Hyperscale Compliance Home

Hyperscale Compliance

Exported on 08/15/2023

Table of Contents

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.

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

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 engine on a single-node Kubernetes cluster by using the helm charts. For more details, refer to the 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 Hyperscale Compliance API documentation.

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 Engine 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 8.0.0

Key	Summary
HM-1 606	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-1 787	Oracle: Load service holds a lock on class level, causing parallel jobs stuck in the flow of preload
HM-1 814	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-1 617	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-1 684	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-1 513	MSSQL - Slowness while performing load with large tables(more than 4M rows and 10 Columns)
HM-1 521	MSSQL - Post load fails with 'transaction log is full due to 'ACTIVE_TRANSACTION' for large tables
HM-1 608	SQLLDR doesn't load the data when the table has enabled triggers owned by a different user

Key	Summary
HM-1 645	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-1 656	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-1 561	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-9 71	For HTTP protocol-type requests, trust store fields are accepted and displayed in response.
HM-1 472	Oracle Unload service doesn't release the lock if fails to initialize the connection pool and the job stuck in a runnning state
HM-1 520	Masking service: Starting execution throwing NPE after container restart
HM-1 522	Update Oracle JDBC Driver to 21.3.0.0

Release 4.1.0

Key	Summary
HM-1 155	Diagnosability: How do I tell which masking job on the masking engine relates to the failed message on the HS Jobs API status
HM-1 168	The error text in Post Load failures is misleading/unclear

Key	Summary
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
HM-858	Status of sub-task coming wrong when overall execution failed
HM-932	Suppress the password message for the controller log

Key	Summary
HM-1 138	The description in the swagger doesn't match the API call
HM-1 140	The error needs more information to diagnose a connector issue.

Release 3.0.0.1

Key	Summary
HM-8 58	Status of sub-task coming wrong when overall execution failed
HM-8 73	Intermittently there is a mismatch in loaded_rows displayed in the load task vs the actual rows loaded in the target table
HM-9 15	Load: driver support plugin throws ORA-02297: cannot disable constraint - dependencies exist error for foreign key

Release 3.0.0

Key	Summary
HM-2 94	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 8.0.0

Key	Summary	Workaround
HM-663	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-812	Application on the registered masking engine is not deleted with the cleanup	None
HM-1196	MSSQL - The job is stuck in a running state while using a filter key having NULL values.	None
HM-1239	MSSQL- Unload fails for a schema or table name having a ']' character in it	None
HM-1240	MSSQL - Unload fails for a column having '.' in its name	None
HM-1246	MSSQL - Unload fails with UnknownFormatConversionException for a table having % in its name	None
HM-1382	Oracle: Dataset having any one entry with invalid schema leaves indexes of other tables as UNUSABLE	None
HM-1397	Oracle Load fails for tables having triggers only with SQL*Loader-937 error	None
HM-1463	MSSQL - Load Service fails when the Table name contains '	None
HM-1523	MSSQL - Job fails while loading masked VARBINARY data	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Key	Summary	Workar ound
HM-19 29	Postload task status shows the old error message for the error field after restarting the execution.	None

Release 7.0.0.0

Key	Summary	Workar ound
HM-66 3	Oracle: The load process is failing with "Error disabling constraint" for identity columns	None
HM-81 2	Application on the registered masking engine is not deleted with the cleanup	None
HM-11 96	MSSQL - The job is stuck in a running state while using a filter key having NULL values.	None
HM-12 39	MSSQL- Unload fails for a schema or table name having a ']' character in it	None
HM-12 40	MSSQL - Unload fails for a column having '.' in its name	None
HM-12 46	MSSQL - Unload fails with UnknownFormatConversionException for a table having % in its name	None
HM-13 82	Oracle: Dataset having any one entry with invalid schema leaves indexes of other tables as UNUSABLE	None
HM-13 97	Oracle Load fails for tables having triggers only with SQL*Loader-937 error	None
HM-14 63	MSSQL - Load Service fails when the Table name contains '	None
HM-15 23	MSSQL - Job fails while loading masked VARBINARY data	None
HM-17 05	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-812	Application on the registered masking engine is not deleted with the cleanup	None
HM-1196	MSSQL Job is stuck in a running state while using a filter key having NULL values.	None
HM-1239	MSSQL- Unload fails for a schema or table name having a ']' character in it	None
HM-1240	MSSQL - Unload fails for a column having '.' in its name	None
HM-1246	MSSQL - Unload fails with UnknownFormatConversionException for a table having % in its name	None
HM-1382	Oracle: Dataset having any one entry with invalid schema leaves indexes of other tables as UNUSABLE	None
HM-1397	Oracle: Dataset having any one entry with invalid schema leaves indexes of other tables as UNUSABLE	None
HM-1463	MSSQL - Load Service fails when the Table name contains '	None
HM-1523	MSSQL - Job fails while loading masked VARBINARY data	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-812	Application on the registered masking engine is not deleted with the cleanup	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-1196	MSSQL Job is stuck in a running state while using a filter key having NULL values.	None
HM-1239	MSSQL- Unload fails for a schema or table name having a ']' character in it	None
HM0-1240	MSSQL - Unload fails for a column having '.' in its name	None

Key	Summary	Workaround
HM-1246	MSSQL - Unload fails with UnknownFormatConversionException for a table having % in its name	None
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-1382	Oracle : Dataset having any one entry with invalid schema leaves indexes of other tables as UNUSABLE	None
HM-1397	Oracle Load fails for table having triggers only with SQL*Loader-937 error	None
HM-1463	MSSQL - Load Service fails when Table name contains '	None
HM-1512	received_objects in Load step are more than succeeded_objects in Masking step intermittently	None
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-1523	MSSQL - Job fails while loading masked VARBINARY data	None
HM-1528	Initial delay in updating response of unload/masking/load execution objects with large number of tables	None

Key	Summary	Workaround
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
HM-812	Application on registered masking engine is not deleted with cleanup	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-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.
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-812	Application on registered masking engine is not deleted with cleanup	None

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-1366	NPE displayed in hyperscale masking service logs just before cleanup is performed	None
HM-1382	Dataset having any one entry with invalid schema leaves indexes of other tables as UNUSABLE	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.

Key	Summary	Workaround
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-812	Application on registered masking engine is not deleted with cleanup	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

Key	Summary	Workaround
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.

Key	Summary	Workaround
HM-789	Error message upon not setting 'ssl' field to False indicates 'insecure_ssl' property which no longer exists in the schema	None
HM-812	Application on registered masking engine is not deleted with cleanup	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-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

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.

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 engine 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.
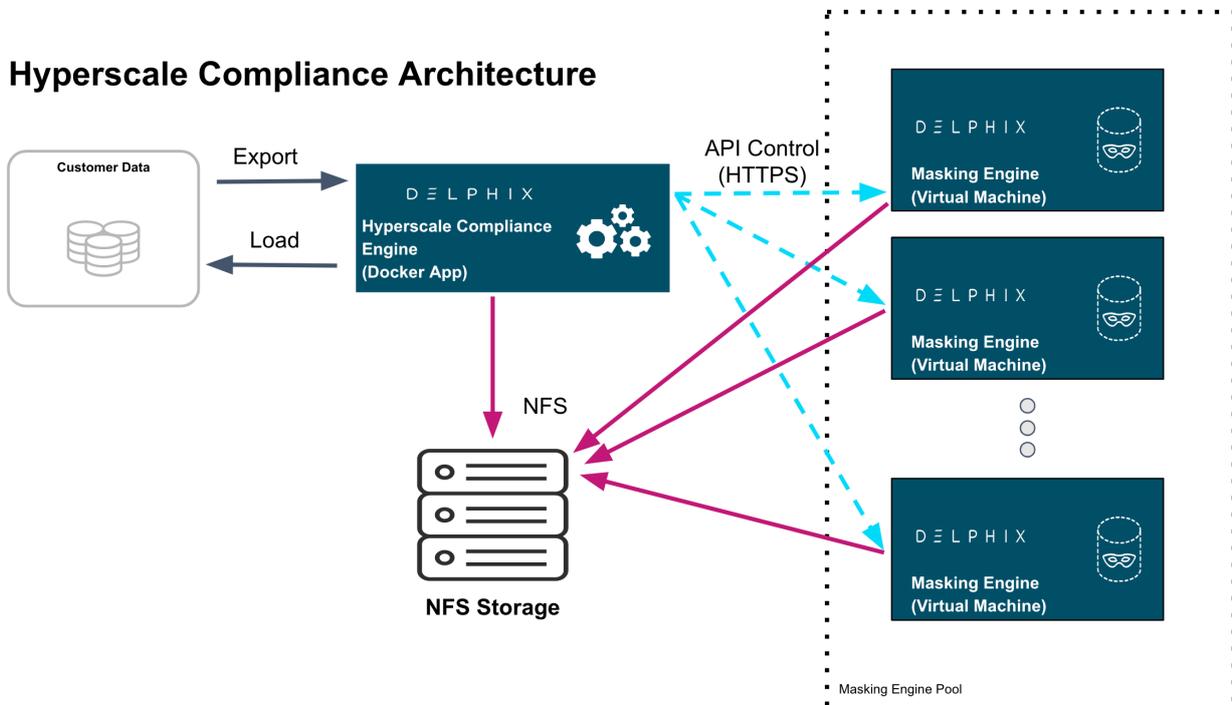
Hyperscale Compliance components

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

Hyperscale Compliance Engine

The Hyperscale Compliance Engine 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 Engine 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 Engine. 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 Engine 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 Engine is responsible for unloading data from the source data source into a series of files located in the staging area. The Hyperscale Compliance Engine requires network access to the source from the host running the Hyperscale Compliance Engine 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 Engine's restart ability feature to restart the job from the point of failure in the load/post-load step.

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
- 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
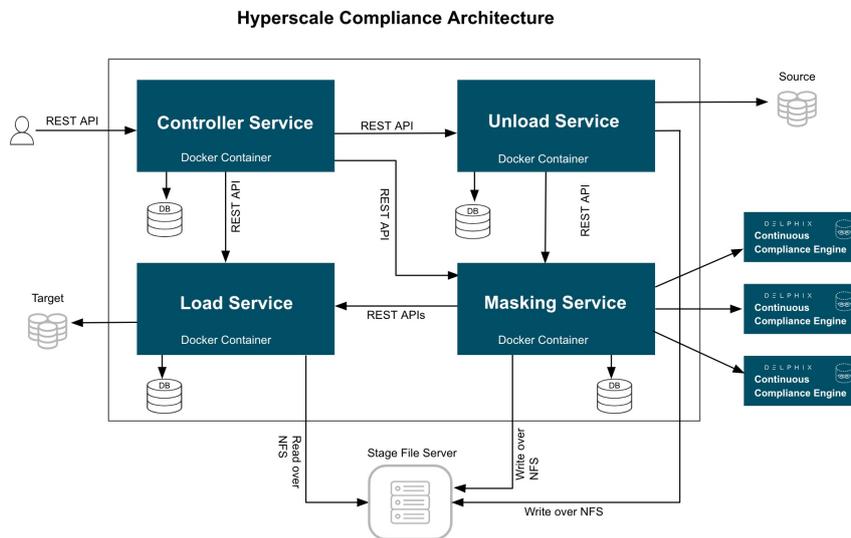
Controller service

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)

> ⓘ 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 Configuration Settings .

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
- XML (only unload/load))
- VARBINARY (only unload/load)
- SMALLINT
- SMALLMONEY
- MONEY
- BIGINT
- NVARCHAR
- TINYINT
- NUMERIC(X,Y)
- DECIMAL(X,Y)
- FLOAT
- NCHAR
- BIT
- NTEXT
- MONEY

Property values

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

For default values, see Configuration Settings .

Known limitations

Below constraints and indexes are not disabled prior to load and are enabled back after the load process:

- Partitioned Indexes

Supported platforms

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

Continuous Compliance Engine versions

- 10.0.0.0 and later

> ⓘ 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.

Continuous Data Engine versions

- 8.0.0.0 and later

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 Engine.

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

Type (Inbound/Outbound)	Port	Reason
Inbound and Outbound	80	HTTP connections to/from the Hyperscale Compliance Engine 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 Engine 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 Engine host.

Deployment

This section covers the following topics:

- Docker Compose
- Kubernetes

Docker Compose

This section covers the following topics:

- Host requirements (Docker Compose)
- Installation and setup (Docker Compose)
- Upgrading the Hyperscale Compliance Engine (Docker Compose)
- How to generate a support bundle (Docker Compose)

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 Engine 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 Engine.
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 Engine 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, 16 GB of memory, 100GB data disk. • Recommended: 16 vCPU, 128GB of memory, 500GB data disk.	OS disk space: 50 GB

Installation and setup (Docker Compose)

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

Hyperscale Compliance installation

Pre-requisites

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

- Download the Hyperscale tar file (`delphix-hyperscale-masking-8.0.0.tar.gz`) from download.delphix.com.
- You must create a user that has permission to install Docker and Docker Compose.
- Install Docker on 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 Engine 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 Engine to be able to read or execute from the directory.

Procedure

Perform the following procedure to install the Hyperscale Compliance Engine.

1. Unpack the Hyperscale tar file.

```
tar -xzf delphix-hyperscale-masking-8.0.0.tar.gz
```

2. After unpacking, you should see 7 docker image tar files. The `controller-service.tar` , `masking-service.tar` , and `proxy.tar` are common docker images for both Oracle and MS SQL data source masking. The `unload-service.tar` and `load-service.tar` image files are required for Oracle unload/load services whereas the `mssql-unload-service.tar` and `mssql-load-service.tar` image files are required for MS SQL data source masking. As such, proceed to load the concerned images into docker:

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
```

3. Create an NFS shared mount, that will act as a **Staging Area**, on the Hyperscale Compliance Engine host where the Hyperscale Compliance engine will perform read/write/execute operations:

1. Create a 'Staging Area' directory. For example: `/mnt/hyperscale/staging_area` . The user(s) within each of the docker containers part of the Hyperscale Compliance Engine 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 'staging_area' directory, along with the directory(`hyperscale`) one level above, require the following permissions, based on the UID/GID of the OS user, so that the Hyperscale Compliance Engine 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, along with the directory(`hyperscale`) one level above, 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, along with the directory(`hyperscale`) one level above, must have a GID of 50 and 770 permission mode.
2. Mount the NFS shared directory on the staging area directory(`/mnt/hyperscale/staging_area`). 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.

> ⓘ The directory created in step 3a ('staging_area') will be provided as the `mountName` and the corresponding shared path from the NFS file server as the `mountPath` in the MountFileSystems API.

4. Configure the following docker container volume bindings for the docker containers by editing the `docker-compose.yaml` file from tar:

1. 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 Engine container path(`/etc/hyperscale`) as a volume binding under the 'volumes' section.
2. [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.
3. [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:
   - The logs generated for each service i.e. unload, controller, masking, and load services.

- The sqlldr utility logs and control files at opt/sqlldr location in the load-service container.
- The file-upload folder at `/opt/delphix/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 and MS SQL data source masking:

For Oracle data source masking:

```
version: "3.7"
services:
  controller-service:
    image: delphix-controller-service-app:${VERSION}
    healthcheck:
      test: 'curl --fail --silent http://localhost:8080/actuator/health | grep UP ||
exit 1'
      interval: 30s
      timeout: 25s
      retries: 3
      start_period: 30s
    depends_on:
      - unload-service
      - masking-service
      - load-service
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-controller-data:/data
      - /home/hyperscale_user/logs/controller_service:/opt/delphix/logs
    - /home/hyperscale_user/uploads:/opt/delphix/uploads
    environment:
      - API_KEY_CREATE=${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=${SKIP_LOAD_SPLIT_COUNT_VALIDATION:-false}
  unload-service:
    image: delphix-unload-service-app:${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
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/unload_service:/opt/delphix/logs
  masking-service:
    image: delphix-masking-service-app:${VERSION}
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-masking-data:/data
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/masking_service:/opt/delphix/logs
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${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
      - /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:
      - "443:443"
    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:

```yaml
version: "3.7"
services:
  controller-service:
    image: delphix-controller-service-app:${VERSION}
    healthcheck:
      test: 'curl --fail --silent http://localhost:8080/actuator/health | grep UP ||
exit 1'
      interval: 30s
      timeout: 25s
      retries: 3
      start_period: 30s
    depends_on:
      - unload-service
      - masking-service
      - load-service
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-controller-data:/data
      - /home/hyperscale_user/logs/controller_service:/opt/delphix/logs
    - /home/hyperscale_user/uploads:/opt/delphix/uploads
    environment:
      - API_KEY_CREATE=${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=${SKIP_LOAD_SPLIT_COUNT_VALIDATION:-false}
  unload-service:
    image: delphix-mssql-unload-service-app:${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
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/unload_service:/opt/delphix/logs
```

```
  masking-service:
    image: delphix-masking-service-app:${VERSION}
    init: true
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-masking-data:/data
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/masking_service:/opt/delphix/logs
    environment:
      - LOGGING_LEVEL_COM_DELPHIX_HYPERSCALE=${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=${DATE_TIMESTAMP_FORMAT:-yyyy-MM-dd HH:mm:ss.SSSS}
    networks:
      - hyperscale-net
    restart: unless-stopped
    volumes:
      - hyperscale-load-data:/data
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/load_service:/opt/delphix/logs
  proxy:
    image: delphix-hyperscale-masking-proxy:${VERSION}
    init: true
    networks:
      - hyperscale-net
    ports:
      - "443:443"
    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.

```
docker-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.

```
docker-compose ps
```

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

```
docker logs -f service_container_name>
```

> (i) Service container name can be accessed by output of the command `docker-compose ps` .

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

```
sudo docker-compose down
```

7. Once the application starts, an API key will be generated that will be required to authenticate with the Hyperscale Compliance engine. 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 you could use the following 'docker' command to retrieve the logs.

```
docker logs -f <service_container_name>
```

> (i) Service container name can be accessed by output of the command `docker-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 Engine, 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 Engine (Docker Compose)

Prerequisite

Before upgrading, ensure you have downloaded the Hyperscale Compliance 8.0.0 tar bundle from the Delphix Download website.

How to upgrade the Hyperscale Compliance Engine

Perform the following steps to upgrade the Hyperscale Compliance Engine to the 8.0.0 version:

1. Run `cd /<hyperscale_installation_path>/` and `docker-compose down` to stop and remove all the running containers.
2. Run `docker rmi -f $(docker images -q)` to delete all the existing images.
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. Untar the patch tar in your existing installation path. `tar -xzvf delphix-hyperscale-masking-8.0.0.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 `docker-compose.yaml` file 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.
6. In the `.env` file supplied with the bundle, set the VERSION property as 9.0.0 (i.e. VERSION=8.0.0).
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
     ```

8. Run `docker-compose up -d` to create containers.
9. Ensure all your mount(s) are configured and accessible, before running a job.

> ⓘ Existing data remains intact after the degradation.

How to generate a support bundle (Docker Compose)

> ⚠ You must install bash shell and yq to generate a Hyperscale support bundle if it is not already installed. For more information on downloading yq, refer to yq Downloads page on GitHub.

1. Find the "generate_support_bundle.sh" script

- Login to Hyperscale VM for which you want to generate the support bundle.
- generate_support_bundle.sh" file is bundled with the release tar file. You can find this script under `tools/ support-scripts` folder, (present under the directory, where you will untar the release tar file on Hyperscale Engine). For example, `/path_to_untarred_hyperscale_product/tools/support- scripts`.

   Example:

```
dlpxuser@delphix:~/test$ cd tools/support-scripts/
dlpxuser@delphix:~/test$ ls -ltr
total 48
-rwxr-xr-x  1 delphix  staff   823 Jul  7 09:55 generate_support_bundle.sh
-rwxr-xr-x  1 delphix  staff   463 Jul  7 09:55 container_information.sh
-rwxr-xr-x  1 delphix  staff  5597 Jul  7 09:55 collect_container_support_info.sh
-rw-r--r--  1 delphix  staff  5316 Jul  7 09:55 README.md
```

2. Modify the "container_information.sh" script parameters

   Change the container_names, mount_path and docker_compose_file_path accordingly. The container_names should be in the same order as mentioned in the below example.

   Example:

```
container_names=(hyperscale-masking-controller-service_1  hyperscale-masking_unload-
service_1 hyperscale-masking_masking-service_1 hyperscale-masking_load-service_1)
mount_path=/home/delphix/hyperscale
docker_compose_file_path=/home/delphix/docker-compose.yaml
```

> 🗒 - container_names: Can be found by running `docker ps` command
>
> - mount_path: Absolute path configured for mount directory in `docker-compose` file which is mapped to `/etc/hyperscale`
>
> - docker_compose_file_path: Absolute path for `docker_compose.yaml` file

3. Execute the "generate_support_bundle.sh" script

- Execute the "generate_support_bundle.sh" script from `tools/support-scripts/` folder.

Example:

```
dlpxuser@delphix:~/test/tools/support-scripts/$ ./generate_support_bundle.sh
....
Generating support bundle tar file...
....
```

- Enter the "Password" when prompted.

4. Find the generated support bundle tar file

The resulting support bundle will be located at `/etc/hyperscale/hyperscale-support-****.tar.gz` inside the container. This means the tar file is generated under the path which is mapped to `/etc/hyperscale` in `docker-compose` file and is directly accessible from Hyperscale VM.

Example:

```
dlpxuser@delphix:~/test$ ls -ltr ../hyperscale/
total 316
drwxrwxrwx 5  1004  1005   4096 Feb  9 10:14 aks-mount
-rw-r--r-- 1 65436 staff 104189 Feb 17 08:52 hyperscale-support-
<current_timestamp>.tar.gz
```

The support bundle tar file contains the following information:

- Hyperscale Logs
- The output of mpstat for CPU utilization info.
- The output of proc/meminfo for memory info.
- The output of proc/cpuinfo for cpu info.
- Files to show the memory limit for the application container and the max usage of the app container in bytes.
- Redacted database file to restore the Hyperscale VM
- Docker compose file

> ⓘ
> - The script generate_support_bundle.sh generates a bare-bones support bundle from a Hyperscale engine running in docker.
> - Execute the generate_support_bundle.sh from the untar location.
> - The resulting support bundle will be at `/etc/hyperscale/hyperscale-support-****.tar.gz` inside the container. This means the tar file is generated under a path that is mapped to /etc/hyperscale in `docker-compose` file and is directly accessible from Hyperscale VM.
> - The user should have privileges or permission to execute the docker command in order to generate the support bundle.

Kubernetes

This section covers the following topics:

- Host requirements (Kubernetes)
- Installation and setup (Kubernetes)
- Bootstrapping API keys (Kubernetes)
- Hyperscale logs (Kubernetes)
- Limitations (Kubernetes)
- How to generate a support bundle (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.
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 Engine will be able to read the source files and write the target files. For more information, see NFS Server Installation.
Staging area directory permissions	The 'staging_area' directory, along with the directory(`hyperscale`) one level above, require the following permissions, based on the UID/GID of the OS user so that the Hyperscale Compliance Engine 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, along with the directory(`hyperscale`) one level above, 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, along with the directory(`hyperscale`) one level above, 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, 16 GB of memory, 50GB OS disk. **Recommended:** 16 vCPU, 128GB of memory, 50GB OS disk.	

Installation and setup (Kubernetes)

Installation requirements

To deploy Hyperscale Compliance via Kubernetes, a running Kubernetes cluster is required to run, the `kubectl` command 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	
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.

Installation

Download the HELM charts

The latest version of the chart can be pulled locally with the following command:

```
curl -XGET https://dlpx-helm-hyperscale.s3.amazonaws.com/hyperscale-helm-8.0.0.tgz -o hyperscale-helm-8.0.0.tgz
```

This command will download a file with the name `hyperscale-helm-8.0.0.tgz` in the current working directory. The downloaded file can be extracted using the following command:

```
tar -xvf hyperscale-helm-8.0.0.tgz
```

This will extract into the following directory structure:

```
hyperscale-helm
    |- values.yaml
```

```
|- README.md
|- Chart.yaml
|- templates
    |-<all templates files>
```

Configure registry credentials for Docker images

For pulling the Docker images from the registry, temporary credentials would need to be configured/overridden in the `values.yaml` file. For getting the temporary credentials, visit the Hyperscale Compliance Download page and log in with your customer login credentials. Once logged in, select the Hyperscale HELM Repository link and accept the Terms and Conditions. Once accepted, login credentials 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>

imageCredentials:
username: <username>
password: <password>
```

Override default values in values.yaml

`hyperscale-helm` is the name of the folder which was extracted in the previous step. In the above directory structure, the `values.yaml` file contains all of the configurable properties with their default values. These default values can be overridden while deploying Hyperscale Compliance, as per the requirements. If the `values.yaml` file needs to be overridden, create a copy of `values.yaml` and edit the required properties.

While deploying Hyperscale Compliance, `values.yaml` file can be overridden using either of the following commands:

```
helm install hyperscale-helm -f <path to edited values.yaml> <directory path of the
extracted chart>
helm install hyperscale-helm <directory path of the extracted chart> --set
<property1>=<value1> --set <property2>=<value2>
```

Configure Helm chart properties of importance

A few commonly used properties for each service (controller, masking, unload and load) with their default values are listed in `values.yaml` . You may find details of these properties on the Configuration Settings page. The following sections talk about some of the important properties that will need to be configured correctly for a successful deployment.

Configure the staging area

By default, a path(`/dlpxdata`) local to the Kubernetes cluster node will be used, via persistent volume claims, to mount the staging area path inside the pods. Override the path by setting up the desired local storage path with the **localStoragePath** property.

If the cluster needs to mount an NFS shared path that will act as the staging area, override the **nfsStorageHost** and **nfsStorageExportPath** properties.

Configure the correct unload/load images for your dataset type

By default, the helm installation will create the Kubernetes pods for the Oracle unload and load services. To have helm create unload and load pods for the other connector types, for example, MSSQL unload and load services, override the following values in the `values.yaml` file:

1. `unload.imageName=mssql-unload-service`
2. `load.imageName=mssql-load-service`

Configure the instantclient path(Applicable for Oracle unload/load):

By default, a path(`/dlpxdata`) local to the Kubernetes cluster node will be used, via persistent volume claims, to mount the Oracle **instantclient** path inside the pods. Override the path by setting up the desired **instantclient** path with the **instantClientStoragePath** and **instantClientRootDirName** properties.

If the cluster needs to mount an NFS shared path that will contain the **instantclient** binaries, override the **nfsInstantClientHost** and **nfsInstantClientExportPath** properties.

Service level properties:

> ⓘ 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.

Configuring the ingress controller

Assuming an ingress controller configuration on the Kubernetes cluster is present when accessing Hyperscale Compliance after the deployment, the ingress controller rule needs to be added for proxy service, along with port 443 (if SSL is enabled) and port 80 (if SSL is disabled).

Additionally, the following annotations will need to be set(this assumes that Kubernetes Ingress NGINX Controller is being used):

1. nginx.ingress.kubernetes.io/backend-protocol=HTTPS
2. nginx.ingress.kubernetes.io/proxy-body-size=50m
3. nginx.ingress.kubernetes.io/proxy-connect-timeout=600
4. nginx.ingress.kubernetes.io/proxy-read-timeout=600
5. nginx.ingress.kubernetes.io/proxy-send-timeout=600

If an ingress controller has not been assigned, then a new ingress resource, with the above requirements, can be created with the following `kubectl` command:

```
kubectl create ingress https-ingress --namespace=<namespace-name> --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
```

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-9.0.0
```

```
$ kubectl get pods --namespace=<namespace-name>
NAME                              READY    STATUS             RESTARTS      AGE
proxy-7786995dc6-2pzt9            1/1      Running            1 (81s ago)   2m28s
controller-service-698ddd77fd-wt65x   0/1      ContainerCreating  0             2m28s
masking-service-6c95cf474d-rjl6l  1/1      Running            0             2m28s
unload-service-6c5dcb9f48-mnk4p   1/1      Running            0             2m28s
load-service-7bfc864cb8-2w6mt     1/1      Running            0             2m28s
```

Bootstrapping API keys (Kubernetes)

Once the application starts, an API key will be generated that will be required to authenticate with the Hyperscale Compliance engine. 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 Engine, 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 `apiKeyEnabled` property in the installed helm chart by running the following command.

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

Hyperscale Compliance 8.0.0 does not support deploying/scaling up the services to multiple nodes in a Kubernetes cluster. This support will be added in a future release.

How to generate a support bundle (Kubernetes)

> ⚠ You must install bash shell and yq to generate a Hyperscale support bundle if it is not already installed. For more information on downloading yq, refer to yq Downloads page on GitHub.

1. Find the "generate_support_bundle.sh" script

- Login to Hyperscale VM for which you want to generate the support bundle.
- generate_support_bundle.sh" file is bundled with the helm package file. You can find this script under the `tools/` directory(present under the directory, where you will untar the helm package tar file). For example, `/path_to_hyperscale_helm/tools/` .

Example:

```
dlpxuser@delphix:~/test$ cd tools/support-scripts/
dlpxuser@delphix:~/test$ ls -ltr
total 16
-rw-r--r--  1 delphix staff  1032 May  3 13:09 generate_support_bundle.sh
-rw-r--r--  1 delphix  staff    54 May  3 13:12 values-redact.properties
```

2. Modify the "values-redact.properties"

The `values-redact.properties` file contains property values (in `values.yaml` file) which are sensitive and should be redacted before adding them to the support bundle. By default, it includes image repository-related properties. The property name will follow a format like `.<rootProperty>.<childProperty>.<innerChildProperty>` .

Example:

```
.unload.loggingLevelRoot // here we want to redact loggingLevelRoot property of
unload
```

3. Execute the "generate_support_bundle.sh" script

Execute the "generate_support_bundle.sh" script from `hyperscale-helm/tools/` directory.

Example:

```
delphix@ip-xx-xxx-xxx-xxx:~$ ./hyperscale-helm/tools/generate_support_bundle.sh
/home/delphix/support/values.yaml created.
.....
Generating support bundle tar file...
.....
```

4. Find the generated support bundle tar file

The resulting support bundle (`hyperscale-support-****.tar.gz`) will be located at the same level where the `hyperscale-helm` was extracted

Example:

```
delphix@ip-10-110-254-92:~$ ls
hyperscale-helm  hyperscale-helm-9.0.0-6.tgz  hyperscale-support-20230502-14-36-36.ta
r.gz
```

The support bundle tar file contains the following information:

- Hyperscale Logs
- The output of `mpstat` for CPU utilization info.
- The output of `proc/meminfo` for memory info.
- The output of `proc/cpuinfo` for CPU info.
- Files to show the application container's memory limit and the app container's max usage in bytes.
- Redacted database file to restore the Hyperscale VM
- Redacted `values.yaml` file

NFS server installation

The Hyperscale Compliance engine 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 engine and the Continuous Compliance Engines. The following are the supported ways by which the filesystem can be shared over NFS(NFSv3/NFSv4):

Delphix Virtualization Engine empty VDB

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

Virtualization 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 Engine's 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 engine's 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 engine's 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 engine's 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 engine and the Continuous Compliance Engines part of the Continuous Compliance Engine cluster.
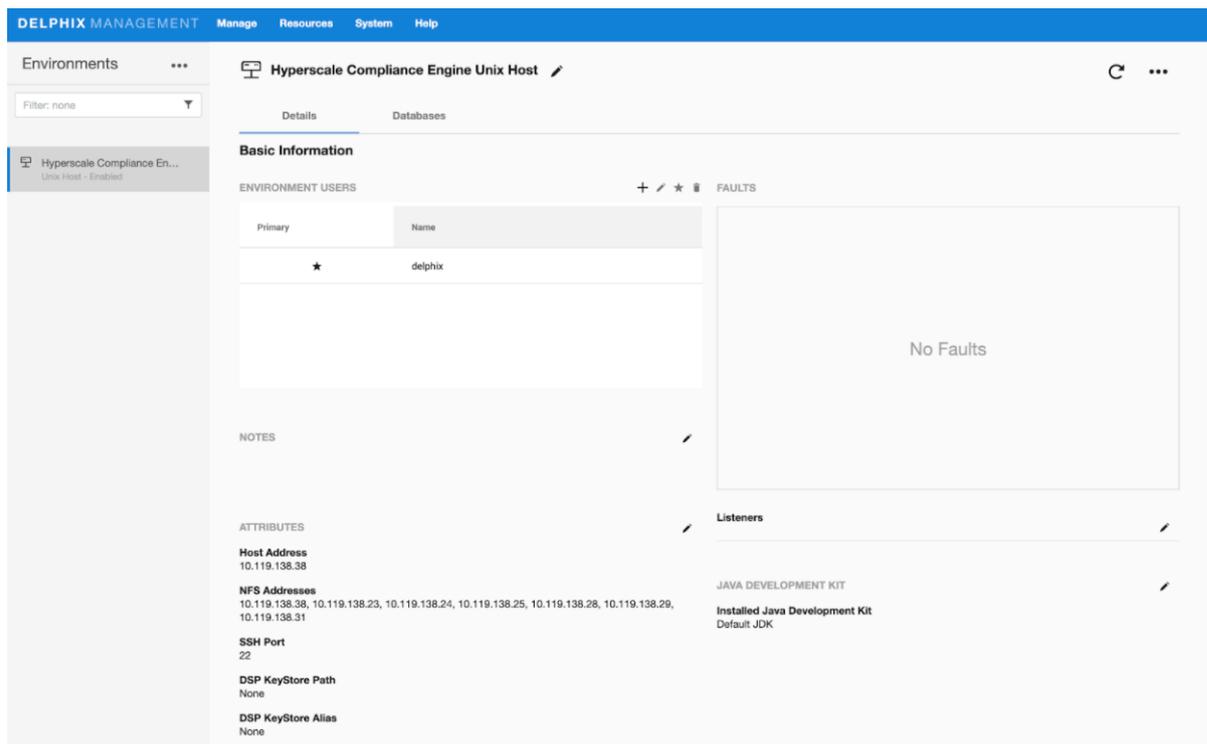
Provision an empty VDB

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 engine's 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 engine 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.

The location of the mounted empty VDB on the Hyperscale Compliance engine's 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 Engine 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 Engine host:

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

Storage requirements for the NFS file server

Considering a single Hyperscale Compliance job execution, the Hyperscale Compliance Engine 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.

Accessing the Hyperscale Compliance API

Open a web browser and type the following in the address bar: `https://<hyperscale-compliance-host-address>/hyperscale-compliance` . Replace `orch ip` with the IP address of the Hyperscale Compliance Engine VM.

Authentication

To authenticate with the Hyperscale Compliance Engine, you must use an API key. It is done by including the key in the HTTP Authorization request header with the type apk.

An example cURL command with the API Key looks like the following:

```
curl --header 'Authorization: apk
1.t8YTjLyPiMatdtnhAw9RD0gRVZr2hFsrfikp3YxVl8URdB9zuaVHcMuhXkLd1TLj'
```

As described in the HTTP Authorization request header documentation, the following is the typical syntax for the authorization header:

```
Authorization: <auth-scheme> <authorisation-parameters>
```

For Basic Authentication, You must include the following header parameters: `Authorization: Basic <credentials>`

For the Bearer Authentication scheme, you must 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. It 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. For more information about initial created API key, refer to step 8 under the Generate a New Key section.

Run the following command to create a new API key.

```
curl -X 'POST' \
   'https://<host-name>/api/v3.0.0/management/api-keys' \
   -H 'accept: application/json' \
   -H 'Authorization: apk
1.t8YTjLyPiMatdtnhAw9RD0gRVZr2hFsrfikp3YxVl8URdB9zuaVHcMuhXkLd1TLj' \
   -H 'Content-Type: application/json' \
   -d '{
   "name": "<name-of-key>"
}'
```

The above command displays a response message similar to the following:

> (i) Copy or save the newly created token from the response as this token value will not be accessible later.

```
{
  "api_key_id": 2,
  "token": "2.ExZtmf6EN1xvFMsXpXlOyhHVYlTuFzCm2yGhpUOQQ5ID8N8oGz79d4yn8ZsPhF46"
}
```

Since you have created a new and secure API key, you must delete the old key for security reasons.

Run the following command to delete the old key.

```
curl -X 'DELETE' \
   'https://<host-name>/api/v3.0.0/management/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 5 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'
```

How to setup a Hyperscale Compliance job

Pre-checks

You must check the following before starting a job:

- Storage space must be 2 times the size of the source data for NFS storage.
- 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 mount 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 Engine will impact the performance of Hyperscale Compliance jobs.
- Currently, the Hyperscale Compliance Engine 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 Engines. 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 Engine) = 40 GB approx`.

API flow to set up 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 a Mount Point
3. Create Connector Info
4. Create a Dataset
5. Create a Job
6. 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`.

> (i)   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
}
```

MountFileSystems API

POST /mount-filesystems (create a file mount)

Request:

```
{
"mountName": "staging_area",
"hostAddress": "de-6014-continuous-data.dlpxdc.co",
"mountPath": "/domain0/group-2/appdata_container-12/appdata_timeflow-13/datafile",
"mountType": "NFS4",
"options": "rw"
}
```

Response:

```
{
```

```
"id": 1,
"mountName": "staging_area",
"hostAddress": "de-6014-continuous-data.dlpxdc.co",
"mountPath": "/domain0/group-2/appdata_container-12/appdata_timeflow-13/datafile",
"mountType": "NFS4",
"options": "rw"
}
```

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"
}
}
```

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",
  }
}
```

> ⚠ 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 Engine's restartability feature to restart the job from the point of failure in the load/post-load step.

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,
"mount_filesystem_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"
    }
]
}
]
}
```

Response (with single table):

```
{
"id": 1,
"connector_id": 1,
"mount_filesystem_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"
        }
    ]
    }
    ]
    }
```

Request (with single table & filter_key in the source):

```
{
"connector_id": 1,
"mount_filesystem_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"
    }
```

```
      ]
    }
  ]
}
```

Response (with single table & filter_key in the source):

```
{
"id": 1,
"connector_id": 1,
"mount_filesystem_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"
    }
]
}
]
}
```

Request (with multiple tables):

```
{
"connector_id": 1,
"mount_filesystem_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"
}
]
},
{
"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,
"mount_filesystem_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"
}
]
},
{
"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"
}
]
}
]
}
```

ⓘ 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 <Link to API Doc>.

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",
  "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.
  a. **Set {"parallelism_degree" :0 }:** Use unmodified DDL provided by Oracle.
  b. **Set {"parallelism_degree": -1}:** Remove any parallel or nonparallel clause from the DDL.
  c. **Set {"parallelism_degree": any positive vale}:** Remove existing parallel degree or nonparallel clause and add Parallel <parallelism_degree> to the DDL.

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",
  "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
  }
}
```

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.

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

The new 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 engine 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 `mount_filesystem_id` (Required) and `data_info_settings` (Optional) for the source and target dataset. This `data_info_settings` will be applicable to all the data_info objects in the dataset. For more information, refer to the /import API page.

The following is an example of the `request blob`.

```json
{
  "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",
    "mount_filesystem_id": 1,
    "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.

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
}
```

> ⓘ 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 sync global settings from a Delphix Continuous Compliance Engine

The new 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 Engine 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
    ]
  }
}
```

ⓘ   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. Pre and post-script import from the Continuous Compliance Engine to Hyperscale Engine is not supported.
2. Import of Kerberos and Custom JDBC drivers connector-based making job is not supported.

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.
> 4. The dataset date format should be the same as the environment variable date format value.
> 5. In one dataset, all the inventories should use the same date format for all date formats.

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
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

Group	Property name	Type	Description	Default value
	`UNLOAD_FETCH_ROWS`	Number	Number of rows to be fetched from the database at a time.	10000
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	SQLLDR properties	20000

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
	`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-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

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
	`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
	`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
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_LEVE L_ORG_SPRINGF RAMEWORK_WEB_ FILTER\_COMMO NSREQUESTLOGG INGFILTER`	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
	`SPARK.DATE.T IMESTAMP.FORM AT`	String	Spark date and timestamp format for unload. This property is only applicable for MS SQL unload image	yyyy-MM-dd HH:mm:ss.S SS
	`SPARK.SMALL. DATE.TIMESTAM P.FORMAT`	String	Spark small date and timestamp format for unload. This property is only applicable for MS SQL unload image	yyyy-MM-dd HH:mm
	`UNLOAD.CONF. DISABLE.REPLI CATION`	Boolean	If set to true, disable database replication for source database. This property is only applicable for MS SQL unload image	true

Group	Property name	Type	Description	Default value
	`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
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`

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- %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
	`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
Load Service	`SQLLDR_SUCCESS_MESSAGE`	String	Message printed by sqlldr on successful loading of data.	'successfully loaded.'
	`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

Group	Property name	Type	Description	Default value
	`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`
	`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-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
	`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
	`SPARK.DATE.TIMESTAMP.FORMAT`	String	Spark date and timestamp format for load. This property is only applicable for MS SQL load image	yyyy-MM-dd HH:mm:ss.SSS

Group	Property name	Type	Description	Default value
	`SPARK.SMALL.DATE.TIMESTAMP.FORMAT`	String	Spark small date and timestamp format for load. This property is only applicable for MS SQL load image	yyyy-MM-dd HH:mm
	`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

⚠ spark.date.timestamp.format and spark.small.date.timestamp.format values should be the same for MS SQL load/unload services. Also if masking is applied on the date/timestamp datatype column, the applied inventory date format should be the same for MS SQL load/unload data format.

ⓘ
- 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 outsidethe container, respective log path has to be mounted by adding volume binding of that path in `docker-compose.yaml` for that service.

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

To view the API client documentation, a downloadable `.html` file is available below.

8_0_0_0_hyperscaleCompli…
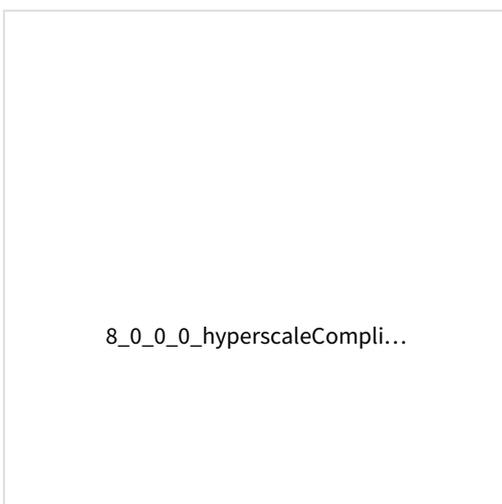
Cleaning up execution data

As part of the Hyperscale execution run, 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. Additionally, unload service, masking service, and load service will also store transient internal data for the execution while running it. This data is also not required once execution is completed and should be cleaned. Following are the three ways this data will be/can be cleaned.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
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.