The number of splits that the files in the source_files list have to be split into. Please ensure that the split number is not too small nor too big for better overall performance.
3. **source_files**: List of all tables with or without partitions.

MongoDB DataSet request:

```
{
  "connector_id": 1,
  "data_info": [
    {
      "source": {
        "database_name": "SRC_DB",
        "collection_name": "SRC_COLLECTION",
        "unload_split": 10
      },
      "target": {
        "database_name": "TGT_DB",
        "collection_name": "TGT_COLLECTION",
        "drop_collection" : "No"  --> (Optional. If set to No, target collection will
not be dropped before loading)
      },
      "masking_inventory": [
        {
          "field_name": "$[*]['relationships'][*]['person']['first_name']",
          "domain_name": "FIRST_NAME",
          "algorithm_name": "dlpx-core:FullName"
        },
        {
          "field_name": "$[*]['address']",
          "domain_name": "ADDRESS",
          "algorithm_name": "dlpx-core:CM Alpha-Numeric"
        }
      ]
    }
  ]
}
```

- **unload_split**: The number of splits that the MongoDB collection has to be split into. You must make sure that the split number is neither too small nor too big for a better overall performance.
- **drop_collection**: It is optional. To leverage Reduced Privilege Operations, you must set property drop_collection to No. After The property is set, the connector will no longer require `clusterAdmin` and `clusterMonitor` privileges. Check the pre-requisites section for more details.
- **masking_inventory**: The masking inventory for MongoDB collection can be generated by using the MongoDB Profiler service, which will call the dataset API with the generated masking_inventory and create the dataset. MongoDB Profiler artifact includes a README file that provides detailed usage instructions.

MongoDB DataSet response:

```
{
  "connector_id": 1,
  "data_info": [
    {
      "source": {
        "database_name": "SRC_DB",
        "collection_name": "SRC_COLLECTION",
        "unload_split": 10
      },
      "target": {
        "database_name": "TGT_DB",
        "collection_name": "TGT_COLLECTION"
      },
      "masking_inventory": [
        {
          "field_name": "$[*]['relationships'][*]['person']['first_name']",
          "domain_name": "FIRST_NAME",
          "algorithm_name": "dlpx-core:FullName"
        },
        {
          "field_name": "$[*]['address']",
          "domain_name": "ADDRESS",
          "algorithm_name": "dlpx-core:CM Alpha-Numeric"
        }
      ]
    }
  ]
}
```

Algorithm and Domain names to be provided in the Data Set request should be used from Continuous Compliance Engine. The Continuous Compliance Engine APIs that could be used to get these names are:

1. Get all algorithms (`GET /algorithms`) for Algorithm Names. Sample Endpoint: https://maskingdocs.delphix.com/maskingApiEndpoints/5_1_15_maskingApiEndpoints.html#getAllAlgorithms
2. Get all domains (`GET /domains`) for Domain Names. Sample Endpoint: https://maskingdocs.delphix.com/maskingApiEndpoints/5_1_15_maskingApiEndpoints.html#getAllDomains

To check about extra parameters that need to be provided in the Data Set request for Date and Multi Column Algorithms, refer to Model `DataSet_masking_inventory` on the Hyperscale Compliance API Documentation page available in the API Reference section of this Documentation.
Alternatively, you can create/update the dataset using payload in a file with below end-points:

1. `POST /data-sets/file-upload`
2. `PUT /data-sets/file-upload/{dataSetId}`

Above endpoints requires a file_upload_ref, which can be generated via `POST /file-upload` endpoint. See Hyperscale Compliance API.

Snowflake DataSet request:

```
{
  "connector_id": 1,
  "data_info": [
    {
      "source": {
        "warehouse_name": "SRC_TEST_WAREHOUSE",
        "schema_name": "SRC_TEST_SCHEMA",
        "database_name": "SRC_TEST_WAREHOUSE",
        "table_name": "SRC_TEST_TABLE",
        "stage_name": "SRC_EXT_STAGE",
        "max_file_size": "104857600"  --> (Optional. If removed from the payload,
snowflake will create a default file of size 16MB )
      },
      "target": {
        "warehouse_name": "TGT_TEST_WAREHOUSE",
        "schema_name": "TGT_TEST_SCHEMA",
        "database_name": "TGT_TEST_WAREHOUSE",
        "table_name": "TGT_TEST_TABLE"
      },
      "masking_inventory": [
        {
          "field_name": "Name",
          "domain_name": "NULL_SL",
          "algorithm_name": "dlpx-core:FullName"
        },
        {
          "field_name": "Address",
          "domain_name": "ADDRESS",
          "algorithm_name": "dlpx-core:CM Alpha-Numeric"
        },
        {
          "field_name": "Details",
          "structured_data_format_id": 1
        }
      ]
    }
  ]
}
```

- **max_file_size (in bytes):** It is optional. The maximum size for each file is set using the MAX_FILE_SIZE copy option. The default value (if not provided) is 16777216 (16 MB) but can be increased to accommodate larger files. The maximum file size supported is 5 GB for Amazon S3. Note that the actual file size and number of files unloaded are determined by the total amount of data and number of nodes available for parallel processing. Read more.
- **stage_name**: A snowflake external stage area that will be used for unloading data from snowflake tables into CSV files and loading these masked CSV data files back into Snowflake tables. The Amazon S3 cloud storage services are currently supported with the snowflake external stage.
- **masking_inventory**: The masking inventory for Snowflake tables can be generated by using the Snowflake Profiler service, which will call the dataset API with the generated masking_inventory and create the dataset. Snowflake Profiler artifact includes a README file that provides detailed usage instructions.

Snowflake DataSet response:

```
{
  "id": 1,
  "connector_id": 1,
  "data_info": [
    {
      "source": {
        "max_file_size": "104857600",
        "warehouse_name": "SRC_TEST_WAREHOUSE",
        "database_name": "SRC_TEST_WAREHOUSE",
        "stage_name": "SRC_EXT_STAGE",
        "schema_name": "SRC_TEST_SCHEMA",
        "table_name": "SRC_TEST_TABLE"
      },
      "target": {
        "warehouse_name": "TGT_TEST_WAREHOUSE",
        "database_name": "TGT_TEST_WAREHOUSE",
        "schema_name": "TGT_TEST_SCHEMA",
        "table_name": "TGT_TEST_TABLE"
      },
      "masking_inventory": [
        {
          "field_name": "Name",
          "domain_name": "NULL_SL",
          "algorithm_name": "dlpx-core:FullName"
        },
        {
          "field_name": "Address",
          "domain_name": "ADDRESS",
          "algorithm_name": "dlpx-core:CM Alpha-Numeric"
        }
      ]
    }
  ]
}
```

Snowflake Job request:

The Snowflake job does not require the source_configs and target_configs objects.

```
{
  "name": "job_1",
  "masking_engine_ids": [
    1
  ],
  "data_set_id": 1,
  "app_name_prefix": "app",
  "env_name_prefix": "env",
  "consider_continuous_compliance_warning_event_as": "SUCCESS",
  "retain_execution_data": "NO",
  "masking_job_config": {
    "max_memory": 1024,
    "min_memory": 0,
    "description": "Job created by Snowflake Hyperscale Masking",
    "feedback_size": 100000,
    "stream_row_limit": 10000,
    "num_input_streams": 1
  }
}
```

Snowflake Job response:

```
{
  "name": "job_1",
  "masking_engine_ids": [
    1
  ],
  "data_set_id": 1,
  "app_name_prefix": "app",
  "env_name_prefix": "env",
  "consider_continuous_compliance_warning_event_as": "SUCCESS",
  "retain_execution_data": "NO",
  "masking_job_config": {
    "max_memory": 1024,
    "min_memory": 0,
    "description": "Job created by Snowflake Hyperscale Masking",
    "feedback_size": 100000,
    "stream_row_limit": 10000,
    "num_input_streams": 1
  }
}
```

Jobs API

POST /jobs (Create a Hyperscale Compliance job)

```
{
  "name": "job_1",
  "masking_engine_ids": [
    1
  ],
  "data_set_id": 1,
  "app_name_prefix": "app",
  "env_name_prefix": "env",
  "consider_continuous_compliance_warning_event_as": "SUCCESS",
  "retain_execution_data": "NO",
  "source_configs": {
    "max_concurrent_source_connection": 30
  },
  "target_configs": {
    "max_concurrent_target_connection": 30,
    "parallelism_degree": 15
  },
  "masking_job_config": {
    "max_memory": 1024,
    "min_memory": 0,
    "description": "Job created by Hyperscale Masking",
    "feedback_size": 100000,
    "stream_row_limit": 10000,
    "num_input_streams": 1
  }
}
```

- For more information on `retain_execution_data flag`, see Cleaning Up Execution Data.

- In the case of Oracle, set ***parallelism_degree*** in the ***target_config***to use the degree of parallelism while re-creating the indexes in the post-load step.
    - **Set {"parallelism_degree" :0 }:** Use unmodified DDL provided by Oracle.
    - **Set {"parallelism_degree": -1}:** Remove any parallel or nonparallel clause from the DDL.
    - **Set {"parallelism_degree": any positive vale}:** Remove existing parallel degree or nonparallel clause and add Parallel <parallelism_degree> to the DDL.
- Also in Oracle you can set `enable_all_constraints_with_novalidate` in the ***target_config*** to force (with true value) to enable constraints with **NOVALIDATE** in the post-load step.
    a. **Set {"enable_all_constraints_with_novalidate": false}:** This is default value, and NOVALIDATE value will be preserved as found at pre-load step.
    b. **Set {"enable_all_constraints_with_novalidate": true}:** with true value, constraints will be enforced to enable with **NOVALIDATE**, here earlier state of **VALIDATE** will be ignored.

Response:

Below properties are only applicable to Oracle Datasource.

```
{
  "id": 1,
  "name": "Test_Job",
  "masking_engine_ids": [
    1,
    2,
    3
  ],
  "data_set_id": 1,
  "app_name_prefix": "Test_App",
  "env_name_prefix": "Test_Env",
  "consider_continuous_compliance_warning_event_as": "SUCCESS",
  "retain_execution_data": "NO",
  "source_configs": {
    "max_concurrent_source_connection": 30
  },
  "target_configs": {
    "max_concurrent_target_connection": 30
  },
  "masking_job_config": {
    "max_memory": 1024,
    "min_memory": 1024,
    "description": "Job created by Hyperscale Masking",
    "feedback_size": 100000,
    "stream_row_limit": 10000,
    "num_input_streams": 1
  }
}
```

File Connector Job request:

In File Connector jobs, **source_configs** and **target_configs** objects can be used to configure various job specific parameters.

```
{
  "name": "job_1",
  "masking_engine_ids": [
    1
  ],
  "data_set_id": 1,
  "app_name_prefix": "app",
  "env_name_prefix": "env",
  "consider_continuous_compliance_warning_event_as": "SUCCESS",
  "retain_execution_data": "NO",
  "masking_job_config": {
    "max_memory": 1024,
    "min_memory": 0,
    "description": "Job created by Hyperscale Masking",
    "feedback_size": 100000,
    "stream_row_limit": 10000,
    "num_input_streams": 1
  },
"source_configs": {
     "writer_type": "pyspark",
     "unload_spark_driver_memory": "4g",
     "unload_spark_driver_cores": 2,
     "unload_spark_log_level": "INFO",
     "max_worker_threads_per_job": 512,
     "staging_push": false,
     "user_provided_format_files": false,
     "skip_pyspark_resource_validation": false
  },
  "target_configs": {
     "writer_type": "pyspark",
     "load_spark_driver_memory": "4g",
     "load_spark_driver_cores": 2,
     "load_spark_log_level": "INFO",
     "staging_push": false
  }
}
```

File Connector Job response:

```
{
 "id": 1,
  "name": "job_1",
  "masking_engine_ids": [
    1
  ],
  "data_set_id": 1,
  "app_name_prefix": "app",
  "env_name_prefix": "env",
  "consider_continuous_compliance_warning_event_as": "SUCCESS",
  "retain_execution_data": "NO",
  "masking_job_config": {
    "max_memory": 1024,
    "min_memory": 0,
    "description": "Job created by Hyperscale Masking",
    "feedback_size": 100000,
    "stream_row_limit": 10000,
    "num_input_streams": 1
  },
"source_configs": {
     "writer_type": "pyspark",
     "unload_spark_driver_memory": "4g",
     "unload_spark_driver_cores": 2,
     "unload_spark_log_level": "INFO",
     "max_worker_threads_per_job": 512,
     "staging_push": false,
     "user_provided_format_files": false,
     "skip_pyspark_resource_validation": false
  },
  "target_configs": {
     "writer_type": "pyspark",
     "load_spark_driver_memory": "4g",
     "load_spark_driver_cores": 2,
     "load_spark_log_level": "INFO",
     "staging_push": false
  }
}
```

MongoDB Job request:

The MongoDB job does not require the **source_configs** and **target_configs** objects.

```
{
  "name": "job_1",
  "masking_engine_ids": [
    1
  ],
  "data_set_id": 1,
  "app_name_prefix": "app",
  "env_name_prefix": "env",
  "consider_continuous_compliance_warning_event_as": "SUCCESS",
  "retain_execution_data": "NO",
  "masking_job_config": {
    "max_memory": 1024,
    "min_memory": 0,
    "description": "Job created by MongoDB Hyperscale Masking",
    "feedback_size": 100000,
    "stream_row_limit": 10000,
    "num_input_streams": 1
  }
}
```

MongoDB Job response:

```
{
  "name": "job_1",
  "masking_engine_ids": [
    1
  ],
  "data_set_id": 1,
  "app_name_prefix": "app",
  "env_name_prefix": "env",
  "consider_continuous_compliance_warning_event_as": "SUCCESS",
  "retain_execution_data": "NO",
  "masking_job_config": {
    "max_memory": 1024,
    "min_memory": 0,
    "description": "Job created by MongoDB Hyperscale Masking",
    "feedback_size": 100000,
    "stream_row_limit": 10000,
    "num_input_streams": 1
  }
}
```

JobExecution API

POST /executions (Create an execution of a Hyperscale job)

Request:

```
{
    "job_id": 1
}

Response: (Immediate response will be like below. Realtime response can be fetched
using GET /executions/{execution_id} endpoint)

{
  "id": 124,
  "job_id": 38,
  "status": "RUNNING",
  "create_time": "2023-05-04T12:43:03.444964",
  "tasks": [
    {
      "name": "Unload"
    },
    {
      "name": "Masking"
    },
    {
      "name": "Load"
    },
    {
      "name": "Post Load"
    }
  ]
}
```

GET /executions/{id}/summary (Returns the job execution by execution id in summarized format)

```
{
  "id": 72,
  "job_id": 5,
  "status": "SUCCEEDED",
  "create_time": "2022-12-18T13:38:43.722917",
  "end_time": "2022-12-18T13:43:16.554603",
  "total_objects": 4,
  "total_rows": 16,
  "tasks": [
    {
      "name": "Unload",
      "status": "SUCCEEDED",
      "start_time": "2022-12-18T13:38:44.184296",
      "end_time": "2022-12-18T13:38:54.972883",
      "received_objects": 4,
      "succeeded_objects": 4,
      "failed_objects": 0,
      "processing_objects": 0,
      "processed_rows": 16,
      "total_rows": 16
    },
    {
      "name": "Masking",
      "status": "SUCCEEDED",
      "start_time": "2022-12-18T13:38:51.979725",
      "end_time": "2022-12-18T13:42:58.569202",
      "received_objects": 4,
      "succeeded_objects": 4,
      "failed_objects": 0,
      "processing_objects": 0,
      "processed_rows": 16,
      "total_rows": 16
    },
    {
      "name": "Load",
      "status": "SUCCEEDED",
      "start_time": "2022-12-18T13:40:39.350857",
      "end_time": "2022-12-18T13:43:12.966492",
      "received_objects": 4,
      "succeeded_objects": 4,
      "failed_objects": 0,
      "processing_objects": 0,
      "processed_rows": 16,
      "total_rows": 16
    },
    {
      "name": "Post Load",
      "status": "SUCCEEDED",
      "start_time": "2022-12-18T13:43:12.981490",
      "end_time": "2022-12-18T13:43:15.764366",
      "metadata": [
        {
          "type": "Constraints",
          "total": 20,
          "processed": 20,
          "status": "SUCCESS",
          "start_time": "2022-12-18T13:43:12.981490",
          "end_time": "2022-12-18T13:43:15.764366"
        },
        {
          "type": "Indexes",
          "total": 10,
          "processed": 10,
          "status": "SUCCESS",
          "start_time": "2022-12-18T13:43:12.981490",
          "end_time": "2022-12-18T13:43:15.764366"
        },
        {
          "type": "Triggers",
          "total": 5,
          "processed": 5,
          "status": "SUCCESS",
          "start_time": "2022-12-18T13:43:12.981490",
          "end_time": "2022-12-18T13:43:15.764366"
        }
      ]
    }
  ]
}
```

GET /executions/{execution_id} (Returns the job execution by execution_id in the detailed format)

The execution response may initially return an approximate number of rows at the start of execution and provide actual values later during the execution.

- For Oracle Connector, Hyperscale uses the view ALL_TABLES to obtain the approximate number of rows, and the `COUNT` SQL query on the `ROWID` column of the corresponding table is used to retrieve the actual row count.
- For MSSQL Connector, the sp_spaceused stored procedure (without `updateusage`) is executed to fetch the approximate number of rows and the same store procedure sp_spaceused is executed to retrieve the actual row count but only after execution of command DBCC UPDATEUSAGE on the corresponding table.
- The controller service uses the actual row count to determine the state of a table for each service and the overall job's execution status.

Request:

```
id: 1
```

Response:

```
{
  "id": 1,
  "job_id": 1,
  "status": "SUCCEEDED",
  "create_time": "2023-04-26T12:34:38.012768",
  "end_time": "2023-04-26T12:37:32.410297",
  "total_objects": 1,
  "total_rows": 499999,
  "tasks": [
    {
      "name": "Unload",
      "status": "SUCCEEDED",
      "start_time": "2023-04-26T12:34:38.027224",
      "end_time": "2023-04-26T12:34:42.435849",
      "metadata": [
        {
          "source_key": "dbo.test_TEMP",
          "total_rows": 499999,
          "status": "SUCCEEDED",
          "unloaded_rows": 499999
        }
      ]
    },
    {
      "name": "Masking",
      "status": "SUCCEEDED",
      "start_time": "2023-04-26T12:34:40.420073",
      "end_time": "2023-04-26T12:35:12.423744",
      "metadata": [
        {
          "source_key": "dbo.test_TEMP",
          "total_rows": 499999,
          "status": "SUCCEEDED",
          "masked_rows": 499999
        }
      ]
    },
    {
      "name": "Load",
      "status": "SUCCEEDED",
      "start_time": "2023-04-26T12:37:08.482240",
      "end_time": "2023-04-26T12:37:22.417561",
      "metadata": [
        {
          "source_key": "dbo.test_TEMP",
          "total_rows": 499999,
          "status": "SUCCEEDED",
          "loaded_rows": 499999
        }
      ]
    },
    {
      "name": "Post Load",
      "status": "SUCCEEDED",
      "start_time": "2023-04-26T12:37:22.426813",
      "end_time": "2023-04-26T12:37:22.814583",
      "metadata": [
        {
          "status": "SUCCEEDED",
          "table_set": [
            "test_TEMP_Result"
          ],
          "object_details": [
            {
              "type": "Triggers",
              "total": 2,
              "processed": 2,
              "status": "SUCCEEDED",
              "start_time": "2023-04-26T12:35:10.325948",
              "end_time": "2023-04-26T12:37:22.804792"
            },
            {
              "type": "Indexes",
              "total": 4,
              "processed": 4,
              "status": "SUCCEEDED",
              "start_time": "2023-04-26T12:35:10.325948",
              "end_time": "2023-04-26T12:37:22.804792"
            },
            {
              "type": "Constraints",
              "total": 5,
              "processed": 5,
              "status": "SUCCEEDED",
              "start_time": "2023-04-26T12:35:10.325948",
              "end_time": "2023-04-26T12:37:22.804792"
            }
          ]
        }
      ]
    }
  ]
}
```

- Only in case of execution failure, the below API can be used to restart the execution: `PUT /executions/{execution_id}/restart` (Restart a failed execution).
- The below API can be used only for manually cleaning up the execution: `DELETE /executions/{execution_id}` (Clean up the execution).

How to sync a Hyperscale job

How to import a job from Continuous Compliance Engine

The `POST /import` endpoint is useful when you have a database masking job setup on a Continuous Compliance Engine and need to use the same masking inventory in a Hyperscale job. You can export the masking job details from a Continuous Compliance Engine and import them into the Hyperscale Compliance Orchestrator using the below steps.

1. Export the masking job from the Delphix Continuous Compliance Engine that needs to be imported on the Hyperscale Engine for the dataset preparation. For more information about exporting a job, refer to Export the job.
2. After the job is exported, you can make a request on the Hyperscale Engine with the new `/import` API endpoint to upload the response blob along with `data_info_settings` (Optional) for the source and target dataset.

The following is an example of the `request blob`.

```
{
  "exportResponseMetadata": {
    "exportHost": "1.1.1.1",
    "exportDate": "Tue Sep 13 12:55:31 UTC 2022",
    "requestedObjectList": [
      {
        "objectIdentifier": {
          "id": 3
        },
        "objectType": "MASKING_JOB",
        "revisionHash": "2873bd283bd"
      }
    ],
    "exportedObjectList": [
      {
        "objectIdentifier": {
          "id": 2
        },
        "objectType": "SOURCE_DATABASE_CONNECTOR",
        "revisionHash": "8723bd8273b"
      },
      {
        "objectIdentifier": {
          "id": 4
        },
        "objectType": "DATABASE_CONNECTOR",
        "revisionHash": "273db2738vd"
      },
      {
        "objectIdentifier": {
          "id": 4
        },
        "objectType": "DATABASE_RULESET",
        "revisionHash": "f8c0997c804c"
      }
    ]
  },
  "blob": "983nd0239nd923ndf023nfd2p3nd923dn239dn293fn293fnb2",
  "signature": "923nd023nd02",
  "publicKey": "f203fn23fn203[fn230[f",
  "data_info_settings": [
    {
      "prop_key": "unload_split",
      "prop_value": "2",
      "apply_to": "SOURCE"
    },
    {
      "prop_key": "stream_size",
      "prop_value": "65536",
      "apply_to": "TARGET"
    }
  ]
}
```

3. The Hyperscale Engine will then process the required data object from the sync bundle and prepare the connector and data objects that are required for the hyperscale job creation.

After successful import,

1. You must provide the password for connectors manually. To do so, perform the following steps:
    a. Get the newly created data-set using `GET /data-sets/{dataSetId}` to get the newly created connector-info id.
    b. Copy the connector-id and call the `GET /connector-info/{connectorInfoId}` and copy the response.
    c. Use the `PUT /connector-info/{connectorInfoId}` and in the body, paste the GET response and add the new password field with password value in the source and target to update the connector password.
    d. If the bundle is passphrase protected, then the same needs to be provided while importing the bundle in the API header as “passphrase”. For more information about how to export a passphrase encrypt bundle, refer to the Export the Object section.

4. The Hyperscale Engine will provide the data object identifier that can be further used as it is (after updating the passwords of the associated connector) to create a hyperscale job or if needed, can also be updated before configuring a job. The following is an example of the `response`.

```
{
    "data_set_id": id
}
```

ℹ After successful import, you must provide the password for connectors manually. To do so, perform the following steps:

1. Get newly created data-set using `GET /data-sets/{dataSetId}` to get the newly created connector-info id.
2. Copy the `connector-id` and call the `GET /connector-info/{connectorInfoId}` and copy the response.
3. Use the `PUT /connector-info/{connectorInfoId}` and in the body, paste the `GET` response and add the new password field with password value in the source and target to update the connector password.
4. If the bundle is passphrase protected, then the same needs to be provided while importing the bundle in the API header as "passphrase". For more information about how to export passphrase encrypt bundle, refer to the Export the object section.

How to re-import a Job from Continuous Compliance Engine

To update the existing dataset on the Hyperscale Compliance Orchestrator with a refreshed ruleset from the Continuous Compliance Engine, use `PUT /import/{datasetId}` endpoint.

Export the masking job from the Delphix Continuous Compliance Engine having refreshed ruleset and re-import the exported bundle into Hyperscale Compliance Orchestrator by providing the existing dataSetId.

ℹ The `data_info_settings` provided in this update request will only be applicable to objects(in the export bundle) which are not present in the existing dataset on Hyperscale Orchestrator.

Script to automatically import/re-import a Job from Continuous Compliance Engine

Hyperscale provides a utility script to automate the steps of syncing masking jobs inventory from Continuous Compliance Engine into the connector and dataset info of Hyperscale Compliance Orchestrator. This utility script is bundled with the release tar file and can be found at `<deployment_directory>/tools/import-scripts/`.

How to sync global settings from a Delphix Continuous Compliance Engine

The `POST /sync-compliance-engines` endpoint is useful when you have global objects set up on a Continuous Compliance Engine and need to use the same global-objects-like algorithms in a Hyperscale job. You can export the details of the global object from a Continuous Compliance Engine and import them into the Hyperscale Compliance Orchestrator using the below steps.

1. Export the global settings from the Delphix Continuous Compliance Engine that needs to be imported on the Hyperscale Clustered Continuous Compliance Engines. For more information about exporting global settings, refer to Syncing all Global Objects.

2. Once the bundle is exported, you can make a request on the Hyperscale Engine with the new `/sync-compliance-engines` endpoint to upload the response blob along with a list of Hyperscale Clusters Compliance Engines. For more information, refer to the /sync-compliance-engines API page. The following is an example of the `request blob`.

```
{
  "exportResponseMetadata": {
    "exportHost": "1.1.1.1",
    "exportDate": "Tue Sep 13 12:55:31 UTC 2022",
    "requestedObjectList": [
      {
        "objectIdentifier": {
          "id": "global"
        },
        "objectType": "GLOBAL_OBJECT",
        "revisionHash": "897weqwj76"
      }
    ],
    "exportedObjectList": [
      {
        "objectIdentifier": {
          "id": 12
        },
        "objectType": "PROFILE_EXPRESSION",
        "revisionHash": "7dc67asch8a"
      },
      {
        "objectIdentifier": {
          "id": "BIOMETRIC"
        },
        "objectType": "DOMAIN",
        "revisionHash": "7edb8ewbd8w"
      },
      {
        "objectIdentifier": {
          "algorithmName": "dlpx-core:Email SL"
        },
        "objectType": "USER_ALGORITHM",
        "revisionHash": "87h823d23d23"
      }
    ]
  },
  "blob": "39fdn23d9834fn3948f348fbw3pd9234nf9p4hf89",
  "signature": "7823hd823bd8",
  "publicKey": "892d3un293dn2p39db8283",
  "compliance_engine_ids": [
    1,
    2
  ]
}
```

ℹ

1. After import, if Hyperscale Clustered Continues Compliance Engines already have same objects with same id or properties, then those objects will be overwritten.
2. If the bundle is passphrase protected, then the same needs to be provided while importing the bundle in the header as "passphrase". For more information about how to export passphrase encrypt bundle, refer to the Export the object section.

Limitations

Hyperscale Job Sync feature has the following limitations:

1. The default maximum supported size for syncing a document or request is 50 MB. You have the option to customize this by mounting a custom `nginx.conf` under the volumes of the proxy service in the `docker-compose.yaml` file and specify client_max_body_size with the new value. For more information, refer to Custom Configuration.
2. Pre and post-script import from the Continuous Compliance Engine to Hyperscale Engine is not supported.
3. Import of Kerberos and Custom JDBC drivers connector-based making job is not supported.

How to restart a Hyperscale job

ℹ The Hyperscale Job restart feature is only available for Oracle and MS SQL Connectors.

A Hyperscale Job Execution can be restarted if the Load or Post Load process of the Execution has failed or been cancelled.

To restart a Hyperscale Execution, use the: */executions/{id}/restart* PUT endpoint of JobExecution API.

When restarting a Hyperscale Execution, if:

- The Load process of the Execution has FAILED or has been CANCELLED, the Job Execution restart endpoint will trigger a restart of the Load process (this will also include a restart of the Pre-Load process).
- The Load process of the Execution has already SUCCEEDED but the Post Load process has FAILED or has been CANCELLED, the Job Execution restart endpoint will trigger a restart of Post Load process only.

To restart a Hyperscale Job Execution, the following conditions on the Job Execution should be met:

1. The status of the Execution should be either FAILED or CANCELLED.
2. The Unload and Masking process of the Execution should have SUCCEEDED status.
3. The cleanup of the Execution should not have been triggered.
4. The retain execution data flag (*retain_execution_data*) of the Job should not be set to NO.
5. Hyperscale containers should not be restarted after the failure or cancellation of the Job Execution.

How to cancel a Hyperscale job

To cancel a Hyperscale Job while the execution is running, the `/executions/{id}/cancel` endpoint of JobExecution API can be used. Here `{id}` is the Hyperscale Execution Id that needs to be cancelled.

Hyperscale Job cancellation is an async process. The progress of Job cancellation can be tracked by checking the Execution Status using `GET /execution/{id}/summary` API endpoint. As the cancellation process starts for a Hyperscale execution, the Execution Status becomes `CANCEL_INITIATED`. Whenever the cancellation process completes for the Execution, the Execution Status gets updated to `CANCELLED`.

During cancellation, the Hyperscale application:

1. Stops the processes running on the Hyperscale Orchestrator with respect to that Execution.
2. Closes the database connections made with the target database.
3. Cancels the Masking Jobs running on the Continuous Compliance Engines and waits for their cancellation to complete before marking Hyperscale Execution as CANCELLED.

- Hyperscale doesn’t stop any running processes on the target database. Hyperscale Job cancellation might leave the target database in an inconsistent state.
- This operation (cancel job) is not applicable for the Delimited and Parquet connectors.

How to generate a support bundle

1. POST /support-bundle API

The POST `/support-bundle` endpoint is to trigger the **asynchronous** process to generate the support bundle. The following is an example of the response:

```
{
  "async_task_id": 1,
  "operation": "SUPPORT_BUNDLE_GENERATE",
  "reference": "hyperscale-support-20240319-13-38-33",
  "status": "RUNNING",
  "start_time": "2024-03-19T13:38:34.325159Z",
  "cancellable": true
}
```

2. GET /async-tasks/{async-task-id} API

Since the generation of support bundle is an asynchronous process, it is helpful to know the current status of these activities by utilizing the GET `/async-tasks/{async-task-id}` endpoint. Here the value of request parameter `async-task-id` will be the value of `async_task_id` field of the response of `/support-bundle` API (defined above)/

In the response of the API, the current status of the `support-bundle` generation can be identified by the field `status`.

The following is an example of the response:

```
{
  "async_task_id": 1,
  "operation": "SUPPORT_BUNDLE_GENERATE",
  "reference": "hyperscale-support-20240319-13-38-33.tar.gz",
  "status": "SUCCEEDED",
  "start_time": "2024-03-19T13:38:34.325159Z",
  "end_time": "2024-03-19T13:39:43.118582Z",
  "cancellable": true
}
```

3. GET /file-download API

GET `/file-download` API can be triggered to download the generated support-bundle when `status` from the response of GET `/async-tasks/{async-task-id}` is received as `SUCCEEDED` or `PARTIAL_SUCCEEDED`.

This API expects query parameter `entity_type` as `SUPPORT_BUNDLE` and `id` as the value of `async_task_id` from the above response.

The response of this API will be in `octet-stream` and a download link will be provided of the support-bundle tar file.

4. Other related APIs

a) DELETE /support-bundle/{async-task-id} API

The DELETE `/support-bundle/{async-task-id}` API is useful to delete any already `SUCCEEDED` or `PARTIAL_SUCCEEDED` or `FAILED` support-bundle file or directory.
This is useful to clean up the disk space consumed by these tar files or directories.

b) PUT /async-tasks/{async-task-id}/cancel API

The PUT `async-tasks/{async-task-id}/cancel` API is useful to cancel any `RUNNING` support-bundle asynchronous generation.
The following is an example of the response:

```
{
  "async_task_id": 1,
  "operation": "SUPPORT_BUNDLE_GENERATE",
  "reference": "hyperscale-support-20240319-13-38-33",
  "status": "CANCELLED",
  "start_time": "2024-03-19T13:38:34.325159Z",
  "end_time": "2024-03-19T13:38:43.118582Z",
  "cancellable": true
}
```

c) GET /async-tasks

This API is useful to list all async tasks. The following is an example of the response:

```
[
  {
    "async_task_id": 1,
    "operation": "SUPPORT_BUNDLE_GENERATE",
    "reference": "hyperscale-support-20240318-09-54-53.tar.gz",
    "status": "SUCCEEDED",
    "start_time": "2024-03-18T09:54:53.419806Z",
    "end_time": "2024-03-18T09:55:53.957697Z",
    "cancellable": true
  },
  {
    "async_task_id": 2,
    "operation": "SUPPORT_BUNDLE_GENERATE",
    "reference": "hyperscale-support-20240318-14-45-03.tar.gz",
    "status": "PARTIAL_SUCCEEDED",
    "start_time": "2024-03-18T14:45:04.00008Z",
    "end_time": "2024-03-18T14:46:04.294322Z",
    "cancellable": true,
    "errors": [
      {
        "message": "Support bundle generation API is not enabled",
        "object_name": "load-service"
      },
      {
        "message": "Support bundle generation API is not enabled",
        "object_name": "unload-service"
      }
    ]
  },
  {
    "async_task_id": 3,
    "operation": "SUPPORT_BUNDLE_GENERATE",
    "reference": "hyperscale-support-20240319-13-38-33",
    "status": "CANCELLED",
    "start_time": "2024-03-19T13:38:34.325159Z",
    "end_time": "2024-03-19T13:39:34.356925Z",
    "cancellable": true
  }
]
```

Configuration settings

1. Possible values of Configuration Settings having Type “Log Level” are TRACE, DEBUG, INFO, WARN, ERROR, FATAL, or OFF.
2. Commonly used properties can be configured in the `.env` file. The other properties must be configured in the `docker-compose.yaml` under the respective service environment.
3. If you define property values in `.env` and `docker-compose` file both, then values from `docker-compose` will take precedence.

The following table lists the Hyperscale Compliance properties with their default values.

Commonly used properties

Group	Property name	Type	Description	Default value
Controller Service	`API_KEY_CREATE`	Boolean	This property is by default uncommented to have the container create a new API key and print it in the logs when starting. Since the value is in the logs, this API key should only be used to bootstrap the creation of other - more secure - API keys and be discarded. Comment it once the bootstrap key is available.	true
	`LOG_LEVEL_CONTROLLER_SERVICE`	Log Level	Hyperscale logging level. This configuration controls the logging level of Hyperscale specific packages. This log level can be increased if Hyperscale service specific actions needs to be monitored closely. **NOTE:** It is recommended to keep this log level to INFO. Increasing the log level can impact application’s performance.	INFO

Group	Property name	Type	Description	Default value
	`API_VERSION_COMPATIBILITY_STRICT_CHECK`	Boolean	These properties are used to check the version compatibility. Setting this as true will enable strict comparison of API versions of different services. In strict comparison, the complete version i.e x.y.z is compared while in other case when this property is set to false, only major version(x out of x.y.z) of APIs will be compared.	false
	`EXECUTION_STATUS_POLL_DURATION`	Milli-seconds	Time duration in which execution status is collected from different services	120000
	`LOAD_SERVICE_REQUIRE_POST_LOAD`	Boolean	Set if the Post Load step needs to be executed.	true
	`SKIP_UNLOAD_SPLIT_COUNT_VALIDATION`	Boolean	Skip ‘split count’ and ‘number of unload files generated’ validation, while determining execution status **Note:** For connector-specific values, see Data Source Support.	false
	`SKIP_LOAD_SPLIT_COUNT_VALIDATION`	Boolean	Skip ‘split count’ and ‘number of masked files loaded by loaded’ validation, while determining execution status **Note:** For connector-specific values, see Data Source Support.	false
	`CANCEL_STATUS_POLL_DURATION`	Milli-seconds	Time duration in which execution status is collected from different services for the CANCEL_INITIATED executions.	60000
	`VALIDATE_UNLOAD_ROW_COUNT_FOR_STATUS`	Boolean	Flag to control if row counts should be considered as a factor to decide the completion status of the unload step of an object.	true

Group	Property name	Type	Description	Default value
	`VALIDATE_MASKED_ROW_COUNT_FOR_STATUS`	Boolean	Flag to control if row counts should be considered as a factor to decide the completion status of the masking step of an object.	true
	`VALIDATE_LOAD_ROW_COUNT_FOR_STATUS`	Boolean	Flag to control if row counts should be considered as a factor to decide the completion status of the load step of an object.	true
	`DISPLAY_BYTES_INFO_IN_STATUS`	Boolean	Flag to control the display of number of bytes in Execution Status response	false
	`DISPLAY_ROW_COUNT_IN_STATUS`	Boolean	Flag to control the display of number of rows in Execution Status response	true
	`NFS_STORAGE_HOST`	String	Host address of the NFS server.	
	`NFS_STORAGE_EXPORT_PATH`	String	Path to the directory on the filesystem to mount.	
	`NFS_STORAGE_MOUNT_TYPE`	String	The type of filesystem. Enum having values- CIFS, NFS3, NFS4.	
	`NFS_STORAGE_MOUNT_OPTION`	String	The options for mount, e.g. “rw”	rw
	`APPLICATION_NAME`	String	Name of the Hyperscale application. For existing customers, this shall be analogous to the mount name used for staging area.	hs-staging-area

Group	Property name	Type	Description	Default value
	`STAGING_STORAGE_TYPE`	`String`	Type of staging area. Supported values - MOUNT, AWS_S3	MOUNT
	`authMechanism`	`String`	Authentication mechanism to access AWS S3 bucket as staging area. Supported values - AWS_SECRET, AWS_ROLE Required if `STAGING_STORAGE_TYPE` is AWS_S3.	
	`awsBucketName`	`String`	Name of AWS S3 bucket used as staging area. Required if `STAGING_STORAGE_TYPE` is AWS_S3 Required if `STAGING_STORAGE_TYPE` is AWS_S3	
	`awsBucketRegion`	`String`	Region of AWS S3 bucket used as staging area (e.g. us-west-2). Required if `STAGING_STORAGE_TYPE` is AWS_S3	
	`awsBucketPrefix`	`String`	Prefix of AWS S3 bucket used as staging area. Optional. e.g. hyperscale/oracle/	
	`awsBucketDelimiter`	`String`	Delimiter for the prefix on AWS S3 bucket used as staging area.	`/`
	`stagingAwsS3SecretName`	`String`	Name of the Kubernetes secret that holds `awsAccessKey` and `awsSecretKey` information	`s3-staging-secre`t

Group	Property name	Type	Description	Default value
	`awsAccessKey`	`String`	Access key ID of the AWS S3 bucket used as staging area. Not required if `stagingAwsS3SecretName` is provided.	
	`awsSecretKey`	`String`	Secret Access key of the AWS S3 bucket used as staging area. Not required if `stagingAwsS3SecretName` is provided.	
Unload Service	`LOG_LEVEL_UNLOAD_SERVICE`	Log Level	Hyperscale logging level. This configuration controls the logging level of Hyperscale specific packages. This log level can be increased if Hyperscale service specific actions needs to be monitored closely. NOTE: It is recommended to keep this log level to INFO. Increasing the log level can impact application's performance.	INFO
	`UNLOAD_FETCH_ROWS`	Number	Number of rows to be fetched from the database at a time.	10000
	`CONCURRENT_EXPORT_LIMIT`	Number	Number of concurrent mongoexport processes to be spawned.	10
	`AUTO_CALCULATE_UNLOAD_SPLIT`	Boolean	Only for Oracle. if true, the unload service will unload data based 'Automatic Calculation of Unload Split' approach by using max_records_per_split , max_parallel_connections_per_table, max_split_per_connection	false
	`MAX_RECORDS_PER_SPLIT`	long	Only for Oracle. Maximum records unloaded in an extracted split file	10000000
	`MAX_SPLIT_PER_CONNECTION`	Integer	Only for Oracle. This param indicates how many splits each database connection can take.	4

Group	Property name	Type	Description	Default value
	`MAX_PARALLEL_CONNECTIONS_PER_TABLE`	Integer	Only for Oracle. This param indicates how many maximum parallel connections are allowed for a table during the unload process	10
Masking Service	`LOG_LEVEL_MASKING_SERVICE`	Log Level	Hyperscale logging level. This configuration controls the logging level of Hyperscale specific packages. This log level can be increased if Hyperscale service specific actions needs to be monitored closely. NOTE: It is recommended to keep this log level to INFO. Increasing the log level can impact application's performance.	INFO
	`INTELLIGENT_LOADBALANCE_ENABLED`	Boolean	Set this to false if need to enable round robin load balancing in place of intelligent load balancing.	true
Load Service	`LOG_LEVEL_LOAD_SERVICE`	Log Level	Hyperscale logging level. This configuration controls the logging level of Hyperscale specific packages. This log level can be increased if Hyperscale service specific actions need to be monitored closely. NOTE: It is recommended to keep this log level to INFO. Increasing the log level can impact application's performance.	INFO
	`SQLLDR_BLOB_CLOB_CHAR_LENGTH`	Number	This property configures maximum length of CLOB/BLOB type column in Oracle Client (SQLLDR).	20000
	`LOAD_FEATURE_FLAG_REPLACE_NULL_LOB`	Boolean	Set if the automated load handling of CLOB/BLOB value (when null or empty) is needed.	true

Group	Property name	Type	Description	Default value
Load Service (MS SQL)	`TARGET_DB_MAX_CONNECTION_POOL_SIZE`	Integer	This commands the size of target database connection pool for MS SQL Post Load process. This is only applicable for the post load process.	15

Other properties

Group	Property name	Type	Description	Default value
Controller Service	`SOURCE_KEY_FIELD_NAMES`	String	Dataset configuration. These fields/ columns are used to uniquely identify source data.	schema_name, table_name
	`LOGGING_LEVEL_ROOT`	Log Level	Logging configuration. This spring boot configuration controls the logging of all the packages/libraries getting used in application. NOTE: Increasing this Log Level will produce too many logs. It is recommended to keep this log level to WARN or below.	WARN
	`LOGGING_FILE_NAME` [1]	String	Log file location & name	/opt/delphix/logs/hyperscale-controller.log

Group	Property name	Type	Description	Default value
	`LOGGING_PATTERN_FILE`	String	Logging pattern for file	`%d{dd-MM-yyyy HH:mm:ss.SSS} \[%thread\] %-5level %logger{36}.%M - %msg%n`
	`LOGGING_PATTERN_CONSOLE`	String	Logging pattern for console	`%d{dd-MM-yyyy HH:mm:ss.SSS} \[%thread\] %-5level %logger{36}.%M - %msg%n`

Group	Property name	Type	Description	Default value
	`LOGGING_PATTERN_ROLLINGFILENAME`[1]	String	Archived file location & name	`/opt/delphix/logs/archived/hyperscale-controller-%d{yyyy-MM-dd}.%i.log`
	`LOGGING_FILE_MAXSIZE`	File Size (String)	Max individual file size	5MB
	`LOGGING_FILE_MAXHISTORY`	Number of Days	History in days (i.e. keep 15 days' worth of history capped at 5GB total size)	15
	`LOGGING_FILE_TOTALCAPSIZE`	File Size (String)	Max limit the combined size of log archives	5GB
	`LOGGING_LEVEL_ORG_SPRINGFRAMEWORK_WEB_FILTER_COMMONSREQUESTLOGGINGFILTER`	Log Level	This configuration controls the logging information of the HTTP requests received by Hyperscale. This is by default set to DEBUG level for logging request URIs of the Incoming Requests.	DEBUG
	`API_VERSION_COMPATIBILITY_RETRY_COUNT`	Number	These properties are used to check the version compatibility. Number of times to retry the comparison if the services are not compatible.	3

Group	Property name	Type	Description	Default value
	`API_VERSION_COMPATIBILITY_RETRY_WAIT_TIME`	Time in milli-seconds	These properties are used to check the version compatibility. Time to wait before next retry if the services are not compatible.	10000
	`SUPPORT_BUNDLE_AUTO_CLEANUP_INTERVAL`	Number	Poll duration in hours to check and clean the generated support bundle.	24
	`SUPPORT_BUNDLE_AUTO_CLEANUP_BUFFER`	Number	To determine when support bundle should be cleaned, use the buffer duration in days. For example, if the buffer duration is 5, then support bundles which ends later than the current time plus five days will be cleaned.	5
	`MOUNT_ACCESS_RETRY_COUNT`	Integer	The `MOUNT_ACCESS_RETRY_COUNT` property allows the Hyperscale engine ten attempts to access the files or directory on the mount server.	10
	`MOUNT_ACCESS_TIMEOUT`	Number	The `MOUNT_ACCESS_TIMEOUT` property allows the Hyperscale engine 20 seconds to access a file or directory.	20
Unload Service	`LOGGING_LEVEL_ROOT`	Log Level	Logging configuration. This spring boot configuration controls the logging of all the packages/libraries getting used in application. NOTE: Increasing this Log Level will produce too many logs. It is recommended to keep this log level to WARN or below.	WARN
	`LOGGING_FILE_NAME` [1]	String	Log file location & name	/opt/delphix/logs/hyperscale-unload.log

Group	Property name	Type	Description	Default value
	`LOGGING_PATTERN_FILE`	String	Logging pattern for file	`%d{dd-MM-yyyy HH:mm:ss.SSS} \[%thread\] %-5level %logger{36}.%M - %msg%n`
	`LOGGING_PATTERN_CONSOLE`	String	Logging pattern for console	`%d{dd-MM-yyyy HH:mm:ss.SSS} \[%thread\] %-5level %logger{36}.%M - %msg%n`
	`LOGGING_PATTERN_ROLLINGFILENAME`[1]	String	Archived file location & name	`/opt/delphix/logs/archived/hyperscale-unload-%d{yyyy-MM-dd}.%i.log`

Group	Property name	Type	Description	Default value
	`LOGGING_FILE_MAXSIZE`	File Size in String	Max individual file size	5MB
	`LOGGING_FILE_MAXHISTORY`	Number of Days	History in days (i.e. keep 15 days' worth of history capped at 5GB total size)	15
	`LOGGING_FILE_TOTALSIZECAP`	File Size in String	Max limit the combined size of log archives	5GB
	`LOGGING_LEVEL_ORG_SPRINGFRAMEWORK_WEB_FILTER\_COMMONSREQUESTLOGGINGFILTER`	Log Level	This configuration controls the logging information of the HTTP requests received by Hyperscale. This is by default set to DEBUG level for logging request URIs of the Incoming Requests.	DEBUG
	`UNLOAD_CONF_DISABLE_REPLICATION`	Boolean	If set to true, disable database replication for source database. This property is only applicable for MS SQL unload image	true
	`UNLOAD_SPARK_DRIVER_MEMORY`	Integer	Spark driver memory to be consumed for unload. This property is only applicable for MS SQL unload image	90% of available memory
	`UNLOAD_SPARK_DRIVER_CORES`	Integer	Spark cores to be consumed for unload. This property is only applicable for MS SQL unload image	90% of available memory
	`UNLOAD_HIKARI_MAX_LIFE_TIME`	Integer	Value in Milliseconds, Please follow maxLifetime from Hikari Documentation.	1800000 (30 min)

Group	Property name	Type	Description	Default value
	`UNLOAD_HIKARI_KEEP_ALIVE_TIME`	Integer	Value in Milliseconds, Please follow keepaliveTime from Hikari Documentation.	300000 (5 min)
	`FILE_DELIMITER`	Char	Column value delimiter character.This configuration for internal use to have data from DB tables into files. But can be updated in specific scenario Note: Use unicode char sequence for Quotation Marks characters (i.e. \u0022 unicode of double-quote) & Non-ASCII characters are **NOT** allowed	, (Single-Quote)
	`FILE_ENCLOSURE`	Char	This configuration for internal use to have data from DB tables into files. But can be updated in specific scenario (i.e. XML data masking) Note: Use unicode char sequence for Quotation Marks characters (i.e. \u0022 unicode of double-quote) & Non-ASCII characters are **NOT** allowed	“ (Double-Quote)
	`FILE_ESCAPE_ENCLOSURE`	Char	This configuration for internal use to have data from DB tables into files. But can be updated in specific scenario (i.e. XML data masking) Note: Use unicode char sequence for Quotation Marks characters (i.e. \u0022 unicode of double-quote) & Non-ASCII characters are **NOT** allowed	“ (Double-Quote)
	`MOUNT_ACCESS_RETRY_COUNT`	Integer	The `MOUNT_ACCESS_RETRY_COUNT` property allows the Hyperscale engine ten attempts to access the files or directory on the mount server.	10
	`MOUNT_ACCESS_TIMEOUT`	Number	The `MOUNT_ACCESS_TIMEOUT` property allows the Hyperscale engine 20 seconds to access a file or directory.	20

Group	Property name	Type	Description	Default value
Masking Service	`LOGGING_LEVEL_ROOT`	Log Level	Logging configuration. This spring boot configuration controls the logging of all the packages/libraries getting used in application. NOTE: Increasing this Log Level will produce too many logs. It is recommended to keep this log level to WARN or below.	WARN
	`LOGGING_FILE_NAME` [1]	String	Log file location & name	/opt/delphix/logs/hyperscale.log
	`LOGGING_PATTERN_FILE`	String	Logging pattern for file	`%d{dd-MM-yyyy HH:mm:ss.SSS} \[%thread\] %-5level %logger{36}.%M - %msg%n`
	`LOGGING_PATTERN_CONSOLE`	String	Logging pattern for console	`%d{dd-MM-yyyy HH:mm:ss.SSS} \[%thread\] %-5level %logger{36}.%M - %msg%n`

Group	Property name	Type	Description	Default value
	`LOGGING_PATTERN_ROLLINGFILENAME` [1]	String	Archived file location & name	`/opt/delphix/logs/archived/hyperscale-%d{yyyy-MM-dd}.%i.log`
	`LOGGING_FILE_MAXSIZE`	File Size in String	Max individual file size	5MB
	`LOGGING_FILE_MAXHISTORY`	Number of Days	History in days (i.e. keep 15 days' worth of history capped at 5GB total size)	15
	`LOGGING_FILE_TOTALSIZECAP`	File Size in String	Max limit the combined size of log archives	5GB
	`LOGGING_LEVEL_ORG_SPRINGFRAMEWORK_WEB_FILTER_COMMONSREQUESTLOGGINGFILTER`	Log Level	This configuration controls the logging information of the HTTP requests received by Hyperscale. This is by default set to DEBUG level for logging request URIs of the Incoming Requests.	DEBUG

Group	Property name	Type	Description	Default value
	`MASKING_ILB_QUEUEINGFACTOR`	Float	This property, Queueing Factor(QF) can be used to increase the number of jobs being assigned to a Continuous Compliance Engine by the factor mentioned. (Net jobCapacity of CCE = CCE’s total jobCapacity * Queueing Factor). It can be used to increase the utilization of Continuous Compliance Engines. Increasing this property value can lead to jobs getting queued on Masking Engines. **NOTE:** This property is only applicable if INTELLIGENT_LOADBALANCE_ENABLED is set to true(default value).	1.0
	`MONITOR_POLL_DURATION`	Time in seconds	Used to set the frequency with which Masking Engines will be polled to fetch job Status	10s
	`ENGINE_FAILURE_POLL_DURATION`	Time in seconds	Incase of Connection/Handshake issues or unavailability of Masking Engines, the subjobs on the engine will be marked as failed after retrying for this duration.	60s
	`MOUNT_ACCESS_RETRY_COUNT`	Integer	The `MOUNT_ACCESS_RETRY_COUNT` property allows the Hyperscale engine ten attempts to access the files or directory on the mount server.	10
	`MOUNT_ACCESS_TIMEOUT`	Number	The `MOUNT_ACCESS_TIMEOUT` property allows the Hyperscale engine 20 seconds to access a file or directory.	20
Load Service	`SQLLDR_SUCCESS_MESSAGE`	String	Message printed by sqlldr on successful loading of data.	‘successfully loaded.’

Group	Property name	Type	Description	Default value
	`LOGGING_LEVEL_ROOT`	Log Level	Logging configuration. This spring boot configuration controls the logging of all the packages/libraries getting used in application. NOTE: Increasing this Log Level will produce too many logs. It is recommended to keep this log level to WARN or below.	WARN
	`LOGGING_LEVEL_COM_DELPHIX_MASKING`	Log Level	Log level for driver support. This configuration controls the logging level of the Masking Driver Support package. This Log Level can be increased when Driver Support Steps of Load Process need to be monitored closely.	INFO
	`LOGGING_FILE_NAME` [1]	String	Log file location & name	/opt/delphix/logs/hyperscale-load.log
	`LOGGING_PATTERN_FILE`	String	Logging pattern for file	`%d{dd-MM-yyyy HH:mm:ss.SSS} \[%thread\] %-5level %logger{36}.%M - %msg%n`

Group	Property name	Type	Description	Default value
	`LOGGING_PATTERN_CONSOLE`	String	Logging pattern for console	`%d{dd-MM-yyyy HH:mm:ss.SSS} \[%thread\] %-5level %logger{36}.%M - %msg%n`
	`LOGGING_PATTERN_ROLLINGFILENAME` [1]	String	Archived file location & name	`/opt/delphix/logs/archived/hyperscale-load-%d{yyyy-MM-dd}.%i.log`
	`LOGGING_FILE_MAXSIZE`	File Size in String	Max individual file size	5MB
	`LOGGING_FILE_MAXHISTORY`	Number of Days	History in days (i.e. keep 15 days' worth of history capped at 5GB total size)	15
	`LOGGING_FILE_TOTALSIZECAP`	File Size in String	Max limit the combined size of log archives	5GB

Group	Property name	Type	Description	Default value
	`LOGGING_LEVEL_ORG_SPRINGFRAMEWORK_WEB_FILTER_COMMONSREQUESTLOGGINGFILTER`	Log Level	This configuration controls the logging information of the HTTP requests received by Hyperscale. This is by default set to DEBUG level for logging request URIs of the Incoming Requests.	DEBUG
	`LOAD_SPARK_BATCH_SIZE`	Integer	Spark batch size for bulk load. This property is only applicable for MS SQL load image	10000
	`LOAD_SPARK_JOB_TABLE_LOCK`	Boolean	Spark job table lock for bulk load. This property is only applicable for MS SQL load image	true
	`LOAD_CONF_DISABLE_REPLICATION`	Boolean	If set to true, disable database replication for target database. This property is only applicable for MS SQL load image	true
	`LOAD_SPARK_DRIVER_MEMORY`	Integer	Spark driver memory to be consumed for load. This property is only applicable for MS SQL unload image.	90% of available memory
	`LOAD_SPARK_DRIVER_CORES`	Integer	Spark cores to be consumed for load. This property is only applicable for MS SQL unload image.	90% of available processors
	`LOAD_HIKARI_MAX_LIFE_TIME`	Integer	Value in Milliseconds, Please follow maxLifetime from Hikari Documentation.	1800000 (30 min)
	`LOAD_HIKARI_KEEP_ALIVE_TIME`	Integer	Value in Milliseconds, Please follow keepaliveTime from Hikari Documentation.	300000 (5 min)

- For each service, the file path(absolute) configured for `logging.file.name` and for `logging.pattern.rolling-file-name` has to be the same. This path is a path inside the respective container.
- For each service, if the log files(configured through `logging.file.name` and `logging.pattern.rolling-file-name`) need to be accessed outside the container, the respective log path has to be mounted by adding volume binding of that path in `docker-compose.yaml` for that service.

Oracle Connector- From the 23.0.0 release onwards, you no longer need to configure any unload/load date format environment variables to mask date/timestamp columns as Hyperscale Compliance introduces a feature to handle multiple date formats automatically in a single Hyperscale job. Therefore, the following environment properties configured in the `docker-compose.yaml` / `values.yaml` file will no longer be used in the product. You can safely remove these properties:

1. `JDBC_DATE_TIMESTAMP_FORMAT` -- For Unload
2. `SQLLDR_DATE_TIMESTAMP_FORMAT` -- For Load

MSSQL Connector- From the 24.0.0 release onwards, a new Hyperscale feature can automatically handle multiple date formats in a single Hyperscale job. Therefore, you no longer need to configure any unload/ load date format environment variables to mask date/timestamp columns.

Following the 24.0.0. release the below environment properties configured in the `docker-compose.yaml/values.yaml` file will not be used and you can safely remove these properties from unload and load services:

1. `SPARK_TIMESTAMP_FORMAT`
2. `SPARK_DATE_FORMAT`
3. `SPARK_SMALL_DATE_TIMESTAMP_FORMAT`

Hyperscale Compliance API

The Hyperscale Compliance API is organized around REST. Our API has predictable resource-oriented URLs, accepts form-encoded request bodies, returns JSON-encoded responses, and uses standard HTTP response codes, authentication, and verbs.

REST

Hyperscale Compliance API is a RESTful API. REST stands for REpresentational State Transfer. A REST API will allow you to access and manipulate a textual representation of objects and resources using a predefined set of operations to accomplish various tasks.

JSON

Hyperscale Compliance API uses JSON (JavaScript Object Notation) to ingest and return representations of the various objects used throughout various operations. JSON is a standard format and, as such, has many tools available to help with creating and parsing the request and response payloads, respectively. Here are some UNIX tools that can be used to parse JSON - Parsing JSON with Unix Tools. That being said, this is only the tip of the iceberg when it comes to JSON parsing and the reader is encouraged to use their method of choice.

API Client
The various operations and objects used to interact with APIs are defined in a specification document. This allows us to utilize various tooling to ingest that specification to generate documentation and an API Client, which can be used to generate cURL commands for all operations.

Accessing the Hyperscale Compliance API

For accessing the Hyperscale Compliance API, see Accessing the Hyperscale Compliance API.

View the API reference

Hyperscale 2025.1.0 API documentation

Cleaning up execution data

As part of the Hyperscale execution run some data files and objects are created which should be cleaned on the execution completion. These files and objects are:

1. The system will create data files (unload service) and masked files (masking service) on the file server. As the data size can be large (2 times of source data) and include sensitive information, therefore, it is important to clean up this data.
2. The system will create multiple data objects like connectors, rulesets, file formats, jobs, etc on the respective Continuous Compliance Engines. Objects created by Hyperscale should be cleaned once Hyperscale execution is complete.
3. Additionally unload service, masking service, and load service will also store transient internal data for the execution while running it. This data is not required once execution is completed.

Following are the three ways this data will be/can be cleaned.

1. **Using retain_execution_data**
   While setting up a Hyperscale Job (`POST /jobs`), you can set the value for retain_execution_dataproperty to the intimate system when it should clean up data automatically based on the table below.

EXECUTION_STATUS	RETAIN_EXECUTION_DATA	CLEAN UP AUTOMATICALLY?
NA(SUCCESS/FAILED/CANCELED)	NO	YES
SUCCESS	ON_ERROR	YES
FAILED	ON_ERROR	NO
CANCELED	ON_ERROR	NO
NA(SUCCESS/FAILED/CANCELED)	ALWAYS	NO

2. **Manual clean up**
Hyperscale exposes a delete API (`DELETE /executions/{id}`) to manually clean up data for execution if it's not already cleaned.

3. **Start a new execution**
While starting a new execution, Hyperscale will first validate if the previous execution data is cleaned. If it’s not cleaned, then Hyperscale will trigger cleanup before starting new execution.

Limitations

As part of the clean-up process, applications created on the Continous Compliance Engines by Hyperscale job execution are not deleted. You must use the Continuous Compliance Engine APIs to delete the application.

Hyperscale profilers

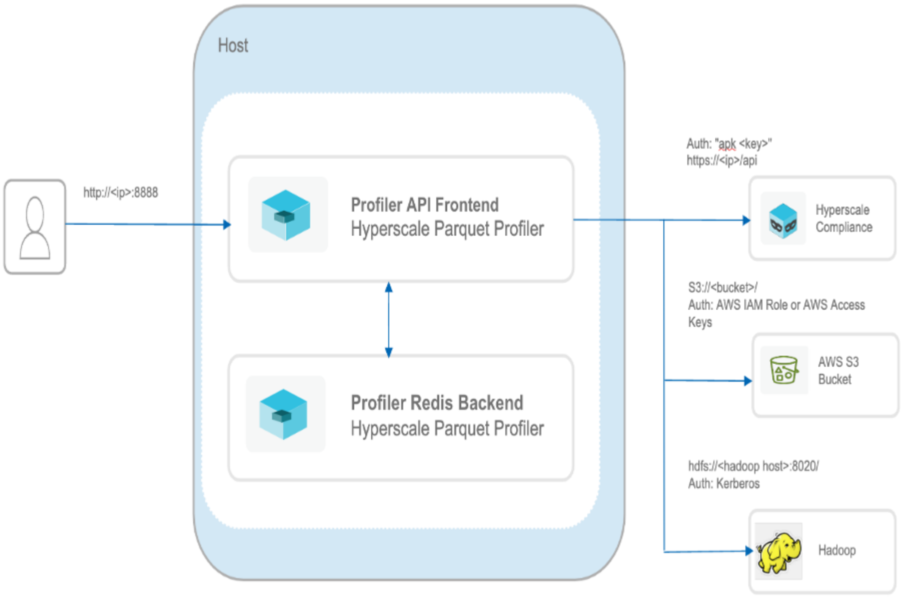
This section covers the following topics:

- Parquet profiler
- Mongo profiler
- Snowflake profiler

Parquet profiler

Identifying the schema of parquet files and aligning columns with the suitable masking algorithms from the Delphix Compliance Engine poses a challenge. Moreover, when dealing with a large number of files (similar/varied) in the source location, establishing a Hyperscale Parquet Connector dataset becomes challenging. The Parquet Profiler serves as a solution to address this complexity.

The main objective of the Parquet Profiler is to analyze the source parquet files and create a Hyperscale Parquet Connector dataset, essentially helping in creating a masking inventory for the source data.



1 Hyperscale Parquet Profiler - Deployment Architecture

> The Parquet Profiler is tightly coupled with the Hyperscale Parquet Connector and cannot be used for other connectors.

Installation and setup (Parquet profiler)

Pre-requisites

1. Download the latest version of the Parquet Profiler from the Delphix download page.
2. Ensure that the host running the profiler has `docker-compose` installed (the profiler has been tested only in the `docker-compose` environment).
3. The profiler supports AWS S3 buckets as the source locations. We need to ensure that the profiler has access to the source location (similar to how access was set up for the Hyperscale Parquet Connector). You can use the following authentication mechanisms:
    - Attaching the EC2 host running the profiler with an AWS IAM role which has access to the source S3 buckets.
        - IAM Roles are designed for applications to securely make AWS-API requests from EC2 instances, without the necessity to manage the security credentials that the applications use.
        - Using the AWS console UI or AWS CLI, attach the IAM role to the EC2 instance running the Hyperscale services. To know more, check the AWS Documentation.
    - Generating an AWS Access Key ID & AWS Secret Acess Key pair for an AWS Role which has the privileges to access the source S3 bucket.
        - Access keys are long-term credentials generated for an IAM user or role. These keys can be for programmatic requests to the AWS CLI or AWS API (directly or using the AWS SDK). For more information, refer to the AWS Documentation.
4. The Profiler also supports Hadoop as a source location. There are two ways of doing a profiler with Hadoop as a source:
    a. **HADOOP-FS:** With this approach, the profiler will access the HDFS path (given in connector_info) with Kerberos authentication and will scan all parquet files in the given HDFS path.
    b. **HADOOP-DB:** In this approach, the profiler will access the hive database with Kerberos authentication. You have to provide a list of tables while creating the profiler task. The profiler will scan the given tables with the help of Hadoop hive database details given in connector-info.
5. Set up the Hyperscale File Connector and add the required ConnectorInfo details.

Procedure

1. Untar the profiler downloaded from Delphix's download page. It should contain the docker images for the profiler and the `docker-compose.yaml` file to run the profiler.

    ```
    tar -xf parquet-profler.tar.gz
    ```

2. Load the `delphix-hyperscale-profiler-api` and `delphix-hyperscale-profiler-backend` docker images.

    ```
    docker load --input delphix-hyperscale-parquet-profiler-api.tar
    docker load --input delphix-hyperscale-parquet-profiler-backend.tar
    ```

3. Edit the `docker-compose` YAML file to map the controller end-point for the `delphix-hyperscale-profiler-api` to interact with.

```
services:
  ...
  profiler-api-service:
  ...
    environment:
    ...
      - CONTROLLER_URL=https://<controller-ip>/api
```

4. (Optional) You can provide the AWS Access keys as environment variables as well, it will be considered as the default credentials to access the source S3 location.

```
services:
  ...
  profiler-api-service:
  ...
    environment:
    ...
      - AWS_ACCESS_KEY_ID=<access_key_id>
      - AWS_SECRET_ACCESS_KEY=<secret_access_key>
      - AWS_DEFAULT_REGION=<region>
```

5. To profile the data which is present on Hadoop, the user has to provide `core-site.xml` , `krb5.cong` and `hadoop.keytab` files as volume mount

```
services:
  ...
  profiler-api-service:
  ...
    volumes:
      ...
      - /path/to/keytab_file/hadoop.keytab:/app/hadoop.keytab
      - /path/to/hadoop/core-site.xml:/app/hadoop/etc/hadoop/core-site.xml
      - /path/to/etc/krb5.conf:/etc/krb5.conf
```

6. Start the profiler service.

```
docker-compose up -d
```

7. Access the profiler swagger UI at `http://<host-ip>:8888` .

Executing a profiler task

Perform the following steps to execute a profiler task

1. Add the authorization key generated for the controller service into the profiler UI.
    - Click on the **Authorize** button and then add the key as follows, `"apk <authorization-key>"`
2. Validate all the configured connectors on the Hyperscale controller using the `/connector-info` GET API endpoint.
3. If the connector-info contains the AWS credentials, then the response will have the AWS credentials hidden.
   Example: /connector-info response with AWS credentials:

   ```
   [
   {
    "source": {
     "type": "AWS",
     "properties": {
       "server": "S3",
       "path": "s3_bucket_source/sub_folder",
       "aws_region": "us-east-1",
       "aws_access_key_id": "AKIA********",
       "aws_secret_access_key": "x2IX********",
       "aws_role_arn": "56436882398"
     }
    },
    "target": {
     "type": "AWS",
     "properties": {
       "server": "S3",
       "path": "s3_bucket_target/sub_folder",
       "aws_region": "us-east-1",
       "aws_access_key_id": "AKIA********",
       "aws_secret_access_key": "x2IX********",
       "aws_role_arn": "56436882398"
     }
    }
   }
   ]
   ```

   As the credentials are masked, the profiler will need the credentials independently (in case the IAM role-based authentication is not used or the AWS credentials are not set using the environment variables).
   Use the `/source-credentials/{connectorId}` post API endpoint to add the credentials mapped to the connector ID received from the controller.
   POST /source-credentials/{connectorId} - Request JSON

   ```
   {
       "aws_access_key_id": "AKIAJSJDFJSBSG",
       "aws_secret_access_key": "x2IXHFKDjskdnmldf&kksdfh%jsdf"
   }
   ```

POST /source-credentials/{connectorId} - Response JSON

```
{
    "connector_id": 1,
    "aws_access_key_id": "AKIA********",
    "aws_secret_access_key": "x2IX********"
}
```

Example: /connector-info response with Hadoop database details:

```
{
    "id": 1,
    "connectorName": "Hadoop_Hive_connector_hive_16",
    "source": {
      "type": "HADOOP-DB",
      "properties": {
        "server": "hdpserver.dlpxdc.co",
        "database_type": "hive",
        "database_port": "10000",
        "database_name": "default",
        "principal_name": "hive/hdpserver.dlpxdc.co@DLPXDC.CO",
        "protocol": "hdfs"
      }
    },
    "target": {
      "type": "HADOOP-FS",
      "properties": {
        "server": "hdpserver.dlpxdc.co",
        "path": "/targetfiles",
        "port": "8020",
        "principal_name": "hive/hdpserver.dlpxdc.co@DLPXDC.CO",
        "protocol": "hdfs"
      }
    }
  }
```

Example: /connector-info response with Hadoop Filesystem details

```
{
    "id": 1,
    "connectorName": "Hadoop_Hive_connector_hive_16",
    "source": {
      "type": "HADOOP-FS",
      "properties": {
        "server": "hdpserver.dlpxdc.co",
        "path": "/sourcefiles",
        "port": "8020",
        "principal_name": "hive/hdpserver.dlpxdc.co@DLPXDC.CO",
        "protocol": "hdfs"
      }
    },
    "target": {
      "type": "HADOOP-FS",
      "properties": {
        "server": "hdpserver.dlpxdc.co",
        "path": "/targetfiles",
        "port": "8020",
        "principal_name": "hive/hdpserver.dlpxdc.co@DLPXDC.CO",
        "protocol": "hdfs"
      }
    }
  }
```

4. The profile sets are essentially a list of all masking algorithms mapped to domain manes which the profiler can assign to columns. No default profile set is created when starting the Parquet Profiler for the first time. To create a default profile set, hit the API endpoint `/profile-sets`. There should now be a default profile set with ID 1.
   GET /profile-sets - Response JSON

```
[
  {
    "exclusions": [
      "_id",
      "_id.oid",
      "$oid",
      "_id.$oid",
      "id"
    ],
    "set_id": 1,
    "date_created": "2023-12-14T12:46:34.686136",
    "name": "DEFAULT",
    "description": "default profiler set",
    "entities": [
      {
        "domain_name": "ZIP",
        "algorithm_name": "dlpx-core:CM Alpha-Numeric",
        "type": "pattern",
        "regex": "(\\b\\d{5}(?:\\-\\d{4})?\\b)",
        "meta_context": [
          "zip",
          "code"
        ]
      },
      {
        "domain_name": "CREDIT CARD",
        "algorithm_name": "CreditCard",
        "type": "DL"
      },
      {
        "domain_name": "DOB",
        "algorithm_name": "DateShiftDiscrete",
        "date_format": "yyyy-mm-dd",
        "type": "DL_DT",
        "min_age_years": 18,
        "max_age_years": 100
      },
      {
        "domain_name": "EMAIL",
        "algorithm_name": "dlpx-core:Email SL",
        "type": "DL"
      },
      {
        "domain_name": "IP ADDRESS",
        "algorithm_name": "dlpx-core:CM Alpha-Numeric",
        "type": "DL"
      },
      {
        "domain_name": "ADDRESS",
        "algorithm_name": "AddrLookup",
        "type": "DL"
      },
      {
        "domain_name": "CITY",
        "algorithm_name": "USCitiesLookup",
        "type": "DL"
      },
      {
        "domain_name": "COUNTRY",
        "algorithm_name": "NullValueLookup",
        "type": "DL"
      },
      {
        "domain_name": "FIRST_NAME",
        "algorithm_name": "dlpx-core:FirstName",
        "type": "DL"
      },
      {
        "domain_name": "LAST_NAME",
        "algorithm_name": "dlpx-core:LastName",
        "type": "DL"
      },
      {
        "domain_name": "FULL_NAME",
        "algorithm_name": "dlpx-core:FullName",
        "type": "DL"
      },
      {
        "domain_name": "TELEPHONE_NO",
        "algorithm_name": "dlpx-core:Phone US",
        "type": "DL"
      },
      {
        "domain_name": "WEB",
        "algorithm_name": "WebURLsLookup",
        "type": "DL"
      },
      {
        "domain_name": "DRIVING_LC",
        "algorithm_name": "DrivingLicenseNoLookup",
        "type": "DL"
      },
      {
        "domain_name": "SSN",
        "algorithm_name": "dlpx-core:CM Alpha-Numeric",
        "type": "DL"
      }
    ],
    "date_last_updated": "2023-12-14T12:46:34.686142"
  }
]
```

5. Generally, the default profile set should be enough for most use cases. But if you want to map different masking algorithms available in your Delphix Compliance Engine to different domains, you should create your own profile set using the `/profile-sets` POST API endpoint. To know more about the profile sets available in your Delphix Compliance Engine, visit here.
   POST /profile-sets - Request JSON

```
{
  "set_id": 2,
  "name": "custom_profile_set",
  "description": "Different Algorithm Mapping",
  "exclusions": [
       "_id",
      "_id.oid",
      "$oid",
      "_id.$oid",
      "id"
  ],
  "entities": [
     {
        "domain_name": "FIRST_NAME",
        "algorithm_name": "dlpx-core:FirstName",
        "type": "DL"
      },
      {
        "domain_name": "LAST_NAME",
        "algorithm_name": "dlpx-core:LastName",
        "type": "DL"
      }
  ]
}
```

   Understanding the profile-set payload parameters:
   a. **name:** Name of the profile set.
   b. **exclusions:** List of fields (or column names) to exclude from the discovery.

c. **entities**: List of entity types to run discovery:
   i. **domain_name**: The domain name must exist in the Compliance Engine. Note, any DL type entities Domain Name cannot be modified
   ii. **algorithm_name**: Any available algorithm whether out of the box or custom can be assigned to any entity type
   iii. **type**: These are the following types of entities are allowed:
      1. **"DL"**: A Deep Learning & NLP based discovery. All DL entities must have their correspondence Domain Name from the table listed here. Example payload:

         ```
         {
           "domain_name": "CREDIT_CARD",
           "algorithm_name": "CreditCard",
           "type": "DL"
         }
         ```

      2. **"context"**: Where users can provide their list of explicit values for discovery. Example payload:

         ```
         {
           "domain_name": "TITLE",
           "algorithm_name": "RandomValueLookup",
           "type": "context",
           "list": [
             "Mr.",
             "Mrs.",
             "Ms.",
             "Miss",
             "Madam",
             "Master"
           ]
         }
         ```

      3. **"pattern"** - The regex-based entity, users can add their regex criteria. Additionally, a list of fields can be supplied to provide further context to support regex discovery. Example payload:

         ```
         {
           "domain_name": "ZIP_CODE",
           "algorithm_name": "dlpx-core:CM Alpha-Numeric",
           "type": "pattern",
           "regex": "(\\b\\d{5}(?:\\-\\d{4})?\\b)",
           "meta_context": [
             "zip",
             "code"
           ]
         }
         ```

POST /profile-sets - Response JSON

```
{
  "set_id": 2,
  "name": "custom_profile_set",
  "description": "Different Algorithm Mapping",
  "exclusions": [
    "_id",
    "_id.oid",
    "$oid",
    "_id.$oid",
    "id"
  ],
  "entities": [
    {
      "domain_name": "FIRST_NAME",
      "algorithm_name": "dlpx-core:FirstName",
      "type": "DL"
    },
    {
      "domain_name": "LAST_NAME",
      "algorithm_name": "dlpx-core:LastName",
      "type": "DL"
    }
  ]
}
```

6. You can now start a profiler task using the `/tasks` POST API endpoint.
   Example 1: POST /tasks - Request JSON

```
{
  "connector_id": 1,
  "set_id": 1,
  "scan_depth": 1000,
  "unique_source_files_identifier": "file_identifier",
  "unload_split": 2,
  "file_type": "parquet"
}
```

Example 2: POST /tasks - Request JSON

```
{

  "connector_id": 16,

  "set_id": 1,

  "scan_depth": 4,

  "unique_source_files_identifier": "string",
  "unload_split": 2,

  "file_type": null,

  "table_list": [

    "people_directory"

  ]

}
```

Understanding the task payload parameters:

a. connector_id - The connector to get the source details from. The profiler will identify all files (recursively) within the source S3 path provided in the connector-info details.

b. set_id - The profiler set ID that the profiler tasks should run against.

c. scan_depth - The number of (random) rows in the parquet file that need to be analyzed by the profiler to determine what kind of sensitive data it is.

d. unique_source_files_identifier - The source key value that the resultant Hyperscale Parquet Connector dataset should be populated with.

e. unload_split - The unload split that the resultant Hyperscale Parquet Connector dataset should be populated with.

f. file_type - The file type should be “**parquet**”.

g. table_list - List of hive tables that will be scanned for profiling.

POST /tasks - Response JSON

```
{   "task_id": "11b92f0f-7c08-4768-97c5-17ce73213dc8",   "status": "RUNNING" }
```

8. The status of the task can be monitored using the `/tasks/{id}` GET API endpoint.
9. Once the status shows “SUCCESS”, the Hyperscale Parquet Connector dataset generated by the profiler is shown as part of the results.
Example 1: GET /tasks/{id} - Response JSON

```
{
  "task_id": "11b92f0f-7c08-4768-97c5-17ce73213dc8",
  "connector_id": 1,
  "data_set_id": null,
  "status": "SUCCESS",
  "set_id": 1,
  "scan_depth": 100,
  "file_type": "parquet",
  "unique_source_files_identifier": "file_identifier",
  "unload_split": 2,
  "results": {
    "connector_id": 1,
    "data_info": [
      {
        "source": {
          "unique_source_files_identifier": "file_identifier_1",
          "file_type": "parquet",
          "unload_split": 2,
          "source_files": [
            "customer/part-00000.gz.parquet",
            "customer/part-00001.gz.parquet",
            "customer/part-00002.gz.parquet",
            "customer/part-00003.gz.parquet",
            "customer/part-00004.gz.parquet",
            "customer/part-00005.gz.parquet",
            "customer/part-00006.gz.parquet",
            "customer/part-00007.gz.parquet",
            "customer/part-00008.gz.parquet",
            "customer/part-00009.gz.parquet"
          ]
        },
        "target": {
          "perform_join": true
        },
        "masking_inventory": [
          {
            "field_name": "c_last",
            "domain_name": "FIRST_NAME",
            "algorithm_name": "dlpx-core:FirstName"
          },
          {
            "field_name": "c_state",
            "domain_name": "LAST_NAME",
            "algorithm_name": "dlpx-core:LastName"
          },
          {
            "field_name": "c_phone",
            "domain_name": "TELEPHONE_NO",
            "algorithm_name": "dlpx-core:Phone US"
          }
        ]
      },
      {
        "source": {
          "unique_source_files_identifier": "file_identifier_2",
          "file_type": "parquet",
          "unload_split": 2,
          "source_files": [
            "district/part-00000.gz.parquet"
          ]
        },
        "target": {
          "perform_join": true
        },
        "masking_inventory": [
          {
            "field_name": "d_name",
            "domain_name": "LAST_NAME",
            "algorithm_name": "dlpx-core:LastName"
          },
          {
            "field_name": "d_street_2",
            "domain_name": "LAST_NAME",
            "algorithm_name": "dlpx-core:LastName"
          },
          {
            "field_name": "d_state",
            "domain_name": "LAST_NAME",
            "algorithm_name": "dlpx-core:LastName"
          }
        ]
      },
      {
        "source": {
          "unique_source_files_identifier": "file_identifier_7",
          "file_type": "parquet",
          "unload_split": 2,
          "source_files": [
            "orders/part-00000.gz.parquet",
            "orders/part-00001.gz.parquet",
            "orders/part-00002.gz.parquet",
            "orders/part-00003.gz.parquet",
            "orders/part-00004.gz.parquet"
          ]
        },
        "target": {
          "perform_join": true
        },
        "masking_inventory": [
          {
            "field_name": "o_id",
            "domain_name": "TELEPHONE_NO",
            "algorithm_name": "dlpx-core:Phone US"
          }
        ]
      },
      {
        "source": {
          "unique_source_files_identifier": "file_identifier_9",
          "file_type": "parquet",
          "unload_split": 2,
          "source_files": [
            "warehouse/part-00000.gz.parquet"
          ]
        },
        "target": {
          "perform_join": true
        },
        "masking_inventory": [
          {
            "field_name": "w_name",
            "domain_name": "CITY",
            "algorithm_name": "USCitiesLookup"
          },
          {
            "field_name": "w_street_1",
            "domain_name": "ZIP",
            "algorithm_name": "dlpx-core:CM Alpha-Numeric"
          },
          {
            "field_name": "w_state",
            "domain_name": "LAST_NAME",
            "algorithm_name": "dlpx-core:LastName"
          },
          {
            "field_name": "w_zip",
            "domain_name": "LAST_NAME",
            "algorithm_name": "dlpx-core:LastName"
          }
        ]
      }
    ]
  },
  "total": 16,
  "identified": null,
  "completion": 100,
  "elapsed_time": "0:06:47.837970",
  "start_time": "2023-12-14T13:32:18.913943",
  "end_time": "2023-12-14T13:39:06.756026",
  "date_created": "2023-12-14T13:32:18.913948",
  "date_last_updated": "2023-12-14T13:32:18.913950"
}
```

Example 2: GET /tasks/{id} - Response JSON

```
{
  "task_id": "7b891bdb-0bd9-455b-a27a-eeb5eec0d5b6",
  "connector_id": 16,
  "data_set_id": null,
  "status": "SUCCESS",
  "set_id": 1,
  "scan_depth": 4,
  "file_type": null,
  "unique_source_files_identifier": "string",
  "unload_split": 2,
  "results": {
    "connector_id": 16,
    "data_info": [
      {
        "source": {
          "unique_source_files_identifier": "string_1",
          "file_type": "parquet",
          "unload_split": 2,
          "source_files": [
            "people_directory"
          ]
        },
        "target": {
          "perform_join": true
        },
        "masking_inventory": [
          {
            "field_name": "first_name",
            "domain_name": "FIRST_NAME",
            "algorithm_name": "dlpx-core:FirstName"
          },
          {
            "field_name": "last_name",
            "domain_name": "LAST_NAME",
            "algorithm_name": "dlpx-core:LastName"
          }
        ]
      }
    ]
  },
  "total": 1,
  "identified": 1,
  "completion": 100,
  "elapsed_time": "0:00:18.952383",
  "start_time": "2024-08-01T20:52:13.397608",
  "end_time": "2024-08-01T20:52:32.375221",
  "date_created": "2024-08-01T20:52:13.397619",
  "date_last_updated": "2024-08-01T20:52:13.397622"
}
```

10. You can push the generated dataset directly from the profiler using the `/data-sets/{task_id}` POST API endpoint. The response contains the ID of the newly created dataset on the controller.
POST /data-sets/{task_id} - Response JSON

```
{
  "data_set_id": 1
}
```

The DL entities within the default Profiler-Set with their algorithms

Type	Domain Name	Algorithm	Description

DL	FULL_NAME	dlpx-core:FullName	Full name detection
DL	FIRST_NAME	dlpx-core:FirstName	First name
DL	LAST_NAME	dlpx-core:LastName	Last name
DL	EMAIL	dlpx-core:Email SL	Email address
DL	TELEPHONE_NO	dlpx-core:Phone US	Phone or Mobile number
DL	DOB	DateShiftDiscrete	Date of Birth
DL	IP ADDRESS	dlpx-core:CM Alpha-Numeric	IP Address
DL	CREDIT CARD	CreditCard	Credit Card
DL	ADDRESS	AddrLookup	Street Address
DL	CITY	USCitiesLookup	City name
DL	COUNTRY	NullValueLookup	Country name
DL	WEB	WebURLsLookup	URL or domain name
DL	DRIVING_LC	DrivingLicenseNoLookup	US driving license
DL	SSN	dlpx-core:CM Alpha-Numeric	Social Security Number

The other available DL entities

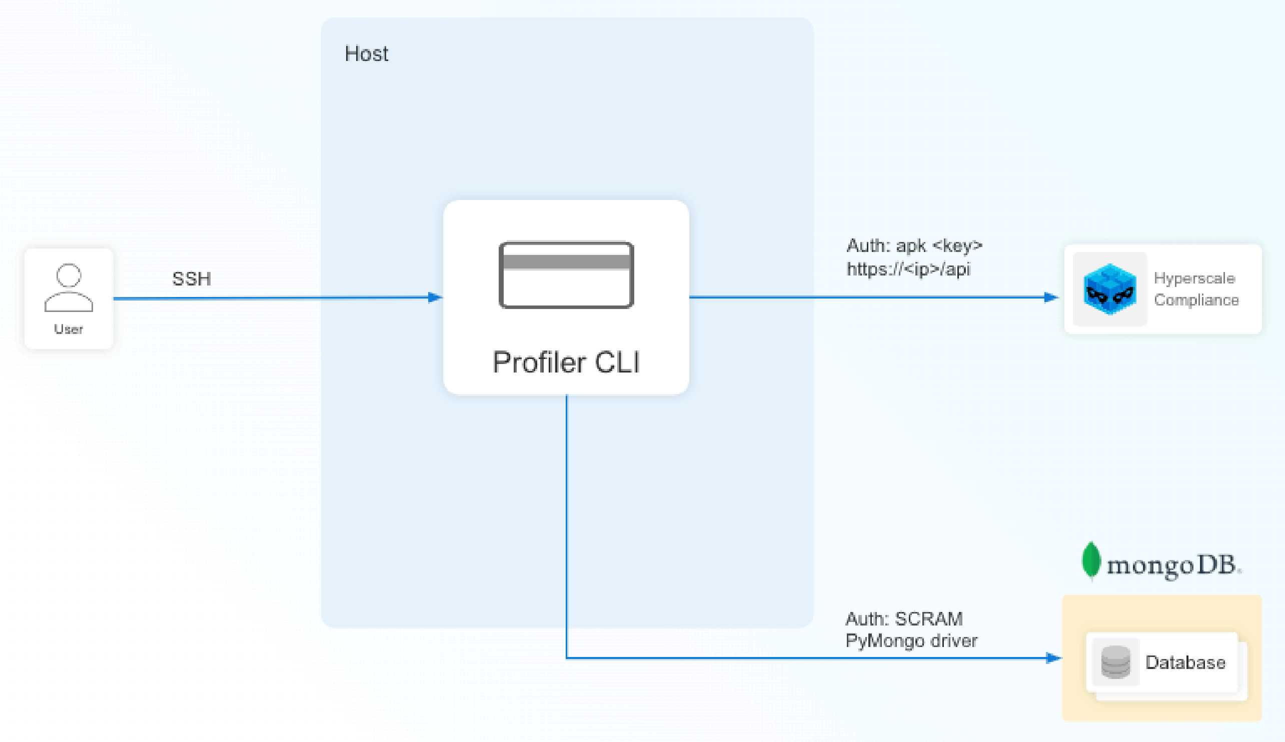
Type	**Domain Name**	**Description**
DL	STATE	State name
DL	STATE_CODE	State Code
DL	CRYPTO	Bitcoin address
DL	IBAN_CODE	The International Bank Account Number (IBAN)
DL	US_BANK_NUMBER	A US bank account number is between 8 to 17 digits.

DL	US_ITIN	US Individual Taxpayer Identification Number (ITIN
DL	US_PASSPORT	A US passport number with 9 digits

Mongo profiler

Identifying the structure of MongoDB documents within collections and applying appropriate masking algorithms using the Delphix Continuous Compliance Engine presents a significant challenge. Furthermore, when faced with a large volume of documents distributed across multiple collections at the source location, establishing a Hyperscale MongoDB connector dataset becomes notably difficult. The MongoDB Profiler emerges as a solution to tackle these complexities.

The primary aim of the MongoDB profiler is to conduct sensitive data discovery by leveraging collection metadata from the source database and generating a masking inventory through algorithmic recommendations. Additionally, it can facilitate the creation of a Hyperscale MongoDB connector dataset, thereby assisting in the development of a masking inventory for the source data.



The MongoDB profiler is tightly coupled with the Hyperscale MongoDB connector and cannot be used for other connectors.

Installation and setup (Mongo profiler)

Pre-requisites

1. Download the latest version of the MongoDB profiler from the Delphix download page.
2. Ensure that the host running the profiler has podman installed.
3. Set up the Hyperscale MongoDB Connector and add the required ConnectorInfo details.

Procedure

1. Untar the profiler downloaded from Delphix's download page. It should contain the container images for the profiler and the `podman-compose.yaml` file to run the profiler.

```
tar xzvf mongo-profiler.tar.gz
```

2. Create `db_connection_conf.yaml` using the template and populate the desired information.

```
cp db_connection_conf.yaml.template db_connection_conf.yaml

cat db_connection_conf.yaml
#
# Copyright (c) 2023 by Delphix. All rights reserved.
#
source_mongodb_uri: "mongodb://example.com:27017/?authMechanism=admin"
source_database: "test_db"
source_collection: "test_collection"
source_username: "test_user"
target_database: "target_db"
target_collection: "target_collection"
```

3. Check the help of `profiler.sh`.

```
./profiler.sh
Usage: ./profiler.sh [OPTIONS]

  Create masking inventory file.

Options:
  -o OPERATION          Operations
                          masking-inventory   Generate masking inventory JSON
file.
                          data-sets           Execute API /data-sets using the
                                                generated masking inventory
file.

  -c DB_CONN_CONF       Path of db_connection_conf.yaml
  [-u UNLOAD_SPLIT]       Refer Hyperscale docs for more details.
                          Required for "data-sets" operation only.

  [-e HSENGINE]           Hostname or IP of HyperScale Engine
  [-k KEY]                API Key for Controller
  [-d CONNECTOR-ID]       Connector ID
  [-help]                 Show this message and exit.


  Examples,

  #./profiler.sh -o masking-inventory -c DB_CONN_CONF.yaml

  #./profiler.sh -o data-sets -c DB_CONN_CONF.yaml -e HSENGINE.com -k APIKEY -d
ID -u UNLOAD_SPLIT
```

Executing a profiler task (mongo profiler)

1. Execute `profiler.sh` to generate masking inventory file.

```
$ ./profiler.sh -o masking-inventory -c db_connection_conf.yaml
Source Database Password:

..
[2024-04-29 13:45:55][INFO] Found podman environment.
..
2024-04-29 17:45:58,148 [INFO] (mongodb_connector:36) - Connected to MongoDB
successfully!
..
2024-04-29 17:46:04,703 [INFO] (masking_inventory_creator:130) - Generating
masking inventory at /app/sample_mflix_movies_masking_inventory.json
2024-04-29 17:46:05,215 [INFO] (masking_inventory_creator:71) - Masking
inventory created and saved to /app/sample_mflix_movies_masking_inventory.json
2024-04-29 17:46:05,239 [INFO] (mongodb_connector:94) - Connection to MongoDB
closed.
..
..
[2024-04-29 13:46:06][INFO] Recommending to review the masking inventory
file[sample_mflix_movies_masking_inventory.json]
[2024-04-29 13:46:06][INFO] in the current directory. Validate it and fix the
algorithms, if required.
```

2. Review the masking inventory file and modify it, if required.
3. Execute `profiler.sh` to create a dataset using the given masking inventory file.

```
$ ./profiler.sh -o data-sets -c db_connection_conf.yaml -e hsc-host-example.com
-k <API Key> -d <connector id> -u <unload split>


[2024-04-29 13:50:58][INFO] Getting source database details from
[db_connection_conf.yaml].
[2024-04-29 13:50:58][INFO] Getting target database details from
[db_connection_conf.yaml].
[2024-04-29 13:50:58][INFO] Payload for API /data-sets is stored in
payload_data_sets.json.
[2024-04-29 13:50:58][INFO] Executing API /data-sets.
[2024-04-29 13:50:58][INFO] Successfully executed API /data-sets.
[2024-04-29 13:50:58][INFO] Newly generated data-sets id:3
```

Snowflake profiler

This section covers the following topics:

- Installation and setup
- Executing a Snowflake profiler job
- Advanced configurations for Snowflake profiler

Installation and setup (Snowflake profiler)

This section covers the following topics:

- Kubernetes setup
- Docker Compose setup

Kubernetes setup

Prerequisites

1. Download the latest version of the Snowflake Profiler from the Delphix download page.
2. Ensure that the host running the profiler has Helm, kubectl and Docker installed. Also ensure it is set up to communicate to the Kubernetes cluster using either MicroK8s or Amazon's EKS.
3. Set up the Hyperscale Snowflake Connector and add the required `ConnectorInfo` details.
4. For accessing Snowflake warehouse, we will require a service account user with key pair authentication for enhanced authentication security as an alternative to basic authentication.
   a. To configure key-pair authentication for a snowflake service account, follow the steps provided in the Snowflake's configuring key-pair authentication.
   b. Use the following commands to generate base64 encoded value of encrypted private key and passphrase.
   To generate base64 encoded value of encrypted private key:

   ```
   echo -n `cat /home/delphix/keys/MKK_TEST_SFHSC_ADMIN_key.p8 | base64`
   ```

   To generate base64 encode value of passphrase:

   ```
   echo -n <passphrase> | base64 -w 0
   ```

5. Similarly, the profiler needs the Controller API Key to authenticate to the controller which also needs to be provided in base64 encoded format. To generate base64 encode value of Controller API key:

   ```
   echo -n “<controller_api_key>” | base64 -w 0
   ```

Procedure

1. Untar the profiler downloaded from Delphix download page. It should contain the Docker images for the profiler and the `snowflake-profiler-helm.tar`.

   ```
   tar -xf snowflake-profler.tar.gz
   ```

2. Load the `delphix-snowflake-profiler.tar` Docker image.

   ```
   cd snowflake-profiler
   docker load --input delphix-snowflake-profiler.tar
   ```

3. Push the image to a registry which can be accessed by the Kubernetes cluster.

   ```
   docker tag delphix-snowflake-profiler-service-app:1.0.0 image.registry.com/
   delphix-snowflake-profiler-service-app:1.0.0
   docker push image.registry.com/delphix-snowflake-profiler-service-app:1.0.0
   (‘docker login <registry>' may be required)
   ```

4. Untar the helm repository and change the directory to snowflake-profiler-helm

```
tar -xf snowflake-profiler-helm.tar.gz
cd snowflake-profiler-helm
```

5. Edit the values.yaml file
   a. Configure the `controllerURL` with the controller URL, IP or hostname.

   ```
   controllerURL: controller.delphix.com
   ```

   b. If required, configure the registry credentials.

   ```
   imageCredentials:
     username: <registry-username>
     password: <registry-password>
     email: <registry-user-email>
   ```

   c. Configure the image details.

   ```
   image:
     repository: image.registry.com/delphix-snowflake-profiler-service-app
     tag: 1.0.0
     pullPolicy: Always
   ```

   d. Configuring the credentials required to connect to Snowflake instance, similar to how it is configured for the Hyperscale Snowflake connector. If the Kubernetes secret is already configured in the same namespace (`.Values.namespace`), you can use the same.

   ```
   secret:
     storedSnowflakeSecretName: stored-secret-name
   ```

   Or you can configure the required values for `snowflakePrivateKey` , `snowflakePassphrase` and `controllerApiKey`

   ```
   secret:
     snowflakePrivateKey: user_encrypted_private_key_base64_encoded
     snowflakePassphrase: passhrase_base64_encoded
     controllerApiKey: controller_api_key_base64_encoded
   ```

   e. Optionally, see additional configurations for further details.
6. Start the snowflake profiler service.

```
helm install snowflake-profiler-helm <path_to_snowflake_profiler_helm_chart> -f
values.yaml
```

7. Create the Kubernetes Ingress to allow connection to the profiler.

Microk8s

For MicroK8s, create an ingress.yaml with the following values ensuring the namespace matches with profiler namespace:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hyperscale-snowflake-profiler-ingress
  namespace: hyperscale-snowflake-profiler-service
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTP"
    nginx.ingress.kubernetes.io/proxy-body-size: "50m"
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "600"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "600"
spec:
  ingressClassName: nginx
  rules:
http:
        paths:
path: /profiler
          pathType: Prefix
          backend:
            service:
              name: snowflake-profiler-k8s-service
              port:
                 number: 8080
```

Amazon AWS EKS

For Amazon AWS EKS, ensure the namespace matches with profiler namespace:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hyperscale-snowflake-profiler-ingress
  namespace: hyperscale-snowflake-profiler-service
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internal
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP":8080}]'
    alb.ingress.kubernetes.io/backend-protocol: HTTP
spec:
  rules:
http:
       paths:
path: /profiler
          pathType: Prefix
          backend:
            service:
              name: snowflake-profiler-k8s-service
              port:
                 number: 8080
```

8. Apply the Ingress configuration:

   ```
   kubectl apply -f ingress.yaml
   ```

9. Access the profiler Swagger UI at http://<host-ip>/profiler

Docker Compose setup

Prerequisites

1. Download the latest version of the Snowflake Profiler from the Delphix download page.
2. Ensure that the host running the profiler has Docker and Docker Compose installed.
3. Set up the Hyperscale Snowflake Connector and add the required `ConnectorInfo` details.
4. For accessing Snowflake warehouse, we will require a service account user with key pair authentication for enhanced authentication security as an alternative to basic authentication.
    a. To configure key-pair authentication for a Snowflake service account, follow the steps provided in the Snowflake's configuring key-pair authentication.
    b. Use the following commands to generate base64 encoded value of encrypted private key and passphrase.
    To generate base64 encoded value of encrypted private key:

    ```
    echo -n `cat /home/delphix/keys/MKK_TEST_SFHSC_ADMIN_key.p8 | base64`
    ```

    To generate base64 encode value of passphrase:

    ```
    echo -n <passphrase> | base64 -w 0
    ```

5. Similarly, the profiler needs the Controller API Key to authenticate to the controller which also needs to be provided in base64 encoded format. To generate base64 encode value of Controller API key:

    ```
    echo -n “<controller_api_key>” | base64 -w 0
    ```

Procedure

1. Untar the profiler downloaded from Delphix download page. It should contain the Docker images for the profiler and the `snowflake-profiler-compose.tar`

    ```
    tar -xf snowflake-profler.tar.gz
    ```

2. Load the `delphix-snowflake-profiler.tar` Docker image:

    ```
    cd snowflake-profiler
    docker load --input delphix-snowflake-profiler.tar
    ```

3. Untar the snowflake-profiler-compose.tar.gz

    ```
    tar -xf snowflake-profiler-compose.tar.gz
    cd snowflake-profiler-compose
    ```

4. Edit the .env file:
    a. Configure the ‘CONTROLLER_URL’ with the controller URL, IP or hostname:

```
CONTROLLER_URL= delphix-controller.com
```

b. Configure the credentials required to connect to Snowflake instance:

```
SNOWFLAKE_PRIVATE_KEY=user_encrypted_private_key_base64_encoded
SNOWFLAKE_PASSPHRASE=passhrase_base64_encoded
CONTROLLER_API_KEY=controller_api_key_base64_encoded
```

c. Optionally, see additional configurations for further details.

5. Start the Snowflake profiler service:

```
docker-compose -f profiler-docker-compose.yaml up -d
```

6. Access the profiler Swagger UI at http://<host-ip>:8080/profiler

Executing a Snowflake profiler job

Start a profiler job

1. Validate the configured connector-info's on the Hyperscale controller using the `/connector-info` GET API endpoint (or `/connector-info/<connector-id>`).
2. Configure a profiler-job using the `/profiler-jobs` POST API endpoint. The details required are like creating a Hyperscale Snowflake Connector data-set without the ‘masking-inventory’. It consists of two additional parameters:

   a. **max_rows_to_fetch** – This determines the number of rows to query when ‘VARINAT’ type data is detected in source table.

   b. **minimum_recognizer_score** – This helps with the overall result of the profiler. It determines the minimum recognizer score that is required for a column name/key name matching a prospective regex of a configured private field. For more information, see Tuning the profiler results.

```
{
  "connector_id": 1,
  "data_info": [
    {
      "max_rows_to_fetch": 5,
      "minimum_recognizer_score": 0.5,
      "source": {
        "warehouse_name": "SOURCE_WH",
        "database_name": "SOURCE_DB",
        "schema_name": "SOURCE_SCHEMA",
        "table_name": "SOURCE_TABLE",
        "stage_name": "SNOWFLAKE_EXTERNAL_STAGE",
        "max_file_size": 16777216
      },
      "target": {
        "warehouse_name": "TARGET_WH",
        "database_name": "TARGET_DB",
        "schema_name": "TARGET_SCHEMA",
        "table_name": "TARGET_TABLE"
      }
    }
  ]
}
```

3. After a profiler job is configured, an execution can be triggered using the `/execution` POST API end point. Keep a note of the execution ID received.

```
{
  "profiler_job_id": 1
}
```

Monitoring execution results

1. To monitor and get the status of an execution, use the `/execution/status/<execution_id>` GET API endpoint.

```
{
  "id": 1,
  "profiler_job_id": 1,
  "status": "SUCCEEDED",
  "start_time": "2025-01-13T10:33:32.824710",
  "end_time": "2025-01-13T10:33:47.114322",
  "error": null,
  "profiled_masking_inventory": [
    {
      "warehouse_name": "SOURCE_WH",
      "database_name": "SOURCE_DB",
      "schema_name": "SOURCE_SCHEMA",
      "table_name": "SOURCE_TABLE",
      "masking_inventory": [
        {
          "field_name": "AGE",
          "domain_name": "AGE",
          "algorithm_name": "dlpx-core:Age SL"
        },
        {
          "field_name": "DETAILS",
          "structured_data_format_id": 9
        }
      ]
    }
  ]
}
```

The ‘ `structured_data_format_id` ’ contains the masking inventory details for ‘ `VARIANT` ’ type data.

2. Optional - to validate the structured data format from the profiler, the `/structure-data-format/<structuredDataFormatId>` is extended and available on the controller.

Creating the Hyperscale Snowflake Connector Data Set

1. After an execution succeeds, verify the data set details using the `/data-set/<execution_id>` GET API endpoint.

```
{
  "connector_id": 1,
  "data_info": [
    {
      "source": {
        "warehouse_name": "SOURCE_WH",
        "database_name": "SOURCE_DB",
        "schema_name": "SOURCE_SCHEMA",
        "table_name": "SOURCE_TABLE",
        "stage_name": "SNOWFLAKE_EXTERNAL_STAGE",
        "max_file_size": 16777216
      },
      "target": {
        "warehouse_name": "TARGET_WH",
        "database_name": "TARGET_DB",
        "schema_name": "TARGET_SCHEMA",
        "table_name": "TARGET_TABLE"
      },
      "masking_inventory": [
        {
          "field_name": "AGE",
          "domain_name": "AGE",
          "algorithm_name": "dlpx-core:Age SL"
        },
        {
          "field_name": "DETAILS",
          "structured_data_format_id": 9
        }
      ]
    }
  ]
}
```

2. You can upload the data set to the controller using the profiler using `/data-set/upload/<execution_id>` POST API endpoint.

Advanced configurations for Snowflake profiler

This section covers the following topics:

- Advanced configurations for Kubernetes deployment
- Advanced configurations for Docker Compose deployment
- Tuning the profiler results

Advanced configurations for Kubernetes deployment

Using an existing namespace

1. The default value of the profiler's namespace is `hyperscale-snowflake-profiler-service`. To use an existing namespace and change the parameter `namespace` in values.yaml

```
namespace: hyperscale-services # example: same as hyperscale services
```

2. And set `createNamespace` to `false`

```
createNamespace: false
```

Creating a NodePort service instead of ClusterIP

1. You can create a `NodePort` service instead of `ClusterIP` service by editing the `service.type` value in values.yaml. This allows you to access the profiler without the need of Kubernetes ingress configuration.

```
service:
  type: NodePort
```

2. Run the following commands to identify the port that needs to be used to access the profiler. The node port can be identified from the output: `8080:<node_port>`

```
$ kubectl get services -namespace <profiler-namespace>
<profiler-namespace>   snowflake-profiler-k8s-service   NodePort    10.152.183.125
<none>        8080:30041/TCP           6s
```

3. The profiler can now be accessed using url http://<node_ip>:<node_port>

Managing the CPU and memory resources available to the profiler pod

1. The `resource` section which is commented by default can be used to manage the resources that should be attached to the profiler service

```
resources:
   requests:
     memory: "256Mi"
     cpu: "100m"
   limit:
     memory: "512Mi"
     cpu: "500m"
```

Advanced configurations for Docker Compose deployment

Changing the target port of the profiler service

1. By default, the profiler service is accessible at TCP port 8080 (http://<host_ip_name>:8080/)
2. If the port is unavailable, it can be changed by editing the `TARGET_PORT` available in .env file.

```
TARGET_PORT=<new_target_port>
```

3. The profiler can then be accessed via http://<host_ip_name>:<new_port>/

Tuning the profiler results

The profiler uses regular expressions to identify sensitive data fields based on the column names and key values of VARIANT type data. These regular expressions are configured in a file called `recognizers.yaml` and each expression is mapped to a masking algorithm and has a matching score. Here is an example configuration:

```
-  name: ACCOUNT_NO
    patterns:
    -   name: ACCOUNT_NO 1
         regex: '(?i)(?>(account|accnt|acct)_?-? ?(number|num|nbr|no|user))($|[ _-])'
         score: 0.67
    supported_entity: ACCOUNT_NO
    supported_language: en
```

To improve the results, you can edit the `recognizers.yaml` to add new regular expressions or to change the score associated to an existing configuration.

1. The default `recognizers.yaml` is available in `/app/src/config/recognizers.yaml`. But the execution will use the `recognizers.yaml` copied over to `/app/delphix/profiler`.
2. Copying the `recognizers.yaml` from the pod/container.
   a. For Kubernetes deployment:

   ```
   kubectl cp --namespace=<profiler-namespace> <snowflake-profiler-service-
   pod-name>:/app/src/config/recognizers.yaml ./recognizers.yaml
   ```

   b. For Docker Compose deployment:

   ```
   docker cp <snowflake-profiler-service-container>:/app/src/config/
   recognizers.yaml ./recognizers.yaml
   ```

3. Edit the `recognizers.yaml` to either add new regex regular expressions or to edit the scores. For instance, in the below example we added an identifier for city name where the column name is enclosed in single quotes:

   ```
   - name: CITY
     patterns:
       ...
       # New city identifier
       - name: CITY 3
         regex: (?i)(?>'(address_?-? ?city|city|city_?-? ?address)')
         score: 0.9
   ```

4. Copy the edited `recognizers.yaml` to the pod/container.
   a. For Kubernetes deployment:

```
kubectl cp --namespace=<profiler-namespace> ./recognizers.yaml
<snowflake-profiler-service-pod-name>:/app/delphix/profiler/
recognizers.yaml
```

b. For Docker Compose deployment:

```
docker cp ./recognizers.yaml <snowflake-profiler-service-container>:/app/
delphix/profiler/recognizers.yaml
```

Debugging the profiler

Debugging profiler for Kubernetes setup

Debugging profiler for Docker Compose setup

Debugging profiler for Kubernetes setup

List the helm deployments

Command: `helm list`

Example output:

```
NAME                                          NAMESPACE    REVISION
UPDATED                                        STATUS
      CHART                              APP VERSION
<snowflake-profiler-helm-installation-name>    default        1        2025-02-05 11:4
0:54.596712927 +0000 UTC        deployed      snowflake-profiler-helm-1.0.0      1.0.0
```

Checking the status of the helm deployment

Command: `helm status <snowflake-profiler-helm-installation-name>`

Example output:

```
NAME: snowflake-profiler
LAST DEPLOYED: Wed Feb  5 11:40:54 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Checking the pod status

Command: `kubectl get pods --namespace=<profiler-namespace>`

Example output:

```
NAME                             READY   STATUS       RESTARTS    AGE
<snowflake-profiler-pod-name>     1/1     Running       0           9m38s
```

Describing the pod details

Command: `kubectl describe pod <snowflake-profiler-pod-name> --namespace <profiler-namespace>`

Example output:

```
Name:             <snowflake-profiler-pod-name>
Namespace:        <profiler-namespace>
Priority:         0
Service Account:  default
...
```

Capturing the pod events

Command: `kubectl events --namespace <profiler-namespace>`

Example output:

```
LAST SEEN            TYPE      REASON
OBJECT                                                     MESSAGE
18s                  Normal    WaitForFirstConsumer    PersistentVolumeClaim/profiler-
yaml-pvc             waiting for first consumer to be created before binding
18s                  Normal    WaitForFirstConsumer    PersistentVolumeClaim/profiler-
database-pvc         waiting for first consumer to be created before binding
18s                  Normal    ScalingReplicaSet       Deployment/snowflake-profiler-
service               Scaled up replica set snowflake-profiler-service-857bfbbd65 to 1
18s                  Normal    SuccessfulCreate        ReplicaSet/snowflake-profiler-
service-857bfbbd65    Created pod: snowflake-profiler-service-857bfbbd65-lhp9p
...
```

Capturing the pod logs

Command: `kubectl logs -f <snowflake-profiler-pod-name> --namespace <profiler-namespace>`

Example output:

```
...
INFO: @ 2025-02-05 12:16:26,093 [MainProcess : 10 : main : : 60] : ****** Starting
the profiler **********
INFO: @ 2025-02-05 12:16:26,094 [MainProcess : 10 : main : : 61] : Sample INFO Log...
WARNING: @ 2025-02-05 12:16:26,094 [MainProcess : 10 : main : : 62] : Sample WARNING
Log...
ERROR: @ 2025-02-05 12:16:26,094 [MainProcess : 10 : main : : 63] : Sample ERROR
Log...
DEBUG: @ 2025-02-05 12:16:26,095 [MainProcess : 10 : main : : 64] : Sample DEBUG
Log...
...
```

Debugging profiler for Docker Compose setup

Checking container status

Command: `docker ps`

Example output:

```
CONTAINER ID   IMAGE                                           COMMAND
CREATED          STATUS          PORTS                                            NAMES
fc852c0efafa   delphix-snowflake-profiler-service-app:1.0.0   "/bin/bash -c 'sourc…"
5 seconds ago    Up 3 seconds    0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
<container_name>
```

Capturing the container logs

Command: `docker logs <container_name>`

Example output:

```
...
INFO:     @ 2025-02-05 15:19:15,914 [MainProcess : 13 : main : <module> : 60] :
****** Starting the profiler **********
INFO:     @ 2025-02-05 15:19:15,915 [MainProcess : 13 : main : <module> : 61] :
Sample INFO Log...
WARNING:  @ 2025-02-05 15:19:15,915 [MainProcess : 13 : main : <module> : 62] :
Sample WARNING Log...
ERROR:    @ 2025-02-05 15:19:15,916 [MainProcess : 13 : main : <module> : 63] :
Sample ERROR Log...
DEBUG:    @ 2025-02-05 15:19:15,916 [MainProcess : 13 : main : <module> : 64] :
Sample DEBUG Log...
...
```