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.

Overview

Delphix Hyperscale Masking is designed to improve the performance of masking large datasets. This feature functions by breaking datasets into smaller pieces, then orchestrating the work of masking those pieces across many Masking Engines. In general, datasets larger than 10 TB in size will see improved masking performance when run on the Hyperscale architecture.

Hyperscale Masking

Introduction

There are three main components to the Hyperscale architecture: the Orchestrator, source/target Connectors, and the Masking Engine Cluster.

Orchestrator

The Orchestrator is responsible for taking a series of files (provided by a Source Connector) and initiating a masking job, in parallel, across nodes in the Masking Engine Cluster. The total throughput of an individual job can be adjusted up or down depending on the number of nodes in the Cluster.

For installation, the Orchestrator is a Linux binary that must be executed on a host running Red Hat CentOS version 7.x - 8.x, or higher.

Orchestrator host requirement on Linux

The subdirectory of **/u01/** will be used throughout the article as a root folder where configuration files will be placed.

Type	**Host Requirement**	**Explanation**
User	There must be an operating system user (orch_os) able to login to the orchestrator host via SSH (port 22). These permissions must be granted as well, and can be done via sudo authorization of the commands: - Permission to **run mount**, **unmount**, **mkdir**, **rmdir**, and **ping** as a super-user with NOPASSWD.	This will be a primary user responsible for running orchestrator executables.
Executable Folder	There must be a directory on the Orchestrator host where the executable can be installed. To install it, simply copy the binary file to the directory and set it to be executable.	The directory must have `-rwxrwx--- (0770)` permissions at minimum. The orch_os user must own the directory and have `-rwxrwx--- (0770)` permissions on each directory in the path leading to the toolkit directory. At least 5 GB of storage is needed at the time of setting up the environment and at least 500MB of free space is required to allow refreshes and maintenance of the toolkit, especially during upgrades.

Type	Host Requirement	Explanation
Apps Folder	Mask application folder, for example: */u01/mask-apps*. /u01/mask-apps/**app-name**	The orch_os user must own the directory. This folder will host configuration files as discussed in the following sections.
Logging	Logs and other runtime temporary files folder, for example: */u01/mask-logs*. /u01/mask-logs/**app-name/timestamp.log**	The orch_os user must own the directory. This folder will host runtime/log files.
Network Connectivity	The Orchestrator must be able to communicate with each node of the Masking Engine Cluster on port 443 or 80 (not recommended). A user can initiate the following tests from the *orch_host_checker.sh* script to check: - OS and Host permissions/access - Network Port Checks	The orch_os user must have sufficient permissions to execute basic connectivity commands on the OS (ping, ip address, traceroute, etc.).
AWS SDK	If source data is in S3 and/or Redshift, then AWS SDK must be installed on the Orchestrator host. AWS access details are also required.	This is optional if the source and destination is on AWS S3 or Redshift.
Hardware Requirements	Minimum: 32 vCPU, 512 GB of memory, 1 TB data disk. Recommended: 64 vCPU, 1 TB of memory, 1 TB data disk.	OS disk space: 20 - 30 GB

Configuration

The configuration is divided into three (3) components/types, as follows:

1. **Masking Engine Cluster** file (engines.json)
2. **Source and Target Connectors** config file (tables_info.json)
3. **Individual Table Inventory** config file (table_name.json)

The list of nodes the Orchestrator will communicate within the Masking Engine Cluster must be defined in the file (e.g. */u01/mask-apps/engines.json*). An example of the syntax to use when defining the cluster nodes in the config file is shown below.

engines.json file

```
{
 "engines": [
   {
     "name": "exec1",
     "IP": "30.0.0.240",
     "protocol": "https",
     "username": "admin",
     "password": "gAAAAABhXockEdXK3GNDr=="
   },
   {
     "name": "exec2",
     "IP": "30.0.0.181",
     "protocol": "https",
     "username": "admin",
     "password": "gAAAAABhXockEdXK6ycDrxJu"
   }
 ],
```

- **Engines (required):** Declares how many masking nodes in the cluster would be used to mask. This example has two (exec1 and exec2).
- **Name:** Determines node identity on execution.
- **IP:** Engine IP address.
- **Username/Password:** The Orchestrator will encrypt any plaintext password.

```
"virt_engines": [
   {
     "engName": "virt1",
     "IP": "10.0.1.100",
     "protocol": "https",
     "username": "admin",
     "password": "gAAAAABhXocklOVOt35DRnc"
   }
 ],
```

- **Virt_Engines (optional):** The Orchestrator can also leverage vFiles for the staging area.
- **EngName:** Name of the Virtualization Engine.

```
"orch_config": [
   {
     "type": "sftp",
     "name": "orch1",
     "IP": "30.0.0.224",
     "hostname": "ip-30-0-0-224",
     "username": "orch_os",
     "password": "gAAAAABhXockU6mvBSPRzH0",
     "sftp_port": 22
   }
 ]
}
```

* **Orch_Config (required):** Orchestrator configures whether to use SFTP (on port 22/SSH) or NFS mount-to-read, and mask files in the staging area. * **Type:** Property accepts either `nfs_mount` or `sftp`.

Source & Target Connectors

Source Connectors are responsible for unloading data from the system of record (SOR) into a series of files located in the Staging Area. These Connectors leverage bulk unload operations offered by the SOR and are therefore unique to each. The prerequisites for running each Connector vary slightly but always require network access to the SOR from the host running the Connector and credentials to run the appropriate unload commands.

The Connectors are installed in the same high-level directory as the Orchestrator. Connectors can be defined in a simple JSON file (similar to the files above) that takes a series of inputs, as described below.

tables_info.json config file

```
{
 "source": {
   "db_type": "oracle",
   "hostname": "ora-source",
   "dbname": "prod_data",
   "username": "user",
   "password": "gAAAAABhXpRmlF",
   "dbport": "1521",
   "jdbc": "Y"
 },
 "target": {
   "db_type": "oracle",
   "hostname": "ora-target",
   "dbname": "dev_data",
   "username": "user",
   "password": "gAAAAABhXffRml",
   "dbport": "1521",
   "jdbc": "Y"
 },
```

- **Source/Target (required):** Defines what source to read data from.
- **DB_Type:** Determines the database system: Oracle, MSSQL, AWS S3, AWS Redshift, NFS, IBM Db2, MongoDB.
- **Hostname:** Hostname or IP of the host.

- **Username/Password:** The Orchestrator will encrypt any plaintext password.
- **Dbport:** Database port number.
- **JDBC:** If the source is connected via JDBC (Y/N).

```
"tables": [
{
 "source_schema": "DELPHIXDB",
 "source_table": "CUSTOMER_EMPLOYMENT_DATA_03",
 "target_schema": "DELPHIXDB",
 "target_table": "CUSTOMER_EMPLOYMENT_DATA_03",
 "filterKey": "N/A",
 "unloadSplit": "1",
 "unloadCols": "*",
 "splitSizeMB": "1000"
},
{
 "source_schema": "DELPHIXDB",
 "source_table": "CUSTOMER_EMPLOYMENT_DATA_02",
 "target_schema": "DELPHIXDB",
 "target_table": "CUSTOMER_EMPLOYMENT_DATA_02",
 "filterKey": "N/A",
 "unloadSplit": "1",
 "unloadCols": "*",
 "splitSizeMB": "1000"
}
],
```

- **Tables:** List of tables to mask.
- **Source_x:** Table schema, table name.
- **Target_x:** Table schema, table name.
  All tables are listed here.

```
],
 "files": [
   {
"format": "parquet",
"source_file": "customers.parquet",
"file_size": 10277,
"source_path": "/mnt/provision/par_dump",
"target_file": "customers.msk_parquet",
"target_path": "/mnt/provision/par_trgt",
"splitSizeMB": "1000"
   }
 ]
}
```

- **Files:** List of files to mask. For example, Parquet files either on S3 or local NFS mount point.
- **Format:** Type of file: Parquet, JSON, CSV, fixed width, etc.

table_name.json config file

```json
{
 "_pageInfo": {
   "numberOnPage": 14,
   "total": 14
 },
 "responseList": [
   {
     "fileFieldMetadataId": 15909,
     "fileFormatId": 1204,
     "recordTypeId": 1204,
     "fieldLength": 0,
     "fieldName": "customerId",
     "fieldPositionNumber": 1,
     "isMasked": false,
     "isProfilerWritable": true
   },
   {
     "fileFieldMetadataId": 15910,
     "fileFormatId": 1204,
     "recordTypeId": 1204,
     "fieldLength": 0,
     "fieldName": "firstName",
     "fieldPositionNumber": 2,
     "algorithmName": "FirstNameLookup",
     "domainName": "FIRST_NAME",
     "dateFormat": "",
     "isMasked": true,
     "isProfilerWritable": true,
     "notes": ""
   }
 }
}
```

- This file will be generated as part of the Orchestrator execution process for each table and field that requires masking.
- The Orchestrator includes a feature allowing the user to easily generate the configuration using a Delphix Masking engine-sensitive data inventory.

Staging area

The Staging Area is where data from the SOR is written to a series of files by the Source Connector. It can be either an object store such as AWS S3 or a file system. The file system can be attached volumes, or it can be supplied via the Delphix Virtualization Engine vFile feature. In either case, there must be enough storage available to hold the dataset in an uncompressed format.

Orchestrator

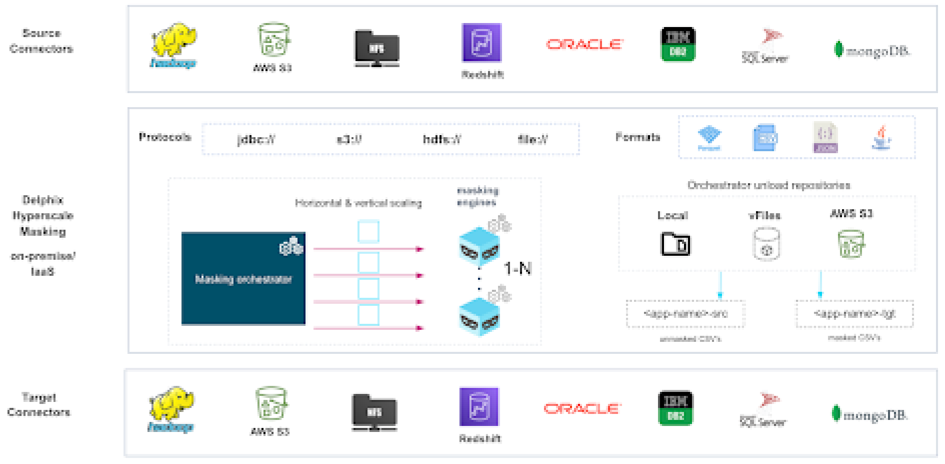
The Orchestrator binary takes a series of the following command arguments:

Command usage

```
dxhs-mask -a ora_customer -e mask_ora -u virt --virt “virt1:mask_ora:/mnt/
provision:Linux Orchestrator:HyperScale Mounts” -m “exec3" -i /u01/mask-apps -o /u01/
mask-logs -d ‘|' -en Y -t ‘\n
-a: application name (ora_customer)
-e: environment name (mask_ora)
-u: unload repository (aws/orch/virt)
    orch means use local nfs mount point for staging
--virt:mnt_name:<mnt_path>:<virtualization env-name>:<dataset-group>
-op: mnt_name:<mnt_path> then --virt argument not required
-m: masking engine name
-i: orchestrator mask folder for config files
-o: orchestrator log folder also for metadata other related files
-d: delimiter to be used for csv files
-en: if config files are encrypted (Y = yes or E = to encrypt)
-t: new line terminator
```

Masking Engine cluster

The Masking Engine Cluster is a group of Delphix Masking Engines v6.0.7+ that will be leveraged by the Orchestrator to run large masking jobs in parallel. Instructions for installing and configuring each Engine can be found in the Masking Documentation.


Source Connectors
AWS S3
NFS
Redshift
ORACLE
IBM DB2
SQL Server
mongoDB
Delphix Hyperscale Masking
on-premise/ IaaS
Protocols
jdbc://
s3://
hdfs://
file://
Formats
Parquet
JSON
Horizontal & vertical scaling
masking engines
Masking orchestrator
1-N
Orchestrator unload repositories
Local
vFiles
AWS S3
<app-name>-src
unmasked CSV's
<app-name>-tgt
masked CSV's
Target Connectors
AWS S3
NFS
Redshift
ORACLE
IBM DB2
SQL Server
mongoDB