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

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 3.0.0.1

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

Release 3.0.0

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

Known issues

This section describes the known issues in Hyperscale Compliance.

Release 3.0.0.1

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

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-935	Load service fails when source DB contains BLOB type data that is not simple text file data	None
HM-1561	Oracle Load Failure: sql loader control files doesn't contain character length when column size is less than 256 CHAR	None
HM-1705	Improper error message in Hyperscale status response if CCE gets mount file system connection error	None

Release 3.0.0

Key	Summary	Workaround
HM-177	Able to POST /hyperscale-masking/jobs with min job memory > max job memory	Change the max job memory value to higher than min job memory in API request.
HM-291	Hyperscale job execution with intelligent load balancer configured is stuck in a loop if job's max memory is more than totalAllocatedMemoryForJobs	Change the max memory to a value under the value of `totalAllocatedMemoryForJobs` property configured on Continuous Compliance Engine.
HM-652	Job execution is stuck in running state if mount server is powered off	Check the health of mount server before starting a job.
HM-663	Load process is failing with "Error disabling constraint" for identity columns	None

Key	Summary	Workaround
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-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

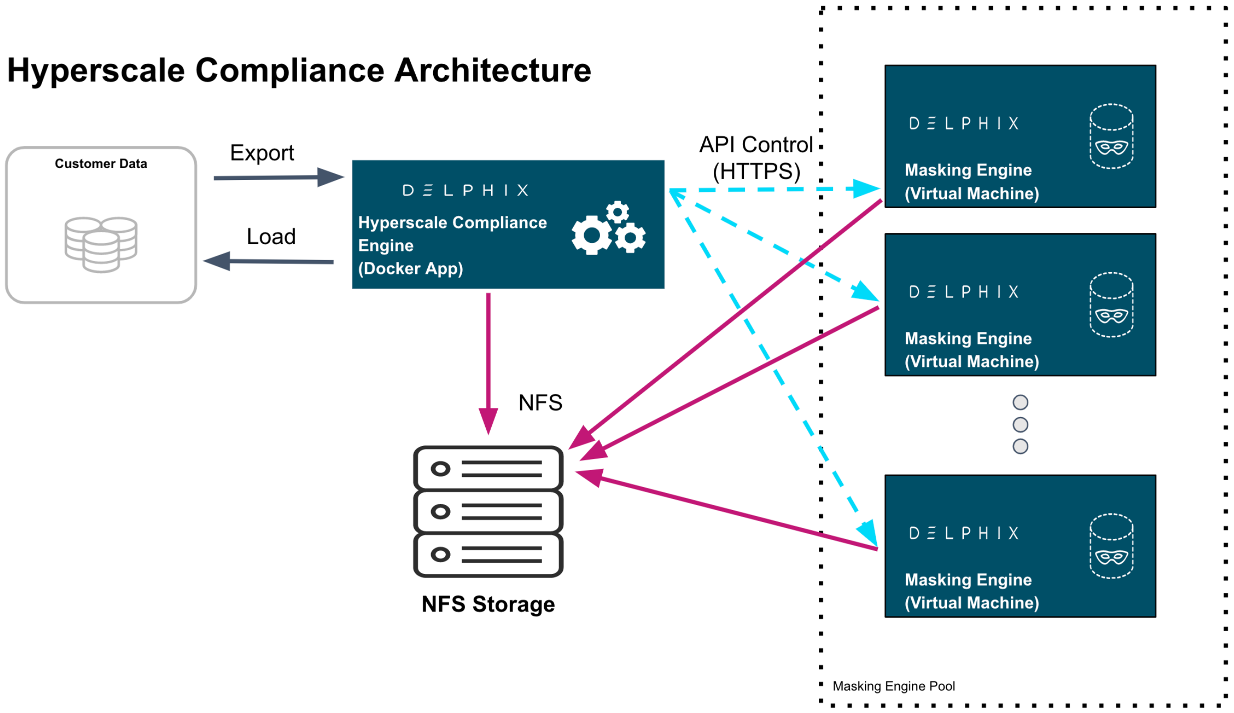
Key	Summary	Workaround
HM-935	Load service fails when source DB contains BLOB type data that is not simple text file data	None
HM-1561	Oracle Load Failure: sql loader control files doesn't contain character length when column size is less than 256 CHAR	None
HM-1705	Improper error message in Hyperscale 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 1 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 delimited files. The Hyperscale Compliance engine then splits the exported data into multiple modules of smaller chunks and configures the masking jobs of all the respective chunks across multiple Continuous Compliance Engines. Upon successful completion of the masking jobs, the masked data is imported back into the database.


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

Hyperscale Compliance components

The Hyperscale Compliance architecture consists of four components mainly; the Hyperscale Compliance 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 constraint) 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.
- **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, 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 sources are configured to be the same data source, a best practice is to restore the single data source from a backup after a failure since the source data source may be in an inconsistent state (rather than only the target data source).

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 Installing Hyperscale Compliance tar.
- Read about the Network Requirements of the Hyperscale Compliance.
- Read about Accessing the Hyperscale Compliance APIs.

Getting started

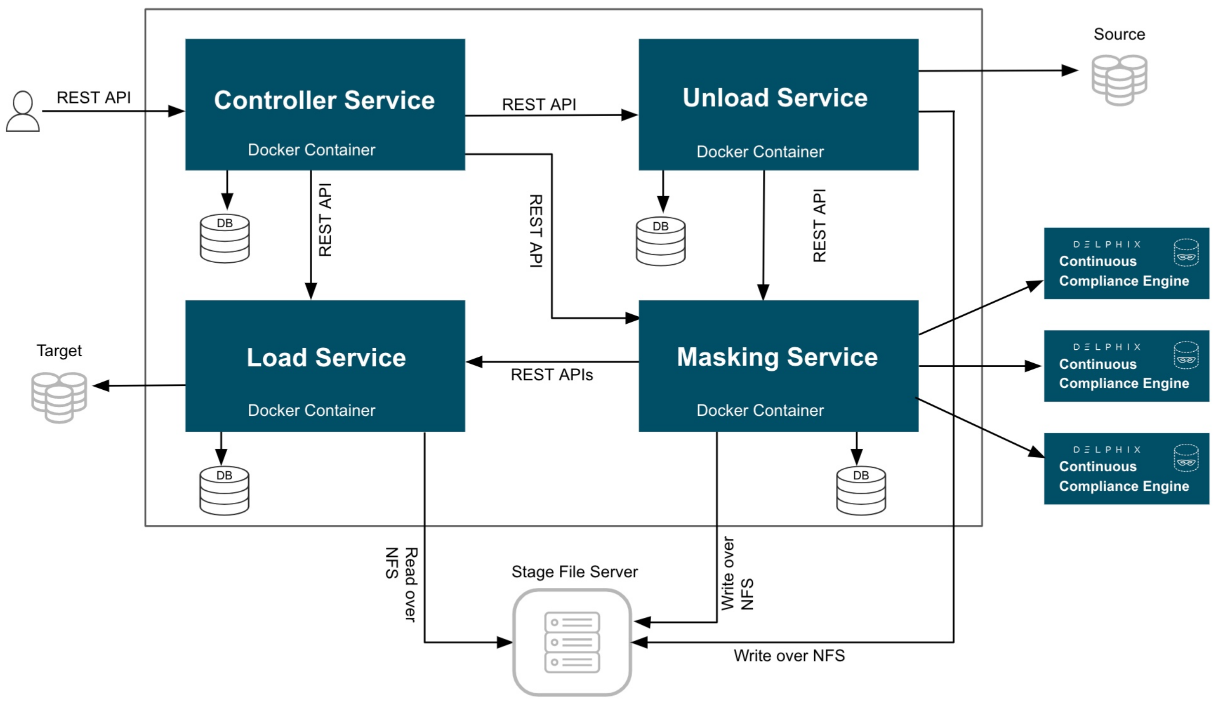
This section covers the following topics:

- Hyperscale Compliance architecture
- Data source support
- Supported platforms
- Network requirements
- Host requirements (Docker Compose)
- Installation
- 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 a 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:
    - Split the CSV file based on the split size.
    - Based on Intelligent load balancing, create and start jobs for splitted 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 DB), 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)

Supported platforms

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

Supported Continuous Compliance Engine

- 6.0.14.0

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

Supported Continuous Data Engine

- 6.0.14.0

Supported Browsers (Only API client)

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

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.	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.
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

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-3.0.0.0.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`
- 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. A group ownership id of 50 and a permission mode of 750 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/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-3.0.0.0.tar.gz`
2. Load the extracted tars into Docker.

```
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
```

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:
    a. 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 a group ownership id of 50. As such, the ‘staging_area’ directory, along with the directory(‘hyperscale’) one level above, require a group ownership id of 50 and a permission mode of 770 so that the Hyperscale Compliance Engine and the Continuous Compliance Engine(s) can perform read/write/execute operations on the staging area.

   b. 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) must be applied again on the staging area directory. **Note:** 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:
   a. For each of the docker containers, except the ‘proxy’ container, add a volume entry binding the staging area path (from 3(a), `/mnt/hyperscale`) to the Hyperscale Compliance Engine container path(`/etc/hyperscale`) as a volume binding under the ‘volumes’ section.
   b. For the **load-service** docker container, add a volume entry that binds the path of the ‘Oracle instant Client’ on the host to the path on the container(`/usr/lib/instantclient`) under the ‘volumes’ section.
   c. [Optional] Step 6b explains how the logs of a given container can be viewed with docker commands. If you would like to redirect the logs of one or more containers to a particular directory, then you have the option to do the same by setting up a logging directory and exposing the same, as a volume binding, in the `docker-compose.yaml` file. This directory again must have a group ownership id of 50 and a permission mode of 770, due to the same reasons as highlighted in step 3a, so that the Hyperscale Compliance Engine can perform read/write/execute operations in the logging directory. The following example includes volume bindings to redirect the docker container logs of each service to separate directories. An example `docker-compose.yaml` file looks like the following:

```
networks:
  hyperscale-net: {}
services:
  controller-service:
    depends_on:
      load-service:
        condition: service_started
      masking-service:
        condition: service_started
      unload-service:
        condition: service_started
    environment:
      API_KEY_CREATE: "true"
    healthcheck:
      interval: 30s
      retries: 3
      start_period: 30s
      test: curl --fail --silent http://localhost:8080/actuator/health |
grep UP || exit 1
```

```
      timeout: 25s
    image: delphix-controller-service-app:3.0.0.0
    init: true
    networks:
      hyperscale-net: null
    restart: unless-stopped
    volumes:
      - hyperscale-controller-data:/data:rw
      - /home/hyperscale_user/logs/controller_service:/opt/delphix/logs
  load-service:
    image: delphix-load-service-app:3.0.0.0
    init: true
    networks:
      hyperscale-net: null
    restart: unless-stopped
    volumes:
      - hyperscale-load-data:/data:rw
      - /mnt/hyperscale:/etc/hyperscale
      - /opt/oracle/instantclient:/usr/lib/instantclient
      - /home/hyperscale_user/logs/load_service:/opt/delphix/logs
  masking-service:
    image: delphix-masking-service-app:3.0.0.0
    init: true
    networks:
      hyperscale-net: null
    restart: unless-stopped
    volumes:
      - hyperscale-masking-data:/data:rw
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/masking_service:/opt/delphix/logs
  proxy:
    depends_on:
      controller-service:
        condition: service_started
    image: delphix-hyperscale-masking-proxy:3.0.0.0
    init: true
    networks:
      hyperscale-net: null
    ports:
      - published: 443
        target: 443
    restart: unless-stopped
  unload-service:
    image: delphix-unload-service-app:3.0.0.0
    init: true
    networks:
      hyperscale-net: null
    restart: unless-stopped
    volumes:
      - hyperscale-unload-data:/data:rw
      - /mnt/hyperscale:/etc/hyperscale
      - /home/hyperscale_user/logs/unload_service:/opt/delphix/logs
```

```
version: '3.7'
volumes:
  hyperscale-controller-data: {}
  hyperscale-load-data: {}
  hyperscale-masking-data: {}
  hyperscale-unload-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`
   a. Run the following command to check if the application is running. The output of this command should show five containers up and running. `docker-compose ps`
   b. Run the following command to access the application logs of a given container. `docker logs -f <service_container_name>` . **Note:** The service container name can be accessed by the output of the command `docker-compose ps` .
   c. The service container name can be accessed by the output of the command `docker-compose ps` .
   d. 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.
   `Docker logs -f <service_container_name>`

ⓘ 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:
1.89lPH1dHSJQwHuQvzawD99sf4SpBPXJADUmJS8v00VCF4V7rjtRFAftGWygFfsqM
```

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.

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 Continuous Data Engine empty VDB

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

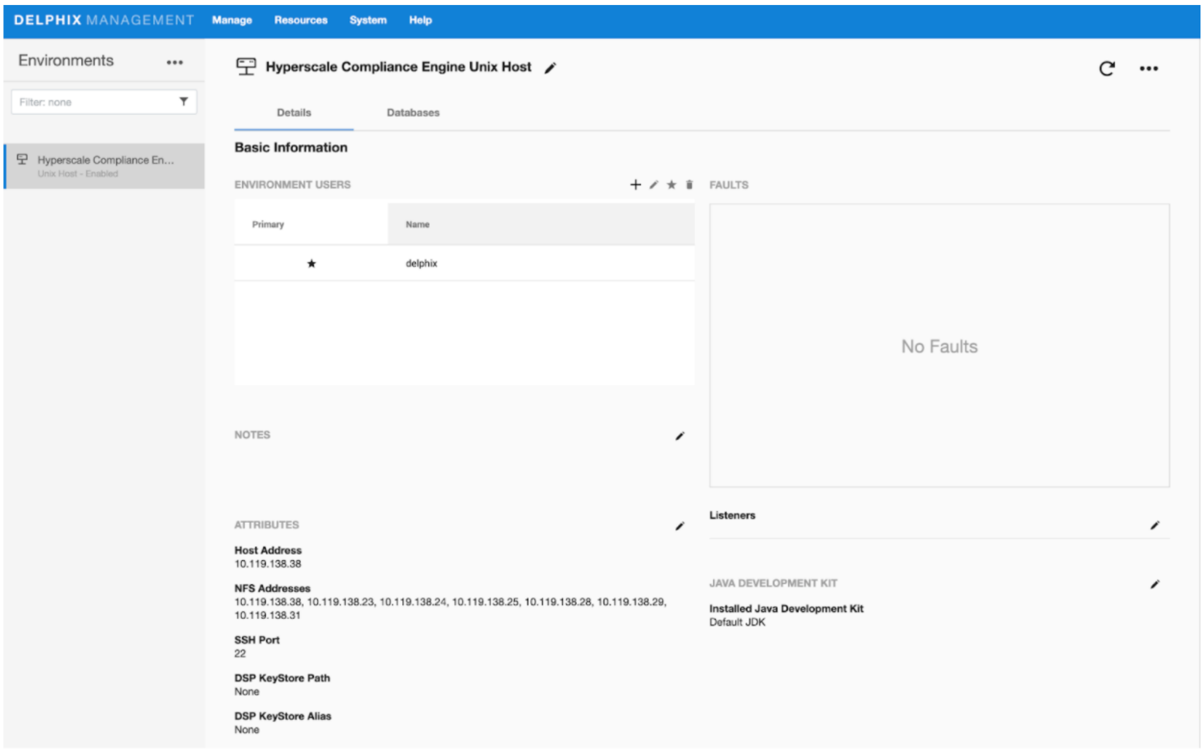
Continuous Data Engine installation

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

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

Discover and configure Hyperscale Compliance 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.



Hyperscale Compliance Engine Unix Host

Basic Information

ENVIRONMENT USERS

Primary	Name
★	delphix

FAULTS

No Faults

NOTES

ATTRIBUTES

Host Address
10.119.138.38

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

SSH Port
22

DSP KeyStore Path
None

DSP KeyStore Alias
None

Listeners

JAVA DEVELOPMENT KIT

Installed Java Development Kit
Default JDK

Provision an empty VDB

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

```
{
  "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 3 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 (after VDB creation) on an empty VDB-mounted folder on the Hyperscale VM.Note
  Note: The permission that is granted before VDB creation will not work. It happens so because Continuous Data Engine removes the write permission from VDB mounted folder after VDB creation.
- Based on the umask value for the user that is used to mount, the permissions for the staging area directory could get altered after the NFS share has been mounted. In such cases, you must re-apply the permissions (i.e. 770) on the staging area directory.
- You must restart the `containers/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.
- If you want to redirect the logs of one or more containers to a particular directory, then you have the option to do the same by setting up a logging directory and exposing the same, as a volume binding, in the `docker-compose.yaml` file. This directory again must have a group ownership id of 50 and a permission mode of 770 as below:
  volumes:
    - `hyperscale-controller-data:/data:rw`
    - `/mnt/hyperscale:/etc/hyperscale`
    - `/home/hyperscale_user/logs/controller_service:/opt/delphix/logs`
- 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 Setup a Hyperscale Compliance Job

The following is the API flow for setting up and executing a Hyperscale Compliance job.

1. Register Continuous Compliance Engine(s)
2. Create 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 url `https://<hyperscale-compliance-host-address>/hyperscale-compliance`.

APIs must be called only in the below order.

Engines API

POST /engines (Register an engine):
Request:

```
{
"name": "Delphix Continuous Compliance Engine 6.0.14.0 on AWS",
"type": "MASKING",
"protocol": "http",
"hostname": "de-6014-continuous-compliance.delphix.com",
"username": "hyperscale_compliance_user",
"password": "password123"
}
```

Response:

```
{
"id": 1,
"name": "Delphix Continuous Compliance Engine 6.0.14.0 on AWS",
"type": "MASKING",
"protocol": "http",
"hostname": "de-6014-continuous-compliance.delphix.com",
"username": "hyperscale_compliance_user",
"ssl": true,
"ssl_hostname_check": true
}
```

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)
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"
}
}
```

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"
}
}
```

⚠ A failure in the load or pre/post load steps (disabling/enabling constraints, triggers, etc.) may leave the target database in an inconsistent state since the load step truncates the target tables when it begins. If the source and target connectors are configured to be the same database/tables, a best practice is to restore the single database from a backup after a failure since the source database may be in an inconsistent state (rather than only the target database).

DataSets API

POST /data-sets (create dataset for Hyperscale Compliance)
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 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 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 Hyperscale Compliance API Documentation page available in API Reference section of this Documentation.

Jobs API

POST /jobs (create a Hyperscale Compliance job)

```
Request:
{
"name": "job_1",
"masking_engine_ids": [
1,2,3
],
"data_set_id": 1,
"app_name_prefix": "app_1",
"env_name_prefix": "env_1",
"retain_execution_data": “NO”,
"masking_job_config": {
"max_memory": 2048,
"min_memory": 1024,
"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.

Response:

```
{
"id": 1,
"name": "job_1",
"masking_engine_ids": [
1,
2,
        3
],
"data_set_id": 1,
"app_name_prefix": "app_1",
"env_name_prefix": "env_1",
"retain_execution_data": “NO”,
"masking_job_config": {
"feedback_size": "100000",
"min_memory": "1024",
"description": "Job created by Hyperscale Masking",
"stream_row_limit": "10000",
"max_memory": "2048",
"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": 1,
"job_id": 1,
"status": "RUNNING",
"create_time": "2022-06-14T12:46:54.139452",
"tasks": [
{
  "name": "Unload"
},
{
  "name": "Masking"
},
{
  "name": "Load"
},
{
  "name": "Post Load"
}
]
}
```

GET /executions/{execution_id} (returns the job execution by execution_id)
Request:

```
iD: 1
```

Response:

```
{
"id": 1,
"job_id": 1,
"status": "SUCCEEDED",
"create_time": "2022-06-10T11:58:39.385186",
"end_time": "2022-06-10T11:59:26.030750",
"tasks": [
{
  "name": "Unload",
  "status": "SUCCEEDED",
  "start_time": "2022-06-10T11:58:39.401906",
  "end_time": "2022-06-10T11:58:46.042788",
  "metadata": [
    {
      "source_key": "SCHEMA_1_TARGET.TABLE_1_TARGET",
      "unloaded_rows": 5,
      "total_rows": 5
    }
  ]
},
{
  "name": "Masking",
  "status": "SUCCEEDED",
  "start_time": "2022-06-10T11:58:39.666638",
  "end_time": "2022-06-10T11:59:16.034657",
  "metadata": [
    {
      "source_key": "SCHEMA_1_TARGET.TABLE_1_TARGET",
      "masked_rows": 5,
      "total_rows": 5
    }
  ]
},
{
  "name": "Load",
  "status": "SUCCEEDED",
  "start_time": "2022-06-10T11:59:07.236429",
  "end_time": "2022-06-10T11:59:16.064497",
  "metadata": [
    {
      "source_key": "SCHEMA_1_TARGET.TABLE_1_TARGET",
      "loaded_rows": 5,
      "total_rows": 5
    }
  ]
},
{
  "name": "Post Load",
  "status": "SUCCEEDED",
  "start_time": "2022-06-10T11:59:16.072760",
  "end_time": "2022-06-10T11:59:16.072760"
}
]
}
```

- 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)
- Below API can be used only for manually cleaning up the execution: `DELETE /executions/{execution_id}` (Clean-up the execution)

Configuration settings

Possible values of Configuration Settings having Type “Log Level” are TRACE, DEBUG, INFO, WARN, ERROR, FATAL, or OFF.

The following table lists the Hyperscale Compliance properties with their default values.

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 bootstrap key is available.	true
	load.service.requirePostLoad	Boolean	Set if the Post Load step needs to be executed.	true
	source.key.field.names	String	Dataset configuration. These fields/columns are used to uniquely identify source data.	schema_name,table_name
	execution.status.poll.duration	Milli-seconds	Time duration in which execution status is collected from different services	120000
	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

Group	Property name	Type	Description	Default value
	logging.level.com.delphix.hyperscale	Log 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
	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`
	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.rolling-file-name[1]	String	Archived file location & name	`/opt/delphix/logs/archived/hyperscale-controller-%d{yyyy-MM-dd}.%i.log`

Group	Property name	Type	Description	Default value
	logging.file.max-size	File Size (String)	Max individual file size	5MB
	logging.file.max-history	Number of Days	History in days (i.e. keep 15 days' worth of history capped at 5GB total size)	15
	logging.file.total-size-cap	File Size (String)	Max limit the combined size of log archives	5GB
	logging.level.org.springframework.web.filter.CommonsRequestLoggingFilter	Log Level	This configuration controls the logging information of the HTTP requests received by Hyperscale. This is by default set to DEBUG level for logging request URIs of the Incoming Requests.	DEBUG
	api.version.compatibility.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.	true
	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	unload.fetch.rows	Number	Number of rows to be fetched from the database at a time.	10000

Group	Property name	Type	Description	Default value
	logging.level.root	Log Level	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.hyperscale	Log 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
	logging.file.name[1]	String	Log file location & name	`/opt/delphix/logs/hyperscale-unload.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.rolling-file-name[1]	String	Archived file location & name	`/opt/delphix/logs/archived/hyperscale-unload-%d{yyyy-MM-dd}.%i.log`
	logging.file.max-size	File Size in String	Max individual file size	5MB
	logging.file.max-history	Number of Days	History in days (i.e. keep 15 days' worth of history capped at 5GB total size)	15
	logging.file.total-size-cap	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
Masking Service	logging.level.root	Log Level	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.hyperscale	Log 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
	logging.file.name 1	String	Log file location & name	`/opt/delphix/logs/hyperscale.log`
	logging.pattern.file	String	Logging pattern for file	`%d{dd-MM-yyyy HH:mm:ss.SSS} \[%thread\] %-5level %logger{36}.%M - %msg%n`
	logging.pattern.console	String	Logging pattern for console	`%d{dd-MM-yyyy HH:mm:ss.SSS} \[%thread\] %-5level %logger{36}.%M - %msg%n`
	logging.pattern.rolling-file-name1	String	Archived file location & name	`/opt/delphix/logs/archived/hyperscale-%d{yyyy-MM-dd}.%i.log`
	logging.file.max-size	File Size in String	Max individual file size	5MB
	logging.file.max-history	Number of Days	History in days (i.e. keep 15 days' worth of history capped at 5GB total size)	15
	logging.file.total-size-cap	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
	remove.splitted.masked.files	Boolean	Flag to indicate retaining or removing staging splitted masked files. This is only applicable when merging of files is done post Masking and before Loading into the target data source.	true
	monitor.initial.delay	Time(in seconds)	Time to delay the first monitoring call to Masking Engines. This is related to Monitoring of Masking Engine Jobs.	1
	monitor.poll.duration	Time(in seconds)	Polling period between subsequent monitoring calls to Masking Engines. This is related to Monitoring of Masking Engine Jobs.	10
	scheduler.poll.duration	Time(in seconds)	Rate at which scheduler polls for available data to schedule load balancing. Configured in seconds.	60
	intelligent.loadbalance.enabled	Boolean	Set this to false if need to enable round robin load balancing in place of intelligent load balancing.	true
	job.monitor.pool.corePoolSize	Number	Job monitor thread pool related configuration Number of core threads. This is related to Monitoring of Masking Engine Jobs.	5

Group	Property name	Type	Description	Default value
	job.monitor.pool.maxPoolSize	Number	Job monitor thread pool related configuration Maximum number of threads in thread pool. This is related to Monitoring of Masking Engine Jobs.	20
	job.monitor.pool.keepAliveSeconds	Time in seconds	Job monitor thread pool related configuration Maximum idle time of threads. This is related to Monitoring of Masking Engine Jobs.	300
	job.monitor.pool.queueCapacity	Number	Job monitor thread pool related configuration Maximum queue capacity. This is related to Monitoring ofMasking Engine Jobs.	50
Load Service	sqlldr-success-message	String	Message printed by sqlldr on successful loading of data.	‘successfully loaded.’
	logging.level.root	Log Level	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	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.level.com.delphix.hyperscale	Log 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
	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`
	logging.pattern.rolling-file-name[1]	String	Archived file location & name	`/opt/delphix/logs/archived/hyperscale-load-%d{yyyy-MM-dd}.%i.log`

Group	Property name	Type	Description	Default value
	logging.file.max-size	File Size in String	Max individual file size	5MB
	logging.file.max-history	Number of Days	History in days (i.e. keep 15 days' worth of history capped at 5GB total size)	15
	logging.file.total-size-cap	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
	sqlldr.blob.clob.char.length	Number	SQLLDR properties	20000

- 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, refer to the static HTML representations here:

3_0_0_hyperscaleComplian…

How to generate a support bundle

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 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 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
```

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 is used to generate 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 located at /etc/hyperscale/hyperscale-support-****.tar.gz inside the container. This means tar file is generated under path which 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 docker command in order to generate the support bundle.

Cleaning up execution data

As part of the Hyperscale execution run, the system will create data files (unload service), split files and masked files (masking service) on the file server. As the data size can be large (3 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.

1. Using retain_execution_data

While setting up a Hyperscale Job (POST /jobs), you can set the value for `retain_execution_data` property to the intimate system when it should clean up data automatically based on the table

EXECUTION_STATUS	RETAIN_EXECUTION_DATA	CLEAN UP AUTOMATICALLY?
NA(SUCCESS/FAILED)	NO	YES
SUCCESS	ON_ERROR	YES
FAILED	ON_ERROR	NO
NA(SUCCESS/FAILED)	ALWAYS	NO

2. Manual Clean Up

Hyperscale exposes a delete API (DELETE /executions/{id}) to manually clean up data for an 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.