Data Control Tower Home

Data Control Tower

Exported on 11/26/2024

Table of Contents

1 Welcome to the Data Control Tower documentation!

This information explains how to deploy Data Control Tower (DCT), use its features, or tune its configurations for optimal performance. The content has been organized into several categories, available from the lefthand navigation.

List of Data Control Tower documentation versions in PDF format.

- DCT 20.0.0 documentation.pdf[1]
- DCT 19.0.0 documentation.pdf[2]
- DCT 18.0.0 documentation.pdf[3]
- DCT 17.0.0 documentation.pdf[4]
- DCT 16.0.0 documentation.pdf[5]
- DCT 15.0.0 documentation.pdf[6]
- DCT 14.0.0 documentation.pdf[7]
- DCT 13.0.0 documentation.pdf[8]
- DCT 12.0.0 documentation.pdf[9]
- DCT 11.0.0 documentation.pdf[10]
- DCT 10.0.0 documentation.pdf[11]
- DCT 9.0.0 documentation.pdf[12]

---

1 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2020.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404896327&version=1
2 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2019.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404896592&version=1
3 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2018.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404896822&version=1
4 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2017.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404897059&version=1
5 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2016.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404910593&version=1
6 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2015.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404909752&version=1
7 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2014.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404908659&version=1
8 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2013.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404907366&version=1
9 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2012.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404906493&version=1
10 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2011.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404905435&version=1
11 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%2010.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404904415&version=1
12 https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%209.0.0%20documentation.pdf?
  api=v2&cacheVersion=1&modificationDate=1728404903343&version=1

- [DCT 8.0.0 documentation.pdf](#)[13]
- [DCT 7.0.0 documentation.pdf](#)[14]
- [DCT 6.0.0 documentation.pdf](#)[15]
- [DCT 5.0.0 documentation.pdf](#)[16]
- [DCT 4.0.0 documentation.pdf](#)[17]
- [DCT 3.0.0 documentation.pdf](#)[18]
- [DCT 2.2.0 documentation.pdf](#)[19]
- [DCT 2.0.0 documentation.pdf](#)[20]

---

[13] https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%208.0.0%20documentation.pdf?
api=v2&cacheVersion=1&modificationDate=1728404902396&version=1

[14] https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%207.0.0%20documentation.pdf?
api=v2&cacheVersion=1&modificationDate=1728404901408&version=1

[15] https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%206.0.0%20documentation.pdf?
api=v2&cacheVersion=1&modificationDate=1728404900556&version=1

[16] https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%205.0.0%20documentation.pdf?
api=v2&cacheVersion=1&modificationDate=1728404899799&version=1

[17] https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%204.0.0%20documentation.pdf?
api=v2&cacheVersion=1&modificationDate=1728404899223&version=1

[18] https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%203.0.0%20documentation.pdf?
api=v2&cacheVersion=1&modificationDate=1728404898245&version=1

[19] https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%202.2.0%20documentation.pdf?
api=v2&cacheVersion=1&modificationDate=1728404897726&version=1

[20] https://delphixdocs.atlassian.net/wiki/download/attachments/243466282/DCT%202.0.0%20documentation.pdf?
api=v2&cacheVersion=1&modificationDate=1728404897278&version=1

2 Quick references

- [Overview](https://dct.delphix.com/docs/latest/dct-overview)[21]
- [Deployment](https://dct.delphix.com/docs/latest/deployment)[22]
- [New features](https://dct.delphix.com/docs/latest/new-features)[23]
- [Fixed issues](https://dct.delphix.com/docs/latest/fixed-issues)[24]
- [Concepts](https://dct.delphix.com/docs/latest/dct-concepts)[25]

---

21 https://dct.delphix.com/docs/latest/dct-overview
22 https://dct.delphix.com/docs/latest/deployment
23 https://dct.delphix.com/docs/latest/new-features
24 https://dct.delphix.com/docs/latest/fixed-issues
25 https://dct.delphix.com/docs/latest/dct-concepts

3 Release notes

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

3.1 New features

> ⚠ DCT support for Docker Compose deprecated with DCT version 13.0.0 (released in January 2024), starting a 12-month depreciation period for all supported versions on Docker Compose. All prior and current product versions on Docker Compose will have continued support until January 2025. Production DCT workloads in Docker Compose are not supported.
> **It is highly recommended that new DCT installations are performed on Kubernetes**[26]**, OpenShift**[27]**, or Appliance OVA**[28].

> ⚠ DCT currently supports Continuous Compliance Engines on version 6.0.13.0 and up, however, Delphix highly recommends upgrading Compliance Engines up to version 20.0.0.0 before connecting them to DCT. Please see this technical bulletin[29] for more details.

3.1.1 Release 22.0.0

- **New Appliance OVA deployment method**
  DCT now offers an on-engine deployment method, distributed as a closed appliance virtual machine image (similar to the standard Continuous Data and Continuous Compliance engines), providing a streamlined way to deploy DCT directly on the engine. Use the pages below to get started.

---

26 https://dct.delphix.com/docs/latest/installation-and-setup-for-kubernetes

27 https://dct.delphix.com/docs/latest/installation-and-setup-for-openshift

28 https://dct.delphix.com/docs/latest/dct-on-engine

29 https://support.delphix.com/Support_Policies_and_Technical_Bulletins/Technical_Bulletins/ TB117_Engine_Sync_Import_and_Algorithm_Rekey_Operations_May_Fail_Unexpectedly_on_Engines_Registered_With_ DCT

- **Faults, alerts, and actions data**
  Continuous Data Engine details now shows faults, alerts, and actions data to provide better auditing.
- **Bulk environment refresh support**
  DCT can now perform environment refresh from both the API and UI for one or more environments at the same time with a single operation. This is particularly powerful when combined with tag-based filtering.
- **Enterprise Compliance Visibility**
  DCT provides visibility of nearly all Compliance Engine objects. This critical central reporting capability provides filterable list views with richly detailed contextual views customized for each object type.
- **One (1) hour session time-outs**
  A new enhancement that allows customers to configure both session expiry and inactivity time. The user's session will expire based on the configured expiry duration. Additionally, the configured inactivity time will determine when the session expires if there is no activity within that specified window. These properties can be configured using the global properties API. The default expiry time is 24 hours and inactivity time is 1 hour.
- **API Key Management and Rotation**
  Introduced the ability to define an API Key expiration time. In addition, administrators can now add, regenerate, or delete an API key on any user account.
- **ServiceNow Spoke**
  Added certification for the Vancouver and Washington releases.

3.1.2 Release 21.0.0

- **VDB Group configuration management (add/remove VDBs and manage the order of execution)**
  DCT can add/remove VDBs to/from the group and arrange the order from which bulk operations (like refresh) will be applied. This is particularly crucial for data sources like Oracle EBS, where application files and the underlying Oracle database need to be refreshed in a specific order.
- **VDB Group Bookmark creation support**
  DCT can create a bookmark across one or more VDBs contained in the group, to provide a human–readable reference point for that specific point in time.
- **VDB Group Refresh support**
  DCT can refresh one or more VDBs contained in the group based on a specific bookmark reference or to the latest snapshot on each VDBs' respective parent (VDB or dSource).
- **Oracle Staging Push support**
  DCT can now manage Oracle-based staging push ingestion from the DCT UI, including the updating of those source settings.
- **Additional Oracle provisioning modes support**
  DCT has added multiple provisioning mode features for Oracle VDB provisioning including:
  - Invoke datapatch support after provisioning VDB/vPDB SQL updates.
  - Support for customized local listeners for RAC VDBs and vCDBs.
  - The ability to update Oracle listeners.

- **Engine-specific faults, alerts, and actions data**
  DCT now reports faults, alerts, and actions data within an engine's details view to provide better auditing capabilities from the UI.
- **Bulk Environment Refresh support**
  DCT now can perform environment refresh from both API and UI for one or more environments at the same time with a single operation. This is particularly powerful when combined with tag-based filtering.
- **Compliance Application Settings Read Only**
  DCT can now ingest and report on all application settings within all connected engines and display those details within a list, including a detailed set of UI views.

3.1.3 Release 20.0.0

- **Replica reporting details**
  DCT can now break down object-specific detail for replication targets to pair with existing details, such as the last successfully completed replication. This will provide additional visibility for administrators tracking Disaster Recovery configurations and statuses.
- **Failover and failback support**
  DCT supports failover with failback support on a replica's detail screen. This enables administrators to identify downed engines and orchestrate disaster recovery centrally from DCT's user interface.
- **Data Vault replication policy configuration**
  DCT now supports the ability to configure replication profiles for Data Vault Target engines from the UI.
- **Discovery Policies (formerly profile sets) read-only**
  DCT now visualizes existing Discovery Policies (known as Profile Sets on a Continuous Compliance engine) on connected compliance engines. This enables administrators to visualize how the engine employs classifiers (or legacy expressions) during Sensitive Data Discovery from a single page.
- **VDB Group List and details page**
  DCT now visualizes VDB groups via a list view found under the Data tab that shows all VDB groups, as well as their statuses and a refresh timestamp. Viewing a VDB Group's details will show all grouped VDBs, with other insights like critical time and masking details.
- **VDB Group creation wizard**
  VDB groups can now be created via the DCT UI from the VDB Group list page. This creation wizard will guide users through picking the correct VDBs with a filterable list of attributes.
- **Refresh and update bookmarks**
  While creating bookmarks from the timeline was introduced in the previous release, users can now update a bookmark and refresh using that bookmark directly from the Timeline, which was not previously available.

3.1.4 Release 19.0.0

- **Snapshots and bookmarks on the VDB timeline**
  The VDB timeline now presents snapshots and bookmarks together in chronological order, making it

easier for any user to understand the bookmark's context and the rough time a refresh will take to complete from the Log Delta time.

- **Oracle dSource hook support**
Hook management is now available for Oracle dSources. The Pre-Sync, Pre-LogSync, and Post-Sync hooks can be created, edited, reordered, and deleted from the dSource's Hooks tab.
- **Oracle add/update environment users**
Users can now add or update Oracle environment users. This is available from the Overview tab, inside of the infrastructure Connections detail page. Users can only be added with a username and password, at present.
- **Oracle cluster node operations**
Users can now add and delete cluster nodes for an Oracle environment. This feature is available under the Attributes tab inside the infrastructure Connections detail page.
- **VDB-to-bookmark tag inheritance**
The default permission model has been updated so that users who have Edit Tags permission on a VDB will also have Edit Tags permission on the VDB's bookmarks.
- **Data Classifier support**
All Classifiers and Expressions (Legacy), from any connected Continuous Compliance engine, can be viewed within DCT's Data Classifier tab. This enables administrators to better monitor their classifier configurations and easily share details with security teams.
- **Failover support**
Added support for failover replication modes that administrators can use to centrally create, manage, and monitor failover and failback disaster recovery scenarios between engines from the DCT UI.

3.1.5 Release 18.0.0

- **Custom hook support**
Hook support has been augmented for pre-existing VDBs. Users can now create, update, and delete hooks, in addition to determining the hook execution order.
- **Selective Data Distribution (SDD) replication support**
DCT now supports a SDD-specific user experience that creates replication profiles specifically for compliant prod-to-non-prod VDB distribution for developer access to masked data.
- **Rule Sets detail views**
DCT's Rule Set UI now includes detail views on associated jobs, table and column-level detail on masking rule assignments, and a sensitive data coverage analysis to provide a visualization of data risk.
- **Bookmark migration for VDB Groups**
The Delphix Self-Service migration workflow has been expanded to now support multi-VDB containers. These will be recreated in DCT as VDB Groups. Read our Delphix Community blog post to learn how to migrate single or multi-VDB containers, templates, and bookmarks.

3.1.6 Release 17.0.0

- **Delphix Self-Service to DCT migration utility**
  Administrators can now migrate existing Delphix Self-Service bookmarks, containers, and permissions to DCT.

- **Compliance enterprise visibility**
  This release expands the user interface to include a list view for Data Classes (formerly known as domains), Compliance File Uploads, Rule Sets, and Connectors. In addition, many of those objects now have a "Details" view that highlights complex relationships and metadata to help administrators proactively manage the connected compliance estate. Rule Sets now have a list view.

1 A connector's Details view featuring compliance coverage that identifies sensitive data risk and scope determined from profiling.

- **Replication management**
  Admins can create, update, monitor, and delete replication profiles. Replication statuses are now visible within the DCT UI to visualize completed, running, stopped, or inactive replication streams paired with meaningful metadata like last run time.

- **Auto-tagging of custom Tags**
Users can now define (per connected engine) a series of custom tags applied to all objects pulled into DCT via telemetry, enabling a fast and efficient way to populate tags within DCT. This will stay in sync with the engines since objects are created or deleted until the feature is deactivated, during which time the tags will persist.

- **Oracle CDB operations**
Centrally start, stop, and delete Oracle CDBs from DCT and report on those CDBs with enhanced capabilities using the addition of the `enabled`, `group_name`, and `status` attributes.
- **Bookmark data timestamps**
The VDB Timeline and Bookmarks pages now have bookmark data timestamps available to enable users to track the lineage of the source data.
- **DCT reporting framework updates**
We have introduced column-based filtering on all list views to provide a spreadsheet-like user experience. Session retention has also been introduced for the VDB list view to maintain page configurations such as filters, sorting, search, etc. even after navigating off of the page.
- **Compliance uploads (read-only)**
This feature shows uploaded files that are on connected Continuous Compliance Engines in DCT. Each upload item will show information about the upload, like file name, file reference, size, and more. It also shows the "consumer", the object that uses and depends on the file, under the Consumers tab or by using the `/masking-file-uploads/{maskingFileId}/consumers` API endpoint. At this time, the feature is read-only and only shows files associated with custom

algorithms and custom extended plugins. Other files that a user might have on a Continuous Complaince Engine are not yet shown (such as Connector files, for example).

3.1.7 Release 16.0.0

- **Data Risk report**
  DCT now shows sensitive data risk for all database connectors on DCT-connected Continuous Compliance Engines. This includes details on whether a data connection has been profiled, contains sensitive data (discovered during profiling), and whether that sensitive data has been masked.

> ℹ️ Record coverage metrics are only supported on Continuous Compliance Engines running version 20.0.0.0 or higher. File masking connections will be added in subsequent releases.

- **Block Storage report (capacity trends)**
  DCT now persists and shows storage trend data for connected Continuous Data Engines. This data defaults to aggregating all connected engine storage, but can be filtered to show a specific engine or group of engines. This report also includes a three and six month projection to assist administrators with capacity forecasting.
    - **Block Storage report (Top 5 Storage Usage Trends):** Building off the Capacity Trends view, DCT shows a breakdown of the top 5 engines (by used capacity) on a stacked trend chart to add a level of granularity on the largest storage consumers. This chart can be filtered via search and includes a projection capability for forecasting.
    - **Block Storage report (Top N Storage Capacity):** DCT now drives better granularity on block storage detail (Reserved, dSource, VDBs, Free Capacity) at the engine level. This view defaults to the top 10 or 15 engines by block storage volume, but this view can be filtered using search.
- **Storage Savings report**
  DCT now shows a granular Storage Savings view to assist users in determining aggregate object counts, storage consumption, and estimated savings. This report defaults to all objects on all engines, but can be filtered via search.
- **Autotagging**
  DCT now has the ability to propagate engine-based groupings as tags in DCT. This includes *Dataset Groups* for Continuous Data, and *Environments* and *Applications* for Continuous Compliance. This feature can be turned on at engine connection or from the engine configuration screen and will stay in sync with an engine until turned off. Auto tags will persist, but will no longer auto update.
- **Replication profile management support**
  Users can now create, update, and delete replication profiles centrally via DCT APIs. Additionally, DCT now has the ability to create replication profiles from the DCT UI via a wizard that can be found on the Replication Profiles List page. Additional replication UI functionality will be added in subsequent releases.

- **Oracle Container Databases enable/disable actions**
DCT users are now able to better manage their Oracle container infrastructure (CDB and vCDB) centrally via the DCT API and UI with enable/disable functionality.
- **Simplified Bookmark sharing**
Bookmark privileges have been updated to simplify the sharing process. By default, Bookmark visibility is set to *Private*, where a user's read and refresh Bookmark permissions are based on the user's same permissions on the parent VDB. Bookmarks can be easily shared with all compatible VDBs (i.e. VDBs with the same ancestry) by toggling the Visibility value as *Public*. This aligns closely with the sharing of bookmarks in Delphix Self-Service.
- **Source and origin Timeline display**
Introduced additional detail to each Timeline on the Timeline History tab. Information helpful to users understand where their data is coming from. The Origin Source and Origin Timeline Location details the originally ingested dSource. The Parent Source and Parent Timeline Location details the last source in the lineage. This might be the original dSource, masked parent VDB, or even a sibling who shared a bookmark.
- **Enhanced Bookmark permissions**
New permissions are introduced for bookmarks operations like create, read, update, etc. Bookmarks also inherit permissions from its parent dataset (VDBs and Bookmarks) now.
- **Terraform Provider**
Changed the default value of `wait_time` to 0 and removed the `storage_size` variable to resolve upgrade issues. In future versions of the Terraform Provider, full upgrade-in-place support will be implemented.
- **Azure support for PasswordVault**
Azure support is now added for PasswordVault, joining HashiCorp and CyberArk.

3.1.8 Realase 15.0.0

- **Enhanced bookmarks organization/search**
The bookmark user experience, including search on VDB and dSource Details pages, has been updated. VDB visibility on the central bookmark, the VDB's bookmark, and VDB wizard pages has been improved. Bookmarks are also supported via DCT-orchestrated replication.
- **Timeline History UX updates**
DCT has consolidated the Active Timeflow and Timeflow History tabs into a single Timeline tab. The new page is similar to the former timeflow history, but now with a search bar and sorting capabilities. You can now sort upon timeline activation [default], timeline creation, parent timeline location, and origin timeline location to enable better identification and filtering for the varying timelines of your datasets.
- **Global Compliance Object Sync Report**
The compliance Global Object is the aggregate representation of all multi-use objects (algorithm, domains, rule sets, etc.) on a Continuous Compliance Engine. These objects, among others, are versioned by the Continuous Compliance Engine to clarify if there have been settings or behavioral changes. DCT now surfaces those versioning details to better illustrate if and where there are differences in objects. This could, for example, expose whether or not two engines will mask with identical outputs or not.

- **Oracle CDB Inventory report**
  DCT now features an Oracle Container Database (CDB) report that lists both virtual and physical resources across all connected engines, as well as associated pluggable database (PDB) relationships.
- **MongoDB and delimited file support via DCT UI**
  DCT now supports executing and modifying existing MongoDB and delimited file Hyperscale Compliance jobs.
- **Kubernetes driver**
  Support has been added for taking a VDB volume snapshot and provisioning a VDB from a snapshot. In addition, various bug fixes impacting general dataset creation and deletion have been resolved.
- **Terraform provider**
  There is a new Database resource in the Terraform Provider that allows users to create and delete PostgreSQL environment source configurations.

3.1.9 Release 14.0.0

- **New APIs for PostgreSQL**
  New APIs have been added, providing the function to create, update, or delete database APIs for PostgreSQL.
- **Bug fixes and improvements**
  This release focuses on bug fixes and general improvements.

3.1.10 Release 13.0.0

- **Algorithm centralization**
  To enable better visibility into compliance, masking algorithms across all connected Continuous Compliance Engines can now be centrally viewed in DCT.
- **Expanded source linking**
  Over the past few years, Staging Push has been introduced, allowing users to bring their own data to Continuous Data. The DCT UI now supports linking for Oracle, SQL Server using Staging Push. In addition, ASE and SDK connectors such as Postgres, Db2, MySQL, HANA, and MongoDB.
- **Hyperscale Compliance UI enhancements**
  Several improvements have been added, including enabling the management of Continuous Compliance Engine assignments of Hyperscale Jobs.
- **DCT Toolkit enhancements**
  Based on user feedback, several improvements have been made to the fully supported CLI for DCT. Notably, dct-toolkit allows users to provide engine names as values for 'id' options.
- **Continuous Data jobs progress**
  Users can track job progress in real-time through the Operations UI.
- **Improved Replication visibility**
  Replication, used for data movement, disaster recovery protection, Sensitive Data Distribution, and more, requires setting up relationships between Continuous Data Engines. To provide better visibility on these relationships, they have been added to DCT. To further simplify this user experience, the target Continuous Data Engine and dataset are now linked together.

- **Engine Performance trend lines**
  Historical trend lines have been added to the recently introduced Engine Performance reports.

3.1.11 Release 12.0.0

- **Delphix Kubernetes driver**
  Virtual databases (vDB) can be provisioned into containers through Helm charts and `kubectl` commands using a new, pre-built DCT integration. Teardown of these containers can also be automated, enabling ephemeral infrastructure for Kubernetes-based applications.
- **Engine performance reports**
  New Continuous Data and Compliance reports have been added to improve the central management of engine infrastructure performance. You can now view allocated resources and track performance for all connected engines, including disk latency, network latency, and throughput.
- **Hyperscale Compliance jobs**
  DCT now has support for creating, managing, and executing Hyperscale Compliance jobs. The concept of Engine Pools have been introduced along with several other enhancements to refine the overall experience.
- **User experience improvements**
  Several enhancements including customizing the columns across all tables, increased color contrast for better accessibility, simplified navigation across tabs, and a variety of visual style improvements.
- **Replication tag inheritance**
  DCT now offers API-based simplified tag management for parent/child replication deployments, optionally syncing source tags to a target replica.
- **Oracle and SQL Server source linking**
  Linking sources for Oracle and SQL Server is now supported in the user experience, along with creating bookmarks.
- **Terraform provider**
  A new Terraform resource facilitates creating and deleting Oracle dSources. This expands on the existing data source support with the provider.

3.1.12 Release 11.0.0

- **dSource linking support (API only)**
  The ability to create dSources for MSSQL (single instance and cluster), Oracle Staging Push, and MSSQL Staging Push has been added. In addition, dSource linking for older engine versions back to 6.0.7 is now supported.
- **Last Refresh column added to the VDB tab**
  A new column called Last Refresh has been added to the VDB tab, which shows the last refresh date/time of a VDB.

- **download.delphix.com**[30] **enhancements for container-based upgrades**
  Changes have been made to download.delphix.com[31], making it easier to build installation and upgrade automation for DCT using repository-management products like JFrog Artifactory.

  - If you are installing DCT for the first time, please follow the standard installation documentation.
  - If you have an existing installation of DCT you must take the following steps to perform upgrades:

    - Update the registry URL to the new address (http://dct.download.delphix.com) in the values.yaml file.
    - Update the credentials with the new permanent credentials in the values.yaml file.
    - If your organization leverages an allow (white) list, please update the list with the new repository URL(s).

3.1.13 Release 10.0.0

- **dSource linking support (API only)**
  We have added the ability to create dSources for Oracle, ASE, and Postgres via API. This is currently available for Delphix Continuous Data Engines on version 8.0 and above. Future releases will include full support for all data sources and inclusion of older engine versions.
- **Hyperscale Compliance UI (read-only)**
  DCT has released a Hyperscale UI under controlled availability. Please reach out to your account team to activate this feature. This enhancement includes the ability to:

  - Register Hyperscale Orchestrators with DCT.
  - Visualize Hyperscale infrastructure configuration parameters such as clusters of engines and mount points.
  - Report on Hyperscale Jobs and Job executions, including near-realtime process feedback.
- **Compliance Job Executions report**
  For connected Continuous Compliance Engines, DCT now has a new report under DCT Insights that displays all recent job executions (including DCT initiated, engine initiated, and Hyperscale initiated jobs) as well as relevant compliance metrics.
- **Provisioning wizard enhancements**
  The DCT provisioning wizard now supports provisioning Oracle multi-instance (multi-tenant) and ASE VDBs.
- **Operations page**
  The DCT Operation page now includes initiator details, which provides visibility of what user/account initiated a job.
- **Support for creating a bookmark at point-in-time**
  DCT now has the ability to create bookmarks on VDBs under a specified point-in-time via both API and UI. This feature is only compatible for databases that support LogSync and have it enabled.

---

30 http://download.delphix.com/
31 http://download.delphix.com/

- **VDB lock/unlock**
  Users now have the ability to lock a VDB, preventing any sort VDB-based activity (refresh, start, stop, delete, and create bookmark).
- **Custom hook support in the DCT UI**
  Delphix users now have the ability to add custom hooks to a VDB at the time of provision (as part of the provision wizard UI) and update them under a VDB details page, so that they will execute during refresh operations.

3.1.14 Release 9.0.0

- **Jenkins support**
  An official Jenkins plugin is now available for Data Control Tower, joining the existing ServiceNow and Terraform integrations. This plugin helps automate the use of data in your CI/CD pipelines and includes support for provisioning and destroying VDBs.
- **DCT Toolkit**
  We've launched a new command line interface (CLI) offering, the DCT Toolkit. The DCT Toolkit allows for remote operation and control of Data Control Tower via your local terminal. Over time, this will replace the previously available DxToolkit.
- **Infrastructure wizard**
  The create infrastructure flow now supports adding Windows standalone and cluster hosts, as well as Linux cluster hosts. This is expanded from the previous flow that supported standalone Linux hosts. This differs from the workflow on the Continuous Data engine in that it represents a single place to create environment host connections for all connected Delphix engines.
- **Provisioning wizard**
  The data provisioning workflow now includes additional database types, including Oracle Single Instance Multi-tenant, Oracle Multiple Instance Single Tenant, and SQL Server Multiple Instance Single Tenant.
- **Operations dashboard improvements**
  Previously, all users could view all operations run by any user on the Operations page. Role-based access can now be provided to different users, showing only operations the user has access to. Simplified text is now used to define the Type column, as well as the Type and Engine Name in the details page.
- **Activity Audit Log Summary report**
  The Activity Audit Log Summary provides a high-level audit log summary capturing the utilization of DCT by displaying user activity and the historical count of actions executed within the platform.
- **Replication mappings**
  This feature helps users differentiate between replicated objects and original objects, in case of master and replicated engine, both are registered with DCT.

3.1.15 Release 8.0.0

- **Operations dashboard**
  Monitor and manage enterprise data activities in real time using a new central view. This provides

visibility to the current status across the full complement of Delphix transactions, including provision, refresh, teardown, and compliance jobs.

- **Provisioning wizard enhancements**
  The data provisioning wizard has been expanded to support additional types, including Oracle Single Instance Linked CDBs and Microsoft SQL Server Single Instance workflows. This will now allow you to provision more data types directly from Data Control Tower.
- **Advanced search tags support**
  The advanced search capabilities now support all user-generated tags. You can use personalized tags related to your unique business needs to refine your search results, such as team names or other specific data points.

3.1.16 Release 7.0.0

- **Provision VDB UI**
  Extending the Developer Experience capability in DCT, users can now provision single-tenant Oracle databases from the user interface using an intuitive wizard workflow.
- **Refresh VDB UI enhancements**
  The VDB list can now be opened in a searchable, paginated list selector from within a dialog by clicking the select button in the input. Additionally, refreshing a VDB by a bookmark is now available.
- **VDB template import**
  Importing and removing imported VDB templates from connected engines is now an available action from the "VDB Config Templates" page.
- **Environment details enhancements**
  Managing cluster environment infrastructure has been made easier with the ability to edit host details directly from the page.

3.1.17 Release 6.0.0

- **Developer Self-Service UI**
  Developers and admins now have the ability to centrally orchestrate common Continuous Data and developer operations from the DCT UI. This includes the ability to refresh, rewind, bookmark, and bookmark share (refresh to relative). This functionality also exposes the notion of **time flows** (non-active timelines), which is a critical tool for viewing past work on a VDB, such as the chronology of test results.
- **Central compliance orchestration**
  The compliance job UI now enables job orchestration and reporting. This includes Job Copy and Execute functions as well as a complete historical job execution log within each compliance job's details view.
- **Bookmark UI**
  Developers and admins now have added visibility of bookmarks, both globally and contextualized, to individual VDBs. These visualizations are dual purpose; for administrators, these screens help with reporting and tagging on bookmarks, while for developers, these screens act as a catalog of actionable data references.

- **Global Bookmark List**
  View all bookmarks across your entire connected Delphix ecosystem. This screen will show bookmarks for both single VDBs and VDB groups.
- **VDB Bookmark List**
  See all bookmarks tied to this individual VDB. This is helpful for sharing bookmarks with team members who have a compatible VDB (same parent and provision point).
- **Environment details page**
  Users can now orchestrate common environment actions via the DCT UI including enable, disable, environment refresh, and delete, as well as editing host details. Note, editing host details is only applicable to standalone environments at this time.
- **Access visibility**
  Object detail pages will include an access tab that provides visibility to user access and the associated permissions for each user. This is a critical enabler for permissions visibility and auditing.
- **Copy/delete functionality on role scopes**
  Scoped roles can now be copied and deleted within the DCT UI. This will enable easier administration, especially around the use of custom roles, as admins can now copy and modify new roles from templates.
- **External Postgres DB support**
  DCT now supports the use of an external Postgres database to house DCT metadata. Previously, DCT supplied and managed its own database, requiring persistent storage within the container platform.

3.1.18 Release 5.0.1

3.1.18.1 Enhancements

- **Data scoped Access Group**
  - **Enhancement in roles**
    Associated permissions in roles are changed from 'string' type to 'permission object' type. For details, see the Role schema in the API References (see page 380).
  - **Custom roles**
    In addition to the 5 pre-seeded fixed roles (Admin, Monitoring, DevOps, Masking, and Owner), DCT provides flexibility to create new custom roles as per user need. Users (Accounts) can create new custom roles by encapsulating any combination of permissions. The custom roles can be configured through a UI configuration screen (screenshot below), in addition to a set of APIs to manage roles. For details, see the API References (see page 380).
- **Updates to existing RBAC model**
  For better usability and allow to set more granular permissions there are following enhancements in the RBAC model:
  - **Renamed Access Group "Policy" to Access Group "Scope"**
  - **Renamed the following APIs related to Access Group actions**

- **<u>Add scope</u>** to an Access Group

  ```
  POST: /access-groups/{accessGroupId}/policies → POST /access-
  groups/{accessGroupId}/scopes
  ```

- **<u>Remove scope</u>** from Access Group

  ```
  DELETE /access-groups/{accessGroupId}/policies/{policyId} →
   DELETE /access-groups/{accessGroupId}/scopes/{scopeId}
  ```

- **<u>Get</u>** Access Group scope

  ```
  GET /access-groups/{accessGroupId}/policies/{policyId} → GET /
  access-groups/{accessGroupId}/scopes/{scopeId}
  ```

- **<u>Update</u>** Access Group scope

  ```
  PATCH /access-groups/{accessGroupId}/policies/{policyId} →
   PATCH /access-groups/{accessGroupId}/scopes/{scopeId}
  ```

- **<u>Add object tags</u>** to Access Group scope

  ```
  POST /access-groups/{accessGroupId}/policies/{policyId}/object-
  tags → POST /access-groups/{accessGroupId}/scopes/{scopeId}/
  object-tags
  ```

- **<u>Remove object tags</u>** from Access Group scope

  ```
  POST /access-groups/{accessGroupId}/policies/{policyId}/object-
  tags/delete → POST /access-groups/{accessGroupId}/scopes/
  {scopeId}/object-tags/delete
  ```

- **<u>Add objects</u>** to Access Group scope

  ```
  POST /access-groups/{accessGroupId}/policies/{policyId}/objects
  → POST /access-groups/{accessGroupId}/scopes/{scopeId}/objects
  ```

- **<u>Remove objects</u>** from Access Groups scope

  ```
  POST /access-groups/{accessGroupId}/policies/{policyId}/
  objects/delete → POST /access-groups/{accessGroupId}/scopes/
  {scopeId}/objects/delete
  ```

- **Renamed the "everything" flag to "scope_type"**
  In order to make it more understandable, we have renamed the everything flag to scope_type.
  There are three possible values for scope_type i.e. SIMPLE, SCOPED and ADVANCED. The
  value SIMPLE corresponds to everything=true and SCOPED corresponds to everything=false.
  The value ADVANCED for scope_type is new enhancement to setting permissions which
  allows users to set permissions (e.g. READ, DELETE) for an object. There is more information
  about ADVANCED scope in next section.

- **Access Group Scope: Advanced scope type**
  In Add objects to access group scope API, now user can define permissions level checks as
  well for an object. For example, earlier when object_id and and object_type are provided in

request payload, all permissions that are defined in scope are applied to this object. But now user can define specific permissions.

- **Masking jobs**
    - CRUD APIs, COPY, Connectors CRUD
- **Masking job execution**
    - Connector Credentials
    - Execution API

3.1.18.2 Custom roles

- Accounts can create new instances of role encapsulating any combination of permission.
- Role name must be unique.
- Custom roles can be updated. Accounts can add or remove permissions to/from the custom roles.
- Custom roles can be deleted. (If they are not associated with any Access Group).

3.1.19 Release 4.0

- Environment Overview List
- Un-virtualized Source Sizing Report
- Global VDB Templates
- Scoped Access Control
- LDAP/AD and SAML/SSO Configuration UI

3.1.20 Release 3.0

- Cluster Node (RAC) management APIs
- Ability to disable username/password authentication globally
- LDAP/Active Directory groups
- CDBs/vCDBs APIs
- VDB Provisioning / update for EDSI (AppData) platforms
- Engine registration wizard
- Access Groups Management UI
- Compliance Engine Management

3.1.21 Release 2.2

3.1.21.1 Deployment

- Introducing Kubernetes and OpenShift support

3.1.21.2 APIs

- Registration of Continuous Compliance Engines
- Masking Connectors
- "Move Masking Job"
- Masking of mainframe objects
- Provisioning enhancements for Oracle multi-tenant and RAC
- LDAP/Active Directory authentication
- Password management
- Initial access management by Permissions, Roles, Policies, and Access Groups (permissions applied to all objects of a type e.g. Stop VDB permission on all VDBs)
- Distributed tracing and logging (Trace ID propagated down call stack)
- Bulk delete of tags

3.1.21.3 UI

- Continuous Data
    - Added tag support to the Infrastructure page
    - New dSources page
    - New VDBs page
- Insights
    - Added an export behavior to the Storage Summary report
    - New dSource Inventory report
    - New VDB Inventory report
- Admin
    - New Accounts page

3.2 Fixed issues

3.2.1 Release 22.0.0 changes

Bug Number	Description
APIGW-10015	Fixed a bug where import of config templates fails if there are replicated config templates.

3.2.2 Release 21.0.0 changes

Bug Number	Description
APIGW-9536	Reduced memory requirement to collect rule sets from connected compliance engines.

3.2.3 Release 20.0.0 changes

Bug Number	Description
APIGW-8504	Resolved an issue where masking job execution sometimes failed due to concurrent updates.

3.2.4 Release 18.0.0 changes

Bug Number	Description
APIGW-8771	Fixed a memory leak, where the virtualization-app was restarting frequently. The reason for the memory leak was a background bookmark migration job for timestamp if the number of bookmarks are more than 100.

3.2.5 Release 17.0.0 changes

Bug Number	Description
APIGW-8156	Fixed a timeout issue when creating a bookmark for a large vDB group.

3.2.6 Security fixes

Bug Number	Bug Introduced in	Description	Security bulletin
APIGW-8057, APIGW-8113, APIGW-8114, APIGW-8142	8.0.0	Arbitrary code execution can be performed by any user permitted to log into the Delphix Data Control Tower (DCT) application, even without admin permissions.	TB118

3.2.7 Release 16.0.1 changes

Bug Number	Description
APIGW-8144	Fixed an issue that masking jobs of a registered engine are not shown in DCT.

3.2.8 Release 16.0.0 changes

Bug Number	Description
APIGW-7480	Fixed the UI on the Infrastructure Connections screen such that long environment names will wrap around instead of spilling unto the next column.

3.2.9 Release 14.0.0 changes

Bug Number	Description
APIGW-7090	Fixed an issue where the username was not displayed in the top navigation bar, if logging in using anything other than username/password.

3.2.10 Release 13.0.0 changes

Bug Number	Description
APIGW-4971	Fixed a problem when `VDB_GROUP_REFRESH` job's update time did not get updated on receiving job updates for the underlying `VDB_REFRESH` jobs.
APIGW-6705	Fixed an installation problem when masking database migration runs out of memory for large tables.

3.2.11 Release 12.0.0 changes

Bug Number	Description
APIGW-6240 APIGW-6241	Fixed a memory exhaustion issue that caused frequent HTTP 500 errors.
APIGW-5052	Fixed an issue when viewing the Summary screen after provisioning a vDB in DCT where the dSource summary name was showing the vDB name.
APIGW-5511	Added the ability to unregister engines in the UI.
APIGW-5585	Fixed an issue where users were unable to logout following SSO token expiration.

3.2.12 Release 10.0.1 changes

Bug Number	Description
APIGW-5406	Fixed an issue where the VDBGroup update API gives an internal error if vdb_ids are the same as ones already present in VDBGroup.
APIGW-5419	Fixed an issue where the VDBGroup delete API gives a 404 error if underlying VDBs are already deleted or not present.

APIGW-5418, APIGW-5517	Fixed an issue where VDB provisioning via DCT GUI fails with, "There was an error trying to process your request."
APIGW-5570	Fixed an issue preventing the link of a Postgres database as a dSource.
APIGW-5571	Fixed an issue where the Provision VDB -> dSource Listing would not show more than 25 items.
APIGW-5574	Hyperscale datasets tables-or-files list and search APIs now return all instead of filtering by dataset.
APIGW-5578	Fixed an issue where Pagination was not working for some of the pages in a testing environment.

3.2.13 Release 10.0.0 changes

Bug Number	Description
APIGW-3931	Fixed an error found when loading the LDAP config page in the UI, if the LDAP config domains have empty values.
APIGW-3961	Fixed the issue where the Environment link to the dSource detail view is broken.
APIGW-4270	Cleaned up pending jobs left by deleted engines.
APIGW-5056	Addressed VDB provisioning failures where there were a lot of concurrent requests.

3.2.14 Release 9.0.0 changes

Bug Number	Description
APIGW-3772	Replicated VDBs/dSources are now identified in the Provisioning wizard.
APIGW-3931	Fixed a null pointer exception during LDAP configuration without a domain.

Bug Number	Description
APIGW-3979	Fixed an issue where the Import VDB Configuration templates dialog was showing the engine ID in error messages. It is now changed to show engine names.
APIGW-3983	Fixed an issue where a new masking job could not be started from DCT when the previous job was cancelled on the masking engine.
APIGW-4009	Fixed an issue where the first and last name will be cleared if incorrect names were entered for SSO.
APIGW-4010	Fixed an issue in the UI where first and last name attributes cannot be reset.

3.2.15 Release 8.0.1 changes

Bug Number	Description
APIGW-4324	Fixed an issue where users who upgraded to DCT 8.0.0 were not able to interact with the UI or connect to the GraphQL service container.
APIGW-4317	Fixed an issue where an error would occur when searching for a VDB in the relative refresh UI.

3.2.16 Release 8.0.0 changes

Bug Number	Description
APIGW-3764	Removed THE requirement on setting credentials if a masking job execution happens on the origin engine.
APIGW-3771	Allows the policy name to be empty when provisioning a VDB.
APIGW-3783	Allows for an existing ImagePullSecret to be provided to to pull docker images.

Bug Number	Description
APIGW-3985	Fixed the "VDB Container is part of a container" error while refreshing from bookmark directly on the VDB > Bookmark tab.
APIGW-3990	Fixed the broken view for a bookmark that has multiple VDBs on the Data > Bookmark tab.

3.2.17 Release 7.0.1 changes

Bug Number	Description
APIGW-3592, APIGW-3594	Previously, a non-admin user that was granted access to a VDB, but not its environment, would get an error accessing the VDB overview. A fix has been implemented to show that the access error is with the environment and not the VDB.
APIGW-3775	Fixed an issue where refreshing from the bookmark wizard was not showing compatible bookmarks.
APIGW-3831	Fixed a certificates import failure if the truststore is on OpenShift.

3.2.18 Release 6.0.1 changes

Bug Number	Description
APIGW-3460	Fixed a request timeout issue.
APIGW-3395	Fixed an issue where the refresh wizard did not update snapshots when selecting different datasets.

3.2.19 Release 6.0.0 changes

Bug Number	Description
APIGW-3223	Fixed an issue where DCT failed to get info from detached dSources.

3.2.20 Release 5.0.3 changes

Bug Number	Description
APIGW-3344	Fixed an issue causing provision failure from RAC dSource to non-RAC target.

3.2.21 Release 5.0.2 changes

Bug Number	Description
APIGW-2979	VDB refresh will no longer fail if the refresh target name is not unique.
APIGW-2981	Fixed an issue where all the Compliance jobs and source jobs on the engine will be deleted when a Compliance engine is unregistered.

3.2.22 Release 5.0.1 changes

Bug Number	Description
APIGW-2463	The default docker-compose.yaml file is now provided with log size and rotation configured for all containers.
APIGW-2735	Fixed an issue where DCT migration failed with "could not create unique index environments_host_pkey".
APIGW-2828	Helm chart now allows cronjob resource limits to be set via the values.yaml.

3.2.23 Release 3.0.0 changes

Bug Number	Description
APIGW-1785	Fixed an issue where Nginx sometimes failed to start after a server restart.

3.3 Supported versions and upgrade matrix

Data Control Tower has minimum engine versions that are actively tested against to ensure optimal interoperability. Please ensure that all connected engines meet the version requirements:

Delphix Engine	Version
Continuous Data	6.0.8.0 or higher
Continuous Compliance	6.0.13.0 or higher

Users can upgrade directly between DCT versions without needing an interim step (i.e., upgrading to a median version before upgrading to the latest).

Where x represents patch version releases (i.e. 5.0.1, 5.0.2, etc.).

Version	Release date	Can upgrade to
2.0.0	Jun 27th, 2022	2.1.0 - 22.0.x
2.1.0	Sep 8th, 2022	2.2.0 - 22.0.x
2.2.0	Oct 17th, 2022	3.0.0 - 22.0.x
3.0.0	Dec 15th, 2022	4.0.0 - 22.0.x
4.0.0	Jan 19th, 2023	5.0.0 - 22.0.x
5.0.x	Feb 16th, 2023	6.0.0 - 22.0.x
6.0.x	Mar 29th, 2023	7.0.0 - 22.0.x

Version	Release date	Can upgrade to
7.0.x	May 9th, 2023	8.0.0 - 22.0.x
8.0.x	Jun 22nd, 2023	9.0.x - 22.0.x
9.0.x	Aug 1st, 2023	10.0.x - 22.0.x
10.0.x	Sep 13th, 2023	11.0.x - 22.0.x
11.0.x	Oct 23rd, 2023	12.0.x - 22.0.x
12.0.x	Nov 30th, 2023	13.0.x - 22.0.x
13.0.x	Jan 24th, 2024	14.0.x - 22.0.x
14.0.x	Feb 21st, 2024	15.0.x - 22.0.x
15.0.x	Mar 20th, 2024	16.0.x - 22.0.x
16.0.x	Apr 17th, 2024	17.0.x - 22.0.x
17.0.x	May 22nd, 2024	18.0.x - 22.0.x
18.0.x	Jun 19th, 2024	19.0.x - 22.0.x
19.0.x	Jul 17th, 2024	20.0.x - 22.0.x
20.0.x	Aug 5th, 2024	20.0.x - 22.0.x
21.0.x	Sep 3rd, 2024	21.0.x - 22.0.x
22.0.x	Sep 30th, 2024	22.0.x

3.4 Deprecated and end-of-life

3.4.1 Release 19.0.0

3.4.1.1 Deprecation notice

- The masking job `copy` operation (accessible from the *Compliance Jobs > Overview* actions menu in the DCT UI), and the `/masking-jobs/{maskingJobId}/migrate` and `/masking-jobs/{maskingJobId}/copy` API endpoints in the MaskingJobs category, will be deprecated from DCT effective with the release of version 20.0.0.

3.4.2 Release 13.0.0

3.4.2.1 Deprecation notice

- DCT support for Docker Compose is now deprecated with the release of DCT version 13.0.0 (January 2024). This begins a 12-month depreciation period for all supported versions on Docker Compose. All prior and current product versions on Docker Compose will have continued support until January 2025. Production DCT workloads in Docker Compose are not supported.
  - **It is highly recommended that new DCT installations are performed on Kubernetes[32] or OpenShift[33].**

---

32 https://dct.delphix.com/docs/latest/installation-and-setup-for-kubernetes
33 https://dct.delphix.com/docs/latest/installation-and-setup-for-openshift

4 DCT overview

4.1 What is Data Control Tower (DCT)?

Today's application and data landscape is an increasingly complex ecosystem of hosting architectures, often represented by a multi-cloud landscape coupled with an explosion of different platforms and services. This fragmented picture of heterogeneous silos makes data governance, automation, and compliance a herculean, if not, an impossible task.
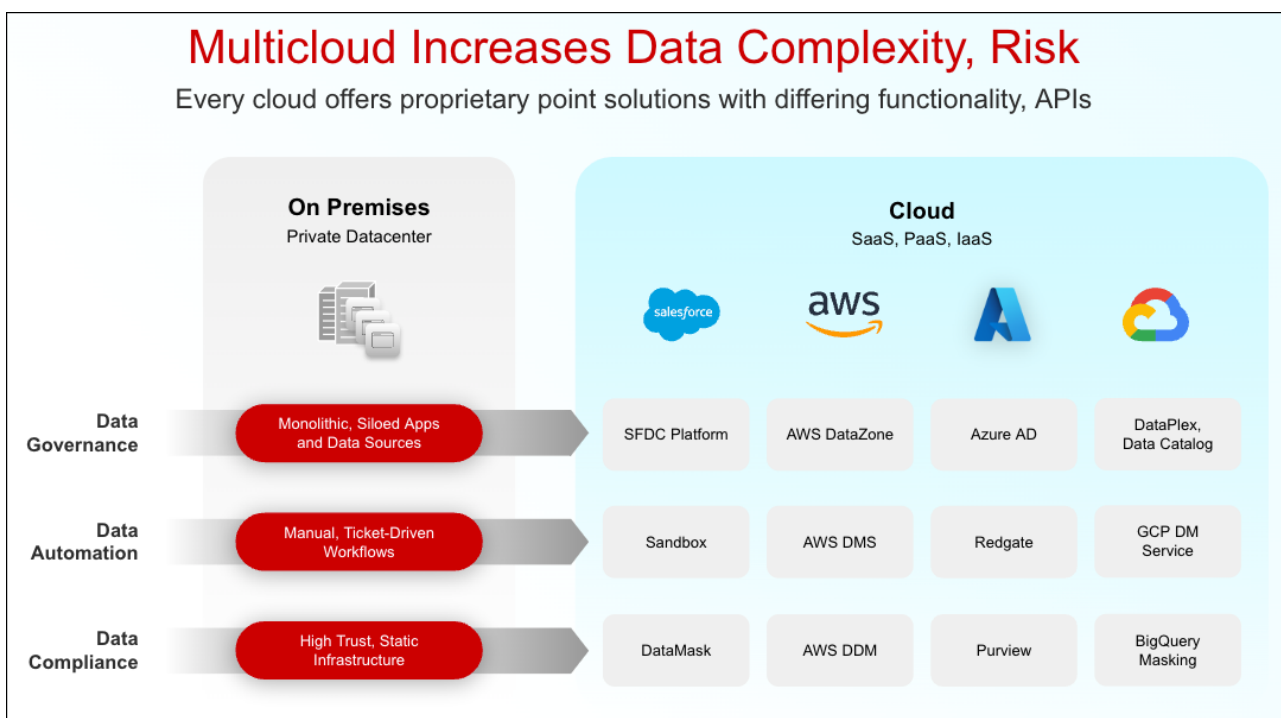
Data Control Tower (DCT) is an enabling Delphix platform that introduces a data mesh to unify data governance, automation, and compliance across all applications and cloud platforms.

Data governance is achieved through operational control and visibility of test data across multicloud applications, databases, environments, and releases. DCT brings data cataloging, tagging, and data access controls for central governance of all enterprise data, while providing the right data at the right time to development teams.

Data automation at CI/CD speed and enterprise scale is easier and more powerful, by combining **DCT with Continuous Data**. A unified API gateway, self-service automation tools, and plug-and-play DevOps integrations streamline the initial configuration and day-to-day workflows.

DCT with Continuous Compliance provides robust data compliance in lower environments, all while reducing costs and enabling fast, quality software development.

5 Getting started

5.1 Planning your deployment

Data Control Tower (DCT) represents a Delphix-wide control plane. It simultaneously powers data governance, automation, and compliance workflows to enable the efficient operation of a broad, complex Delphix deployment at scale. In order to deliver scalability, service-level performance tuning, and robust resiliency, DCT leverages container technology to deliver a bespoke experience for administrative teams based on their own internal guidelines.

> **For Kubernetes, OpenShift, and docker-compose**
> Before starting a DCT deployment, please contact your enterprise IT organization to determine what container platforms, configurations, and policies apply for container-hosted applications. It is helpful to include a container administrator as part of the DCT install process.
>
> **For Appliance (OVA)**
> The above note is not applicable. If you deploy via Appliance (OVA), you will need to choose a hypervisor with similar configurations found here[34].

5.2 Container platform support

Data Control Tower (DCT) supports deployment methods for Kubernetes, OpenShift, and via an Appliance OVA. If you do not see a platform of choice, please reach out to your account team for more details.

> DCT is designed to run and is supported on any Certified Kubernetes platform[35] that supports Helm[36]. The product is also OCI-compliant and may use any container runtime within a certified Kubernetes platform that implements the OCI Runtime Specification including CRI-O, Docker, and Podman.
> Delphix regularly tests against a range of popular Kubernetes platforms with the goal of covering a representative sample of implementations. The following Kubernetes platforms have been explicitly tested by Delphix and are recommended for use: MicroK8s, AWS EKS, Azure AKS, and OpenShift on VMWare vSphere.

---

34 https://cd.delphix.com/docs/latest/checklist-of-information-required-for-installation

35 https://www.cncf.io/certification/software-conformance

36 https://helm.sh/docs/topics/kubernetes_distros/

5.2.1 Kubernetes

DCT currently supports all popular deployment models of Kubernetes as long as the service runs a minimum of Kubernetes 1.25 and above. This includes Amazon Elastic Kubernetes Service (Amazon EKS), Azure Kubernetes Service (AKS), and beyond.

5.2.2 OpenShift

DCT also supports all popular deployment models of Openshift as long as the service runs a minimum version of 4.12 or above. This includes Red Hat OpenShift on IBM Cloud and any other cloud provider's service.

5.2.3 Appliance (OVA)

This new DCT deployment method is distributed as a closed appliance virtual machine image, similar to the standard Continuous Data and Continuous Compliance engines, providing a streamlined way to deploy DCT directly on the engine.

5.3 Data Control Tower deployment architecture

Whether an organization wants to deploy a Data Control Tower (DCT) per business unit (organizational silos), per network (datacenter-specific DCT), or globally (**the most common option**), DCT can adapt to many deployment scenarios.

> Delphix recommends to deploy a single, global DCT for all Delphix Engines, for the purposes of achieving a single control plane and data governance solution.

DCT-based communication is lightweight, requiring simple commands or a small telemetry payload to facilitate most workflows. The below graphic demonstrates this style of communication:
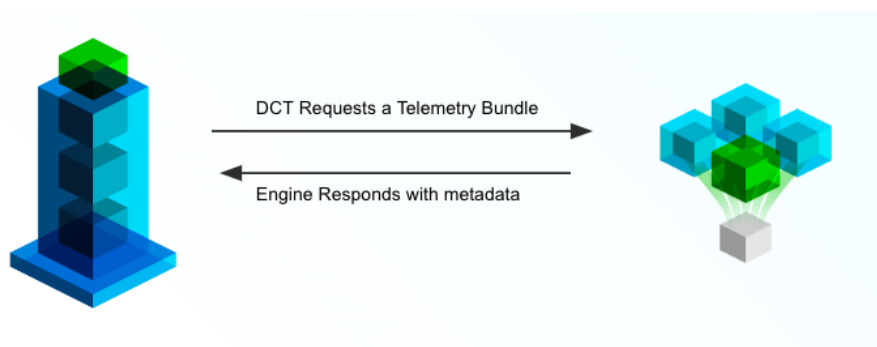
DCT simply logs into the engines as a user would and leverages engine APIs to perform commands or extract metadata.

> ⊟ DCT requires HTTP/HTTPS to facilitate communication with engines and requires ports 80/443 to reach engines in other networks.

> ⊟ DCT does not directly interface with business-critical databases, it will only communicate with engines to perform operations and inquire about system statuses. The Delphix Engine, which is generally co-located with your data, does all the heavy lifting.

5.4 Plan your tagging strategy

DCT tags serve as the Delphix-wide business metadata system. These **Key:Value** pairs can be applied to any object and used for search and filter in virtually every DCT workflow, from automation to administration, all the way to access control.

> ⊟ It is paramount to develop a tagging strategy prior to deployment in order to develop a scalable metadata solution.

Some examples of popular tagging strategies:

Theme	Sub-topics	Tag (Key:Value) example
Owner	Application, Business, Project, Team (scrum, QA,…)	(Owner: Finance App), (Owner: AppTeam Alpha), (Owner: John Doe), …
Application		(Application: Alpha)
Environment		(Environment: Non Production)
Location	Data Center, Region, Name, Cloud	(Geo: West Coast), (Data Center: Azure WC), …

In addition to designing which tags to implement, please consider who will have access to creating tags (i.e. developer vs admin-only, etc.), which will impact how teams are able to collaborate with one another.

Also, Delphix recommends that the DCT administrative team creates Delphix-wide documentation on these tagging standards to reduce the risk of deviation.

5.5 Plan your Access Control strategy

DCT implements a model found in other types of software called **Attribute Based Access Control** (ABAC). This model is incredibly flexible, but requires detailed configurations to perfect your use cases. In DCT's model, there are four entity types (defined below). Familiarize with each entity, as they are the foundational blocks of DCT's ABAC model.

Entity	Description	Managed by
Accounts (aka Users)	A single or shared user who can authenticate with DCT (UI or API).	Create manually or via Identity Provider (IdP), such as SSO or LDAP. Accounts are independent of Delphix Engines.
Access Groups	A collection of accounts that share one or more characteristics, such as a Team or Permission set. Equivalent to an Active Directory group.	Manually created. Populated manually or via the `login_groups` tag.
Roles and Permissions	The collection of read, write, and delete permissions forms a reusable, named role.	Some roles are provided out of the box, but Admins can build their own from the available permissions. Individual permissions are immutable.
Objects	Units, such as VDBs, Bookmarks, and Environments, that are managed across the Delphix Platform.	Automatically identified by DCT from the connected engines. Assigned to Roles via various models. The CD and CC Engines supply these objects.

Each entity is linked to another through manual or automated assignment. A manual (or direct) assignment is a good approach for early implementations, however, that can be challenging to maintain as teams grow. As an alternative, tagging is recommended to perform automatic assignments based on your custom configuration. The below diagram shows how each entity is linked together. The directions below start with **Accounts creation** to **Access Groups** with **Role assignments**, and finish with **Object mappings**.
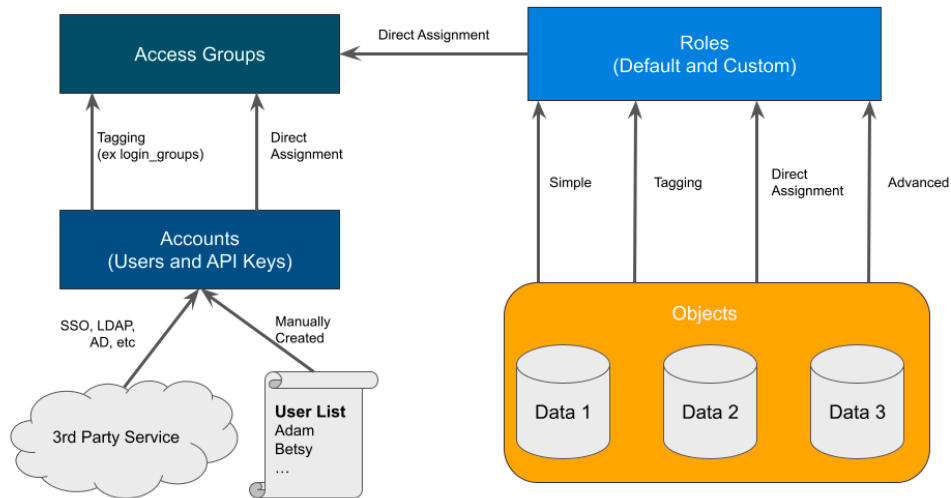
Understanding your team structure is imperative to identify the best access model. Usually, organizations have existing groupings defined in their Identify Provider (IdP). These groups are typically organized in one of two ways:

- A team dedicated towards a central goal (such as a Product Development team).
- A group of individuals with similar permissions (such as Security Administrators).

Understanding the purpose of each group should be a guide in how the Roles and Permissions are designed. For example, the Alpha product development team might have full permission to manage existing VDBs and create new bookmarks for their team's "Alpha" objects. On the other hand, Security Admins might have sweeping read and disable access across the entire platform to ensure compliancy. Iterating through each Access Group and designing custom, but re-useable roles, based on the Principle of Least Privilege[37], will produce a streamlined rollout.

---

[37] https://en.wikipedia.org/wiki/Principle_of_least_privilege

6 Deployment

The articles in this section will explain all of the required steps to deploy DCT on your container platform of choice.

Data Control Tower is a container-based architecture that is currently certified with **Kubernetes** and **OpenShift** to align with common enterprise container standards. The DCT architecture is comprised of multiple micro-services that are each run on individual pods. This lends DCT to be a highly flexible deployment by enabling customers and IT organizations to enact their own backup, scaling, and resiliency standards associated with hosting container-based applications. Below is an architectural diagram of all the services that make up DCT, as well as the persistent storage for maintaining relationship metadata.
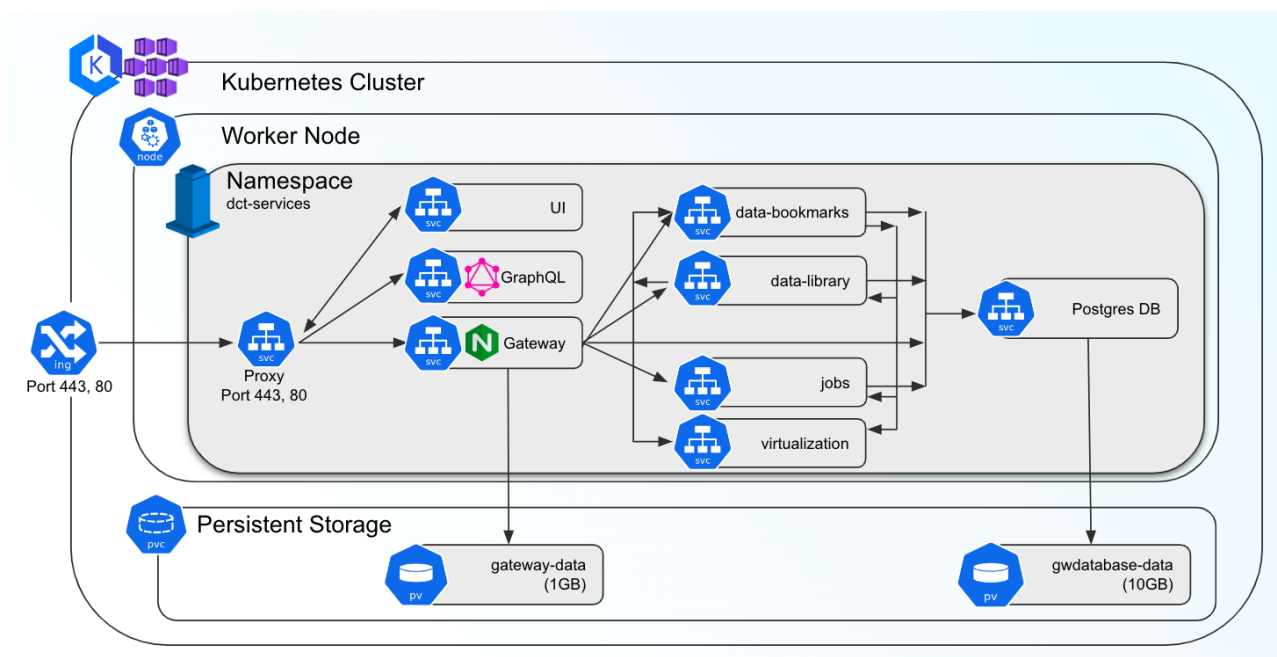
DCT is **multi-cloud enabled**, which means that a single DCT instance can be deployed to orchestrate (via HTTPS) Continuous Data and Continuous Compliance workloads with Delphix Engines located in other networks. Alternatively, DCT can be localized to engines located within a network. DCT is a lightweight management application, which means that it does not require a highly performant connection to complete its work, and can serve as a central management layer for Delphix Engines globally.
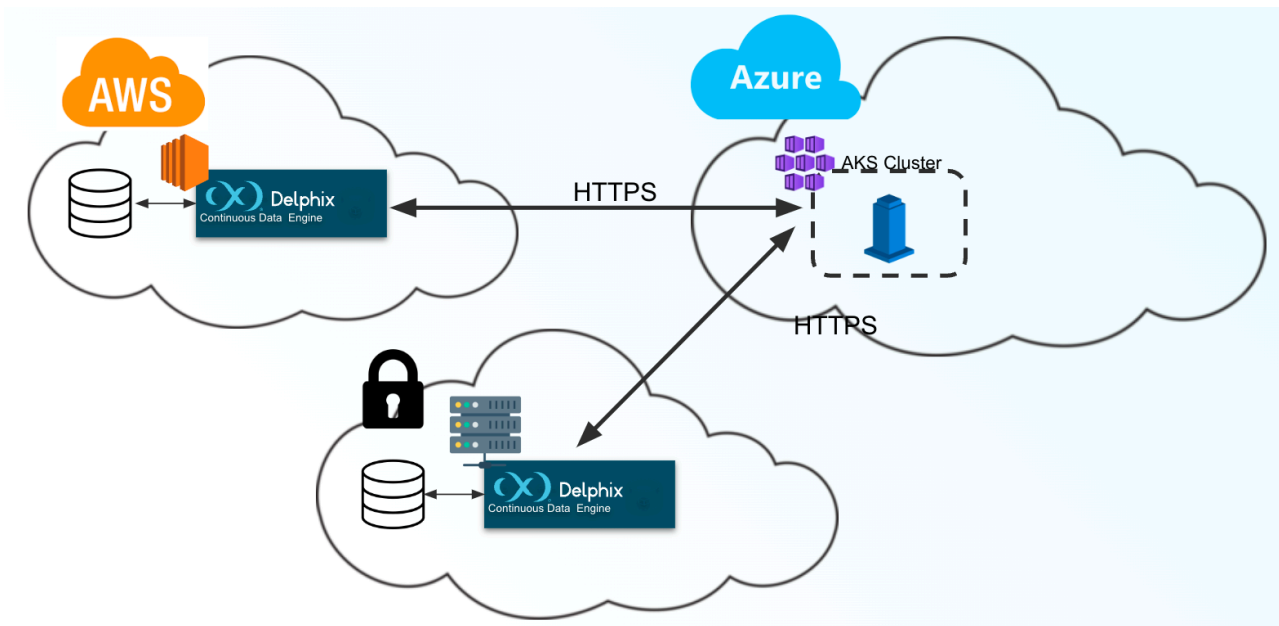
6.1 Kubernetes

6.1.1 Installation and setup for Kubernetes

> Before getting started, Delphix recommends engaging your Kubernetes Admin for deployment guidance on available Kubernetes deployment platforms (AKS, EKS, etc.). Configurations like node sizing and persistent volume settings will need to be determined.

6.1.1.1 Hardware requirements

The hardware requirements for Data Control Tower (DCT) on Kubernetes are listed below. In addition to these requirements, **inbound port 443** must be open for API clients, and **outbound port 443** to engines. This is the minimum total resource request for the Kubernetes deployment of DCT. Individual service-level resource requests are contained in the **values.yaml** file and can be overridden during deployment.

- **CPU**: 4-Core

- **CPU architecture**: x86_64
- **Memory**: 16GB
- **Storage**: 50GB
- **Port**: 443

The recommended minimum 50 GB of storage is shared across the Kubernetes cluster (i.e. hosts). All pods and/or services use this storage for mounted volumes and other utilities, including image storage.

In a single node cluster, if shared volumes are not externalized, the host requires the full 50 GB of storage. If the persistent volume is mounted externally, the host requires 37 GB of storage, since the default storage required by the database (10 GB), gateway (1 GB), masking (1 GB), and virtualization-app (1 GB) draws from the external storage. The default storage configuration for the database, gateway, masking, and virtualization-app can be modified in values.yaml.

> Many users may have default container settings as part of their Kubernetes or OpenShift infrastructure management. It is paramount to compare those default settings with the recommended minimum performance specifications (please engage your container infrastructure team to verify). If those default setting are lower, please update them to the minimum or higher.
>
> For users who need to have limits set, you can start with the following sample configuration.

```
dataBookmarks:
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
    limits:
      memory: "512Mi"
      cpu: "300m"
dataLibrary:
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
    limits:
      memory: "1024Mi"
      cpu: "800m"
database:
  resources:
    requests:
      memory: "256Mi"
      cpu: "200m"
    limits:
      memory: "4096Mi"
      cpu: "1100m"
gateway:
  resources:
```

```
      requests:
        memory: "512Mi"
        cpu: "400m"
      limits:
        memory: "2048Mi"
        cpu: "400m"
graphql:
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
    limits:
      memory: "1024Mi"
      cpu: "200m"
ui:
  resources:
    requests:
      memory: "128Mi"
      cpu: "100m"
    limits:
      memory: "256Mi"
      cpu: "200m"
jobs:
  resources:
    requests:
      memory: "256Mi"
      cpu: "200m"
    limits:
      memory: "1024Mi"
      cpu: "400m"
jobsCleaner:
  resources:
    requests:
      memory: "256Mi"
      cpu: "200m"
    limits:
      memory: "512Mi"
      cpu: "400m"
masking:
  resources:
    requests:
      memory: "512Mi"
      cpu: "500m"
    limits:
      memory: "2304Mi"
      cpu: "700m"
virtualizationApp:
  resources:
    requests:
      memory: "512Mi"
      cpu: "600m"
    limits:
```

```
      memory: "2560Mi"
      cpu: "700m"
virtualization:
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
    limits:
      memory: "512Mi"
      cpu: "300m"
```

6.1.1.2 Kubernetes overview

Data Control Tower can be deployed in a matter of minutes, once a Kubernetes cluster has been identified and deployment details have been aligned with your Kubernetes administrator. The installation consists of three components:

- **Kubernetes cluster**: The identified infrastructure to which DCT will be deployed.
- **HELM**: This deploys DCT as a Kubernetes application by referencing HELM charts (.yaml files) that make up the DCT install, either by an external Helm repository (https://dlpx-helm-dct.s3.amazonaws.com[38], this is the quickest path to installing DCT, as it largely automated) or via local install (this is accomplished by downloading the Helm charts directly via the DCT .tar file on download.delphix.com[39]).
- `kubectl` : Is a command line tool that enables administrative communication with the deployed pods (most useful post-deployment or after an upgrade).

> DCT is designed to run and is supported on any Certified Kubernetes platform[40] that supports Helm[41]. See the full statement in the Getting started[42] page.

6.1.1.3 Installation requirements (Kubernetes)

DCT requires a running Kubernetes cluster; This could be an on-premises cluster, Azure AKS or AWS EKS cluster. DCT also requires a `kubectl` command line tool to interact with Kubernetes cluster and HELM for deployment on to the cluster.

---

38 https://dlpx-helm-dct.s3.amazonaws.com/
39 http://download.delphix.com
40 https://www.cncf.io/certification/software-conformance
41 https://helm.sh/docs/topics/kubernetes_distros/
42 https://dct.delphix.com/docs/latest/getting-started#id-(17.0.0)Gettingstarted-Containerplatformsupport

Requirement	DCT recommended version	Comments
Kubernetes Cluster	1.25 or above	
HELM	3.9.0 or above	Install HELM as the package manager using the HELM installation[43] article.
kubectl	1.25.0 or above	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. To install kubectl follow the instructions at https://kubernetes.io/docs/tasks/tools/ .

6.1.1.4 Installing DCT from external HELM repository

6.1.1.4.1 Network requirements

The machine on which them *helm* commands will be executed must be able to connect to `https://dlpx-helm-dct.s3.amazonaws.com` (TCP on port 443). Otherwise, see the Offline mode[44] below.

6.1.1.4.2 Process

Add the DCT HELM repo with the following, which will link the DCT HELM repo to the local client HELM repo:

```
helm repo add dct-services https://dlpx-helm-dct.s3.amazonaws.com
```

Update the added repos with the following, which will update the local HELM charts metadata (index.yaml).

```
helm repo update
```

Pull the helm charts with the following, which is used to download the .tgz file.

---

43 https://helm.sh/docs/intro/install/
44 https://delphixdocs.atlassian.net/wiki/spaces/DCT/pages/edit-v2/688688#OFFLINE-MODE

```
helm pull dct-services/delphix-dct --version x.0.0
```

The downloaded file is then extracted using the following command (where `x.0.0` should be changed to the version of DCT being installed):

```
tar -xvf delphix-dct-x.0.0.tgz
```

Update the following properties in **values.yaml** which is present in the extracted folder `delphix-dct`.

- To generate the bootstrap APIKey, set `apiKeyCreate: true`.
- Provide image credentials to pull images from docker registry.
    - `username: <retrieved from download.delphix.com>`
    - `password: <retrieved from download.delphix.com>`

For getting the image credentials, visit the [Delphix DCT Download](#)[45] page and login with your customer login credentials. Once logged in, select the **DCT Helm Repository** link and accept the Terms and Conditions. Use password from the popup screen, as shown below.

Helm Credentials: DCT Helm Repository ✕

These credentials are linked to your organization, rather than your individual Delphix Support account, and will not expire, unless unused for an extended period of time.

Helm URL	https://dlpx-helm-dct.s3.amazonaws.com/
Username	BDOAXpNq-y4M
Password	01e89511-f200-40a0-8680-96a258cf64f3
Organization	Delphix

Close

OFFLINE MODE

Offline mode is supported for scenarios including:

- Clusters that are air-gapped and cannot access the Delphix Docker registry.

---

45 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

- Instances where Delphix Docker images must be pushed to an organization's internal registry for security scans or to comply with other security policies.

For these scenarios, a jumpbox machine with access to: Delphix Docker registry or download site[46] and the organization's internal registry is required. This jumpbox must also meet the installation prerequisites.

1. Get the Docker images

The two ways to get the Delphix Docker images are:

- Pull the images from Delphix Docker registry.

- Download and load the images from tarball.

2. Pull the images from Delphix Docker registry

Network requirements: Pulling images from the Delphix Docker registry requires the jumpbox machine to have access to `https://dct.download.delphix.com` and `https://prod-us-west-2-starport-layer-bucket.s3.us-west-2.amazonaws.com` (TCP on port 443). If a firewall is blocking access to either address, see Download and load the images from tarball[47] below.

- Login into the jumpbox machine.

- Pull the Docker images from the Delphix Docker registry.

```
$ docker pull dct.download.delphix.com/delphix-dct:nginx-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:app-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:data-bookmarks-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:delphix-data-
library-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:graphql-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:ui-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:jobs-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:postgres-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:virtualization-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:masking-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:virtualization-app-
x.0.0
```

- Re-tag all above Docker images. Below is a sample re-tag example for one image:

```
$ docker tag dct.download.delphix.com/delphix-dct:nginx-x.0.0 <local
registry url>:nginx-x.0.0
```

Note: Re-tag all remaining DCT images in the same format.

---

46 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT
47 https://delphixdocs.atlassian.net/wiki/spaces/DCT/pages/edit-v2/688688#3.-Download-and-load-the-images-from-tarball

- Push re-tagged images into the internal registry. Below is a ample push example for one image:

```
$ docker push <local registry url>:nginx-x.0.0
```

3. Download and load the images from tarball

If you do not have direct access to Delphix HELM and Docker repositories, download the HELM charts and Docker images packages as delphix-dct-x.0.0.tar.gz, which is uploaded on the download site[48].

- Login into the jumpbox machine.

- Download the HELM chart and Docker images packages as delphix-dct-x.0.0.tar.gz, which is uploaded on the download site[49].

- Extract the downloaded .tar.gz file to get the HELM chart and Docker images.

- Load extracted Docker images using following:

```
for image in *tar; do sudo docker load --input $image; done
```

- Re-tag all above Docker images. Sample re-tag example for one image:

```
$ docker tag registry.delphix.com/delphix-dct:nginx-x.0.0 <local
registry url>:nginx-x.0.0
```

Note: Re-tag all remaining DCT images in the same format.

- Push re-tagged images into internal registry. Sample push example for one image:

```
$ docker push <local registry url>:nginx-x.0.0
```

4. Deploy

- Update values.yaml for registry name and image credentials:

    a. `registry: <local registry url>`

    b. `username: <local registry username>`

    c. `password: <local registry password>`

- Install.

    a. Follow the instructions below.

---

48 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT
49 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

In version 8.0.0 and above, instead of username/password, an option is available to use credentials from a pre-existing Kubernetes Secret[50]. To do so, instead of providing a username/password, users must create the Kubernetes Secret in the same namespace as the one used for DCT, and reference the registryKey as follows:

```
imageCredentials:
  # registry to pull docker images from.
  registry: dct.download.delphix.com/delphix-dct
  # username to login to docker registry. Do not set if registryKey is set.
  username:
  # password to login to docker registry. Do not set if registryKey is set.
  password:
  # Name of an existing docker registry key to use to pull images.
  registryKey: <insert-secret-name-here>
```

After updating the values.yaml, install it using the following command:

```
helm install dct-services delphix-dct
```

> **delphix-dct** is the name of the folder which was extracted in the previous step. dct-services is the chart name which is given for this deployment.
>
> In the above directory structure, the **values.yaml** file contains the properties which are configured above. If you want to configure additional properties in values.yaml, create a values.yaml using the sample below. Deploy DCT using following command to use the custom values.yaml:
>
> ```
> helm install -f <path to edited values.yaml> dct-services <directory
> path of the extracted chart>
> ```

6.1.1.5 Sample values.yaml file

A sample values.yaml file can be downloaded below.

---

[50] https://kubernetes.io/docs/concepts/configuration/secret/

Once deployment is complete, check the status of the deployment using the following command (where `X.0.0` should be changed to the version of DCT being installed):

```
helm list
NAME               NAMESPACE    REVISION    UPDATED
STATUS      CHART                    APP VERSION
dct-services       default      1           2023-01-10 19:33:41.713202 -0900
deployed    delphix-dct-x.0.0        x.0.0
```

> 🗎 Assuming an ingress controller configuration on the Kubernetes cluster is present, when accessing DCT 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).

6.1.2 Ingress setup

Ingress exposes HTTP and HTTPS routes from outside the cluster toDCT running within the cluster. Find out more about Ingress from their official documentation[51].

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

---

51 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.27/#ingress-v1-networking-k8s-io

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

6.1.2.1 Expose proxy HTTP port (80) for non-encrypted traffic

After setting up an Ingress, TLS will be terminated by the HTTP server/load balancer/proxy implementing the Ingress, and not DCT. First, disable the TLS (SSL) configuration of DCT itself, making it expose port 80 for non encrypted traffic. To do so, edit the **values.yaml** to unset the `useSSL` property.

Either expose proxy on SSL port or non SSL port:

```
useSSL: false
```

Then run `helm upgrade` to apply the changes:

```
helm upgrade dct-services -f <path to edited values.yaml> <directory path of the
extracted chart>
```

Now, the proxy pod accepts unencrypted traffic on port 80.

6.1.2.2 Ingress controller installation and route creation

An [Ingress controller](#)[52] is required to continue. Expand a section below based on your Kubernetes environment to show the corresponding **Ingress controller installation** and **Ingress route creation** instructions.

Microsoft Azure AKS

6.1.2.2.1 Ingress controller installation

Please follow these [instructions](#)[53] to install an Nginx Ingress controller. A simple setup can be installed with these commands:

```
NAMESPACE=ingress-basic
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm install ingress-nginx ingress-nginx/ingress-nginx \
  --create-namespace \
  --namespace $NAMESPACE \
  --set controller.service.annotations."service\.beta\.kubernetes\.io/azure-load-
balancer-health-probe-request-path"=/healthz
```

---

52 https://kubernetes.io/docs/concepts/services-networking/ingress-controllers
53 https://learn.microsoft.com/en-us/azure/aks/ingress-basic

6.1.2.2.2 Ingress route creation

Create a file named **ingress.yaml**.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dct-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: nginx
  rules:
    - http:
        paths:
        - path: /
          pathType: Prefix
          backend:
          service:
              name: proxy
              port:
                  number: 80
```

Apply the Ingress resource with `kubectl apply` :

```
kubectl apply -f ingress.yaml --namespace=ingress-basic
```

To configure TLS, see [Use TLS with an Ingress controller](#)[54].

Amazon AWS EKS

6.1.2.2.3 Ingress controller installation

Please follow these [instructions](#)[55] to install an [AWS load balancer controller](#)[56] (An Ingress controller which configures AWS application load balancers).

6.1.2.2.4 Ingress route creation

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

---

54 https://learn.microsoft.com/en-us/azure/aks/ingress-tls
55 https://docs.aws.amazon.com/eks/latest/userguide/alb-ingress.html
56 https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html

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

Alternatively, you may use certificate discovery[57] to have the ALB select a matching certificate from AWS Certificate manager[58] based on the host name.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dct-ingress
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internal
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/ssl-redirect: '443'
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}, {"HTTPS":443}]'
spec:
  tls:
  - hosts:
    - www.example.com
  rules:
    - http:
        paths:
         - path: /
           pathType: Prefix
           backend:
```

---

57 https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.5/guide/ingress/cert_discovery/
58 https://aws.amazon.com/certificate-manager/

```
            service:
              name: proxy
              port:
                number: 80
```

Apply the Ingress resource with `kubectl apply` :

```
  kubectl apply -f ingress.yaml --namespace=ingress-basic
```

This creates an [application load balancer](#)[59], which forwards all traffic to DCT.

Other

6.1.2.2.5 Ingress controller installation

For **self-hosted Kubernetes**, you can install an Nginx Ingress controller with:

```
  helm upgrade --install ingress-nginx ingress-nginx --repo https://
  kubernetes.github.io/ingress-nginx --namespace ingress-nginx --create-namespace
```

Reference [environment specific instructions](#)[60] for a complete list of supported platforms and specific instructions.

6.1.2.2.6 Ingress route creation

Create a file name **ingress.yaml**.

```
  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    name: dct-ingress
    annotations:
      nginx.ingress.kubernetes.io/ssl-redirect: "true"
  spec:
    ingressClassName: nginx
    rules:
      - http:
          paths:
            - path: /
              pathType: Prefix
              backend:
                service:
                  name: proxy
                  port:
```

---

59 https://docs.aws.amazon.com/eks/latest/userguide/alb-ingress.html
60 https://kubernetes.github.io/ingress-nginx/deploy/

```
            number: 80
```

Apply the Ingress resource with `kubectl apply` :

```
kubectl apply -f ingress.yaml --namespace=ingress-basic
```

Review the Ingress-Nginx instructions[61] to setup TLS.

6.1.3 Bootstrapping API keys

API keys are the default method used to authenticate with DCT, which is done by including the API key in the HTTP Authorization request header[62] with the `apk` type.

> ℹ️ API keys are long-live tokens and as a result, do not automatically expire in the future. They remain valid until they are deleted or destroyed from DCT. Rotating API keys periodically is strongly recommended to avoid security risk.

Client URL (cURL) is a CLI tool for transferring data via URLs, which is how the authorization to DCT can pass. A demonstration cURL authorization with an example API key would appear as follows.

API key

```
1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3
```

Command

```
curl --header 'Authorization: apk
1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3'
```

cURL, similar to web browsers and other HTTP clients, will not connect to DCT over HTTPS unless a valid TLS certification has been configured for the Nginx server.

> ℹ️ cURL must be on version 7.43 or higher.

---

61 https://kubernetes.github.io/ingress-nginx/user-guide/tls/
62 https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization

6.1.3.1 Bootstrap first API key

The recommended process is that you only use the first (bootstrapped) API key to create another API key, then delete the first key promptly afterwards. This process is best for enhanced security, because the bootstrap key is more broadly accessible during setup, as it might be hardcoded in scripts, configuration files, or recorded in logs.

> ℹ️ This process can be repeated as many times as needed, for example, in a case where existing API keys are lost or have been deleted.

Once the DCT application is started, edit the **values.yaml** file and modify the `apiKeyCreate` property to have a string value of `true`.

```
apiKeyCreate: true
```

Upgrade DCT with:

```
helm upgrade dct-services <directory path of extracted chart>
```

If the values.yaml file needs to be overridden from outside, use:

```
helm upgrade dct-services -f <path_to_values.yaml> <directory path of extracted chart>
```

A similar output should appear in the `gateway` pod logs with the newly generated key (please note, this is an example key):

```
NEWLY GENERATED API KEY: 1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3
```

The `gateway-pod-name` will appear in the format of `gateway-xxx` and can be found using:

```
kubectl get pods -n dct-services
```

The logs for the `gateway` pod can be accessed using:

```
kubectl logs <gateway-pod-name> -n dct-services
```

The newly generated key can now be used to authenticate with DCT, as exemplified in the first section of this page. Remember to prefix the API key value with `apk`, as follows:

```
curl --header 'Authorization: apk
1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3'
```

Edit the values.yaml and set the `apiKeyCreate` property back to a string value of `false` and upgrade DCT again with the same command used above.

```
helm upgrade dct-services <directory path of extracted chart>
```

Like before, if the values.yaml file needs to be overridden from outside, use:

```
helm upgrade dct-services -f <path_to_values.yaml> <directory path of extracted chart>
```

6.1.3.2 Create and manage API keys

As recommended above, the first API key should be used to create a new, secure admin key. To do this, create a new Account entity and set the `generate_api_key` property to `true`. The `username` value should be the desired name to uniquely identify the account.

> ℹ️ If the cURL version is below 7.43, replace `--data-raw` with `--data` in the following.

```
curl --location --request POST 'https://<hostname>/v2/management/accounts' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk
1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3' \
--data-raw '{
 "username": "secure-key",
 "generate_api_key": true,
 "is_admin": true
}'
```

A similar response should be received as shown below:

```
{
 "id": 2,
 "token": "2.vCfC0MnpySYZLshuxap2aZ7xqBKAnQvV7hFnobe7xuNlHS9AF2NQnV9XXw4UyET6"
```

```
    "username":"secure-key"
  }
```

Now that the new, secure API key has been created, the old one must be deleted for the security purposes outlined in the first section of this page. To do this, make the following request:

```
curl --location --request DELETE 'https://<hostname>/v2/management/api-clients/<id>'
 \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk
2.vCfC0MnpySYZLshuxap2aZ7xqBKAnQvV7hFnobe7xuNlHS9AF2NQnV9XXw4UyET6'
```

The `<id>` referenced above is the numeric ID of the Account entity—the integer before the period in the token. For example, the `id` of `1.0p9PMkZ04Hgy0ezwjhX...` is `1`.

To list all of the current Accounts, use:

```
curl --location --request GET 'https://<hostname>/v2/management/accounts/' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>'
```

6.1.4 DCT logs for Kubernetes

All DCT containers log to stdout and stderr so that their logs are processed by Kubernetes. To view container level logs running on the Kubernetes cluster use:

```
kubectl logs <pod_name> -n dct-services
```

Log aggregators can be configured to read from `stdout` and `stderr` for all of the pods as per the requirements.

6.1.5 Admin topics

- Deployment upgrade for Kubernetes
- Factory reset DCT for Kubernetes

6.1.5.1 Deployment upgrade for Kubernetes

This page covers the upgrade process for DCT deployments on Kubernetes.

In DCT 12.0.0 and above, one new service (container) has been introduced named **Masking**.

The memory and CPU requirement for this newly added service (container) is 512Mi and 500m, respectively.

This container creates a new persistent volume of 1GB dynamically. If you are using pre-existing static persistent volumes for other existing pods (i.e. gateway and database), similarly to these pods, create a new persistent volume claim for the masking pod, then update the `maskingPvcName` property in the values.yaml with the name of static persistent volume claim.

For customizing the **masking** service resources (i.e. requests and limits) and properties, please refer to the values.yaml file in the Helm chart of the current release.

Starting from DCT 15.0.0 we have introduced one new service (container) named **virtualization-app**.

The memory and CPU requirement for this newly added service (container) is 512Mi and 600m, respectively.

This container creates a new persistent volume of 1GB dynamically. If you are using pre-existing static persistent volumes for other existing pods (i.e. gateway and database), similarly to these pods, create a new persistent volume claim for the virtualization-app pod, then update the `virtualizationAppPvcName` property in the values.yaml with the name of the static persistent volume claim.

For customizing **virtualization-app** service resources (i.e. request and limits) and properties, please refer values.yaml file in the Helm chart of thre current release.

Create a new folder called **dct-x.0.0**, where `x.0.0` should be changed to the version of DCT being installed (e.g. if on 5.0.2, it would be 6.0.0).

```
mkdir dct-x.0.0
```

Update the added repos with the following, which will update the local HELM charts metadata (index.yaml).

```
helm repo update
```

Pull the HELM charts with the following:

> 🗐 This command will download a file named **delphix-dct-x.0.0.tgz** in the folder dct-x.0.0.

```
cd dct-x.0.0
helm pull dct-services/delphix-dct --version x.0.0
```

The downloaded file is then extracted using the following command (where `x.0.0` should be changed to the version of DCT being installed):

```
tar -xvf delphix-dct-x.0.0.tgz
```

Which will extract into the following directory structure:

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

Create a new values.yaml inside to the dct-x.0.0 folder (e.g. parallel to delphix-dct folder), use sample values.yaml given below. Copy the configured properties from previous version to this newly created values.yaml[63].

> 🗐 This values.yaml file should only contain the modified values from the previous version of deployment and not the entire values.yaml file as it is.

Update the username and password in values.yaml. It can be obtained from https://download.delphix.com[64]. Here are some notes in regards to this step in the process:

- This username and password update in values.yaml is only required if the user using DCT provided a Docker Registry directly in the deployment (i.e. values.yaml).

> 🗐 **OFFLINE MODE**
>
> Offline mode is supported for scenarios including:

- Clusters that are air-gapped and cannot access the Delphix Docker registry.

- Instances where Delphix Docker images must be pushed to an organization's internal registry for security scans or to comply with other security policies.

For these scenarios, a jumpbox machine with access to both the Delphix Docker registry or download site[65] and the organization's internal registry is required. This jumpbox must also meet the installation prerequisites.

1. Get the Docker images

The two ways to get the Delphix Docker images are:

- Pull the images from Delphix Docker registry.

- Download and load the images from tarball.

2. Pull the images from Delphix Docker registry

- Login into the jumpbox machine.

- Pull the Docker images from the Delphix Docker registry.

```
$ docker pull dct.download.delphix.com/delphix-dct:nginx-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:app-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:data-bookmarks-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:delphix-data-
library-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:graphql-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:ui-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:jobs-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:postgres-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:virtualization-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:masking-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:virtualization-app-
x.0.0
```

- Re-tag all above Docker images. Below is a sample re-tag example for one image:

```
$ docker tag dct.download.delphix.com/delphix-dct:nginx-x.0.0 <local
registry url>:nginx-x.0.0
```

Note: Re-tag all remaining DCT images in the same format.

- Push re-tagged images into the internal registry. Below is a sample push example for one image:

```
$ docker push <local registry url>:nginx-x.0.0
```

---

[65] https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

3. Download and load the images from tarball

If you do not have direct access to Delphix HELM and Docker repositories, download the HELM charts and Docker images packages as [delphix-dct-x.0.0.tar.gz](#)[66], which is uploaded on the [download site](#)[67].

- Login into the jumpbox machine.

- Download the HELM chart and Docker images packages as [delphix-dct-x.0.0.tar.gz](#)[68], which is uploaded on the [download site](#)[69].

- Extract the downloaded .tar.gz file to get the HELM chart and Docker images.

- Load extracted Docker images using following:

```
for image in *tar; do sudo docker load --input $image; done
```

- Re-tag all above Docker images. Sample re-tag example for one image:

```
$ docker tag registry.delphix.com/delphix-dct:nginx-x.0.0 <local
registry url>:nginx-x.0.0
```

Note: Re-tag all remaining DCT images in the same format.

- Push re-tagged images into internal registry. Sample push example for one image:

```
$ docker push <local registry url>:nginx-x.0.0
```

4. Deploy

- Update values.yaml for registry name and image credentials:

    a. `registry: <local registry url>`

    b. `username: <local registry username>`

    c. `password: <local registry password>`

- Install.

    a. Follow the instructions below.

- Run the HELM upgrade command:

---

66 https://download.delphix.com/file/12747/delphix-dct-9.0.0.tar.gz
67 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT
68 https://download.delphix.com/file/12747/delphix-dct-9.0.0.tar.gz
69 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

```
helm upgrade -f values.yaml dct-services delphix-dct
```

6.1.5.2 Factory reset DCT for Kubernetes

To clean DCT installation run following command:

```
helm delete dct-services
```

> ⚠️ This process will delete services pod and database both.

6.1.6 MicroK8s (Kubernetes)

This section describes how to deploy DCT in Micro-Kubernetes.

6.1.6.1 MicroK8s overview

6.1.6.1.1 Introduction

As the deployment and management of multi-container applications have become prominent for achieving scalability and reliability, Delphix introduced Kubernetes support for DCT in the past. Kubernetes (K8s) is an open-source platform tailored toward automatic management, scaling, and maintenance of containerized workloads. Data Control Tower (DCT) with Kubernetes, and now MicroK8s, work to streamline deployments and minimize operational overhead for users.

MicroK8s is a solution for those seeking the benefits of Kubernetes, without the complexity and resource requirements of a full-scale deployment. MicroK8s is a single-package, fully conformant Kubernetes distribution that is both lightweight and production-ready. Designed to run on Linux, Windows, and MacOS, MicroK8s offers a seamless experience across platforms.

> ℹ️ DCT is designed to run and is supported on any Certified Kubernetes platform[70] that supports Helm[71]. See the full statement in the Getting started[72] page.

6.1.6.1.2 Transition from Docker Compose

Transitioning from Docker Compose to MicroK8s for DCT would leverage Kubernetes' robust features while still maintaining simplicity and minimizing operational overhead, providing the benefit of scalability, reliability, and agility offered by containerized deployments.

For more information in regards to MicroK8s, visit the MicroK8s documentation[73].

6.1.6.2 MicroK8s requirements and prerequisites

Deploying Data Control Tower (DCT) on MicroK8s requires consideration of both hardware and software prerequisites to ensure a smooth and efficient setup. This page outlines the essential requirements and preparatory steps needed to deploy DCT on MicroK8s successfully.

6.1.6.2.1 Hardware requirements

To deploy DCT on MicroK8s, these hardware specifications must be met to handle the computational and storage demands of the application. The minimum hardware requirements include:

- **CPU**
    - 4-Core Processor
- **CPU architecture**
    - x86_64, ensuring compatibility with the majority of modern computing environments.
- **Memory**
    - 16GB RAM, to adequately support the operating system, MicroK8s, and DCT components.
- **Storage**
    - 50GB of available disk space, to store the MicroK8s components, DCT application files, and any data processed or managed by DCT.
- **Network ports**
    - Inbound port 443 must be open for API clients, facilitating secure HTTPS connections to the DCT services.
    - Outbound port 443 should be open to engines, allowing DCT to communicate with external services and repositories.

---

70 https://www.cncf.io/certification/software-conformance

71 https://helm.sh/docs/topics/kubernetes_distros/

72 https://dct.delphix.com/docs/latest/getting-started#id-(17.0.0)Gettingstarted-Containerplatformsupport

73 https://microk8s.io/docs/

These requirements represent the minimum configuration needed for a MicroK8s deployment of DCT. Depending on the scale of your deployment and the expected workload, additional resources may be required.

6.1.6.2.2 Software prerequisites

Before proceeding with the installation of MicroK8s, ensure that the following software prerequisites are met:

- **Operating system:** Linux, Windows, or MacOS, compatible with MicroK8s. Each platform has specific instructions for installing and configuring MicroK8s, which must be followed to ensure compatibility.
- **Snap or Homebrew:**
  - For Linux installations, `snap` is required to install MicroK8s. If `snap` is not already installed, it must be added to the system.
  - MacOS users will need `brew` (Homebrew) to install MicroK8s.

6.1.6.2.3 Network configuration

- **Open ports:** Verify that the required network ports (443 inbound and outbound) are open and not blocked by firewalls or network policies.
- **DNS and ingress:** Configuring DNS and ingress correctly is vital for routing traffic to your DCT deployment. MicroK8s includes an `enable dns` and `enable ingress` command to simplify this process.

6.1.6.3 MicroK8s installation and setup

Deploying Data Control Tower (DCT) on MicroK8s begins with the proper installation and setup of MicroK8s. This section provides detailed steps for installing MicroK8s on Linux (Ubuntu, CentOS, and Red Hat), MacOS, and Windows platforms, along with initial configuration steps to prepare your environment for DCT deployment.

To get started, please visit one of the corresponding pages:

- MicroK8s on Linux (online mode)

6.1.6.3.1 MicroK8s on Linux (online mode)

Installation and setup for MicroK8s on Linux distributions.

6.1.6.3.1.1 Ubuntu

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

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

    a. If snap is not installed on your system, follow the instructions here[74] to install it.

2. **Join the MicroK8s group:**
MicroK8s creates a user group to enable seamless command execution. Add your user to this group and set the correct permissions for the `.kube` caching directory with:

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

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

3.
```
su - $USER
```

4. **Check MicroK8s status:**
Ensure MicroK8s is correctly installed and ready by checking its status:

```
microk8s status --wait-ready
```

5. **Enable add-ons:**
Enable essential add-ons for DCT deployment:

```
microk8s enable hostpath-storage
microk8s enable helm
microk8s enable dns
microk8s enable ingress
```

6. **Create an alias for** `kubectl` **and** `helm`
Facilitate command usage with an alias for `microk8s kubectl` and `microk8s helm`

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

6.1.6.3.1.2 CentOS and Red Hat

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

---

74 https://snapcraft.io/docs/installing-snapd

1. **Enable** `snapd` :

   First, enable the EPEL repository and install `snapd` :

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

   a. Then, start and enable `snapd` :

2.
   ```
   sudo systemctl enable --now snapd.socket
   ```

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

3.
   ```
   sudo ln -s /var/lib/snapd/snap /snap
   ```

4. **Install MicroK8s:**

   With `snapd` enabled, you can now install MicroK8s using `snap` :

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

5. **Group and permissions:**
   Follow the same steps as for Ubuntu to add your user to the MicroK8s group and adjust permissions for the `.kube` directory.

6. **Check status and enable add-ons:**
   Verify MicroK8s installation and enable the necessary add-ons as outlined in the Ubuntu section.

7. **Alias for** `kubectl` **and** `helm`
   Create and source the alias for kubectl and helm as described for Ubuntu.

6.1.6.4 Deploying DCT on MicroK8s

This page provides steps for deploying Data Control Tower (DCT) on MicroK8s using Helm charts. The process involves adding the DCT Helm repository, configuring the deployment through `values.yaml` , and handling deployments in both online and offline environments.

6.1.6.4.1 Adding the DCT Helm repository

This command registers the DCT services repository with your Helm client, making the charts available for your deployment.

```
helm repo add dct-services https://dlpx-helm-dct.s3.amazonaws.com
```

6.1.6.4.2 Updating Helm repositories

After adding the DCT repository, update your local Helm charts metadata to ensure you have the latest versions available:

```
helm repo update
```

6.1.6.4.3 Downloading and preparing the DCT Helm chart

Pull the required version of the Helm chart for DCT using the Helm pull command. Replace `x.0.0` with the specific version of DCT you intend to install:

```
helm pull dct-services/delphix-dct --version x.0.0
```

Extract the downloaded chart with the following command:

```
tar -xvf delphix-dct-x.0.0.tgz
```

6.1.6.4.4 Configuring the deployment

Navigate to the extracted folder and update the `values.yaml` file with necessary configurations:

- To automatically generate a bootstrap API key, set `apiKeyCreate: true`.
- Provide the Docker registry credentials (username and password) for pulling images. These credentials can be retrieved from the Delphix DCT Download page by logging in with your user login.

6.1.6.4.5 Deploying DCT

With the `values.yaml` file configured, deploy DCT using Helm with the following command:

```
helm install dct-services delphix-dct
```

6.1.6.4.6 Handling offline mode

For environments without direct access to the Delphix Helm and Docker repositories:

1. Download the `delphix-dct-x.0.0.tar.gz` package from the download site.

2. Extract the Helm chart and Docker images from the downloaded file.

3. Load the Docker images into your local Docker environment using:

```
for image in *tar; do sudo docker load --input $image; done
```

4. Re-tag and push the Docker images to your local registry. Follow the format for tagging images:

```
docker tag registry.delphix.com/delphix-dct:app-x.0.0 <local registry url>:app-x.0.0
```

5. Update the `values.yaml` with your local registry name and credentials.

6.1.6.4.7 Utilizing MicroK8s' built-in registry for DCT Docker images

MicroK8s includes a built-in Docker registry that allows you to store Docker images directly on your MicroK8s instance. This is particularly useful for single VM solutions, providing a more integrated and straightforward approach to managing Docker images for DCT deployment.

6.1.6.4.7.1 Process

1. MicroK8s comes with an add-on registry that can be enabled with:

```
microk8s enable registry
```

    a. This command sets up a Docker registry within MicroK8s, accessible on port 32000.

2. After loading your DCT Docker images into MicroK8s, they must be re-tagged to use them in the local MicroK8s registry. Use the following format for re-tagging your images:

```
docker tag registry.delphix.com/delphix-dct:app-x.0.0 localhost:32000/delphix-dct:app-x.0.0
```

3. Once re-tagged, push the images to the MicroK8s registry by running:

```
docker push localhost:32000/delphix-dct:app-x.0.0
```

    a. This command uploads the re-tagged DCT images to the built-in Docker registry of your MicroK8s instance.

4. With the images now hosted in the MicroK8s registry, update the `values.yaml` file to reference the local images. Change the image repository to use the MicroK8s registry URL:

```
registry: localhost:32000
```

This will have Helm pull the Docker images from the MicroK8s internal registry during the deployment of DCT, to streamline deployment within a single VM solution.

6.1.6.5 Ingress route creation and accessing DCT

After successfully deploying Data Control Tower (DCT) on MicroK8s, the final step involves creating an ingress route to manage external traffic to DCT services efficiently. This page covers the process of configuring ingress within your MicroK8s environment.

There are two options two configure ingress:

1. With valid certificates(Recommended).

2. With default self sign certificate.

6.1.6.5.1 1. With valid certificates

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

6.1.6.5.1.1 Expose proxy HTTP port (80) for non-encrypted traffic

After setting up an Ingress, TLS will be terminated by the HTTP server/load balancer/proxy implementing the Ingress, and not DCT. First, disable the TLS (SSL) configuration of DCT itself, making it expose port 80 for non encrypted traffic. To do so, edit the **values.yaml** to unset the `useSSL` property.

Either expose proxy on SSL port or non SSL port:

```
useSSL: false
```

Then run `helm upgrade` to apply the changes:

```
helm upgrade dct-services -f <path to edited values.yaml> <directory path of the
extracted chart>
```

Now, the proxy pod accepts unencrypted traffic on port 80.

6.1.6.5.1.2 Create Secret with SSL certificate.

This section explains the process of adding the certificates to a kubernets secret, which is then used while creating the ingress.

These follow points are required:

- Administrative access to the MicroK8s cluster.

- The new SSL certificate(public key) and private key files ready for deployment for e.g server.crt and server.key.

```
kubectl create secret tls ingress-tls --namespace dct-services --key <private key> --cert <public certificate>
```

6.1.6.5.1.3 Creating the Ingress Class

Begin by defining an Ingress Class, which specifies the Ingress Controller that will manage the ingress resources. Create a file named `ingressClass.yaml` with the following content:

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: nginx
  namespace: dct-services
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: k8s.io/ingress-nginx
```

This configuration sets up an Nginx Ingress Controller as the default class for handling ingress in the `dct-services` namespace.

6.1.6.5.1.4 Configuring the ingress resource

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

Note: Replace `[HOSTNAME]` with the actual hostname or IP address of your MicroK8s cluster.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dct-ingress
  namespace: dct-services
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - [HOSTNAME]
    secretName: ingress-tls
  rules:
    - host: "[HOSTNAME]"
      http:
```

```
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: proxy
              port:
                number: 80
```

This ingress configuration directs all HTTP traffic arriving at the root path (`/`) to the `proxy` service on port 80.

6.1.6.5.1.5 Applying the ingress configuration

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

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

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

6.1.6.5.2 2. With default self sign certificate

6.1.6.5.2.1 Creating the Ingress Class

Begin by defining an Ingress Class, which specifies the Ingress Controller that will manage the ingress resources. Create a file named `ingressClass.yaml` with the following content:

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: nginx
  namespace: dct-services
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: k8s.io/ingress-nginx
```

This configuration sets up an Nginx Ingress Controller as the default class for handling ingress in the `dct-services` namespace.

6.1.6.5.2.2 Configuring the ingress resource

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

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dct-ingress
  namespace: dct-services
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  ingressClassName: nginx
  rules:
    - http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: proxy
              port:
                number: 443
```

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

6.1.6.5.2.3 Applying the ingress configuration

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

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

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

6.1.6.5.2.4 Accessing DCT

Once the ingress route is successfully created and active, you can access the DCT UI and Swagger documentation through the following URLs:

- **DCT UI:** `<https://[HOSTNAME]:443 >`
- **Swagger API Documentation:** `<https://[HOSTNAME]:443/api >`

Replace `[HOSTNAME]` with the actual hostname or IP address of your MicroK8s cluster.

Note: If ingress is configured with default certificate, browser will show a warning message.

6.1.6.6 MicroK8s troubleshooting

6.1.6.6.1 Troubleshooting

Here are some common issues and solutions that could come up while implementing DCT on MicroK8s.

- **Issue:** MicroK8s services fail to start.
    - **Solution:** Ensure that all system requirements are met and that there are no conflicts with port allocations. Check the system logs for specific error messages.
- **Issue:** DCT deployment fails due to Helm chart errors.
    - **Solution:** Verify that the Helm version is compatible with the DCT Helm chart. Ensure that the `values.yaml` file is correctly configured according to the deployment needs.

6.1.6.7 MicroK8s backup and restore

Backing up and restoring DCT data on MicroK8s involves the following steps, which help ensure data integrity and availability, especially when using hostpath volume mounts.

6.1.6.7.1 Backup

1. Determine which data or directories need to be backed up. This could include database files, application state, configuration files, and any other critical data.

2. Example command to backup a DCT data directory:

```
sudo cp -r /var/snap/microk8s/common/default-storage/dct-data /path/to/backup/
location/dct-data-backup
```

   a. **For Hostpath Volumes:** Navigate to the directory where the hostpath volumes are stored on your MicroK8s node. Use standard file system operations to copy the relevant directories and files to a backup location. This could be an external storage device or a remote backup service.

6.1.6.7.2 Restore

1. Ensure the DCT services that use the data are temporarily stopped to prevent data corruption during the restore process.

2. Example command to restore DCT data from backup:

```
sudo cp -r /path/to/backup/location/dct-data-backup /var/snap/microk8s/common/
default-storage/dct-data
```

a. **For Hostpath Volumes:** Copy the backup data back into the original hostpath volume directory. Ensure that file permissions and ownership are correctly set to match the requirements of your DCT deployment.

3. Once the data is restored, restart the DCT services to make sure they are recognized and utilize the restored data.

> ℹ️ Schedule regular backups of critical data to reduce the risk of significant data loss.

6.2 OpenShift

6.2.1 Installation and setup for OpenShift

6.2.1.1 Hardware requirements

The hardware requirements for Data Control Tower to deploy on OCP are listed below. In addition to these requirements, inbound port 443 or 80 must be open for API clients. This is the minimum total resource requirement for the deployment.

- **CPU**: 4-Core
- **CPU Architecture**: x86_64
- **Memory**: 16GB
- **Storage**: 50GB
- **Port**: 443

The recommended minimum 50 GB of storage is shared across the Kubernetes cluster (i.e. hosts). All pods and/or services use this storage for mounted volumes and other utilities, including image storage.

In a single node cluster, if shared volumes are not externalized, the host requires the full 50 GB of storage. If the persistent volume is mounted externally, the host requires 37 GB of storage, since the default storage required by the database (10 GB), gateway (1 GB), masking (1 GB) and virtualization-app (1 GB) draws from the external storage. The default storage configuration for the database, gateway, masking, and virtualization-app can be modified in the values.yaml file.

Many users may have default container settings as part of their Kubernetes or OpenShift infrastructure management. It is paramount to compare those default settings with the recommended minimum performance specifications (please engage your container infrastructure team to verify). If those default setting are lower, please update them to the minimum or higher.

For users who need to have limits set, you can start with the following sample configuration.

```
dataBookmarks:
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
    limits:
      memory: "512Mi"
      cpu: "300m"
dataLibrary:
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
    limits:
      memory: "1024Mi"
      cpu: "800m"
database:
  resources:
    requests:
      memory: "256Mi"
      cpu: "200m"
    limits:
      memory: "4096Mi"
      cpu: "1100m"
gateway:
  resources:
    requests:
      memory: "512Mi"
      cpu: "400m"
    limits:
      memory: "2048Mi"
      cpu: "400m"
graphql:
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
    limits:
      memory: "1024Mi"
      cpu: "200m"
```

```
ui:
  resources:
    requests:
      memory: "128Mi"
      cpu: "100m"
    limits:
      memory: "256Mi"
      cpu: "200m"
jobs:
  resources:
    requests:
      memory: "256Mi"
      cpu: "200m"
    limits:
      memory: "1024Mi"
      cpu: "400m"
jobsCleaner:
  resources:
    requests:
      memory: "256Mi"
      cpu: "200m"
    limits:
      memory: "512Mi"
      cpu: "400m"
masking:
  resources:
    requests:
      memory: "512Mi"
      cpu: "500m"
    limits:
      memory: "2304Mi"
      cpu: "700m"
virtualizationApp:
  resources:
    requests:
      memory: "512Mi"
      cpu: "600m"
    limits:
      memory: "2560Mi"
      cpu: "700m"
virtualization:
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
    limits:
      memory: "512Mi"
      cpu: "300m"
```

6.2.1.2 Installation requirements (OpenShift)

> ℹ️ DCT is designed to run and is supported on any Certified Kubernetes platform[75] that supports Helm[76]. See the full statement in the Getting started[77] page.

DCT requires a running OpenShift cluster to run, `oc` command line tool to interact with OpenShift cluster, and HELM for deployment onto the cluster.

Requirement	DCT Recommended Version	Comments
OpenShift Cluster	4.12 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/.
oc	4.11.3 or above	To install `oc`, follow the instructions at https://docs.openshift.com/container-platform/4.16/cli_reference/openshift_cli/getting-started-cli.html.

> 📋 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 **imageCredentials** section.

---

75 https://www.cncf.io/certification/software-conformance

76 https://helm.sh/docs/topics/kubernetes_distros/

77 https://dct.delphix.com/docs/latest/getting-started#id-(17.0.0)Gettingstarted-Containerplatformsupport

6.2.1.3 Installation process

6.2.1.3.1 Jump box setup

6.2.1.3.1.1 OC login

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

```
oc login https://openshift1.example.com --token=<<token>>
```

6.2.1.3.1.2 Verify KubeConfig

HELM will use the configuration file inside the **$HOME/.kube/** folder to deploy artifacts on an OpenShift cluster.

Be sure the config file has the cluster context added, and the current-context is set to use this cluster. To verify the context, run this command:

```
oc config current-context
```

6.2.1.3.1.3 Create a new project

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

```
oc new-project dct-services --description="DCT Deployment project" --display-name="dct-services"
```

6.2.1.3.1.4 Installing Helm

Install HELM using the following installation instructions mentioned at https://helm.sh/docs/intro/install/.

6.2.1.3.2 Deploy DCT chart

Add the DCT HELM repo with the following, which will link the DCT HELM repo to the local HELM repo:

```
helm repo add dct-services https://dlpx-helm-dct.s3.amazonaws.com
```

Update the added repos with the following, which will update the local HELM charts metadata (index.yaml):

```
helm repo update
```

Pull the HELM charts with the following, which is used to download the .tgz file:

```
helm pull dct-services/delphix-dct --version x.0.0
```

The downloaded file is then extracted using the following command (where `x.0.0` should be changed to the version of DCT being installed):

```
tar -xvf delphix-dct-x.0.0.tgz
```

Update the following properties in the values.yaml file, present in the extracted `delphix-dct` folder.

- To generate the bootstrap APIKey, set `apiKeyCreate: true`.
- Provide image credentials to pull images from docker registry.
    - `username: <retrieved from download.delphix.com>`
    - `password: <retrieved from download.delphix.com>`
- `isOpenshift` should be set to `true`.

To get the image credentials, visit the Delphix DCT Download[78] page and login with your user login credentials. Once logged in, select the **DCT Helm Repository** link and accept the Terms and Conditions. Use password from the popup screen, as shown below.

Helm Credentials: DCT Helm Repository ✖

These credentials are linked to your organization, rather than your individual Delphix Support account, and will not expire, unless unused for an extended period of time.

Helm URL	https://dlpx-helm-dct.s3.amazonaws.com/	📋
Username	BDOAXpNq-y4M	📋
Password	01e89511-f200-40a0-8680-96a258cf64f3	📋
Organization	Delphix	

Close

---

[78] https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

6.2.1.4 Offline mode

Instructions for offline mode

An offline mode is supported for use cases where:

1. The cluster can be air gapped (it does not have access to the Delphix Docker registry).

2. Delphix Docker images should first be pushed to the organization's internal Artifactory or registry for scanning and to comply with any other security policies in place.

If one such use case exists, retrieve the Delphix Docker images, Helm chart, and deploy. A jump box machine is needed, which has access to both Delphix Docker registry[79]/download site[80] and the organization's internal Artifactory/registry. In addition, the jump box machine must satisfy the installation requirements.

6.2.1.4.1 Get the Docker images

There are two ways to retrieve the Delphix Docker images:

1. Pull the images from the Delphix Docker registry.

2. Download and load the images from tarball.

6.2.1.4.1.1 **1. Pulling images from Docker registry**

- Login into the jump box machine.
- Pull the Docker images from the Delphix Docker registry.

```
$ docker pull dct.download.delphix.com/delphix-dct:nginx-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:app-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:data-bookmarks-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:delphix-data-library-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:graphql-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:ui-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:jobs-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:postgres-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:virtualization-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:masking-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:virtualization-app-x.0.0
```

- Re-tag all above Docker images. This is a sample re-tag for one image:

```
$ docker tag dct.download.delphix.com/delphix-dct:nginx-x.0.0 <local registry
url>:nginx-x.0.0
```

Note: Re-tag all remaining DCT images in the same format.

---

[79] http://dct.download.delphix.com/delphix-dct
[80] https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

- Push re-tagged images into the internal registry. This is a sample push example for one image:

```
$ docker push <local registry url>:nginx-x.0.0
```

6.2.1.4.1.2 **2. Download tarball images**

If you do not have direct access to Delphix HELM and Docker repositories, download the HELM charts and Docker images packages as delphix-dct-x.0.0.tar.gz[81], which is available on the download site[82].

- Login into the jump box machine.
- Download the HELM chart and docker images packages as delphix-dct-x.0.0.tar.gz[83], which is available on the download site[84].
- Extract the downloaded.tar.gz file to get the HELM chart and Docker images.
- Load extracted Docker images using following:

```
for image in *tar; do sudo docker load --input $image; done
```

- Re-tag all above Docker images. Sample re-tag example for one image:

```
$ docker tag registry.delphix.com/delphix-dct:nginx-x.0.0 <local registry
url>:nginx-x.0.0
```

Note: Re-tag all remaining DCT images in the same format.

- Push re-tagged images into internal registry. Sample push example for one image:

```
$ docker push <local registry url>:nginx-x.0.0
```

6.2.1.4.2 Deploy

1. Update the values.yaml file for registry name and image credentials:
   a. Registry: `<local registry url>`
   b. Username: `<local registry username>`
   c. Password: `<local registry password>`
2. Install.
   a. Follow the instructions below.

---

81 https://download.delphix.com/file/12747/delphix-dct-9.0.0.tar.gz
82 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT
83 https://download.delphix.com/file/12747/delphix-dct-9.0.0.tar.gz
84 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

6.2.1.4.2.1 Find and update fsGroup in values.yaml file

The **fsGroup** field is used to specify a supplementary group ID. All processes of the container, the owner of the volume, and any files created on the volume are also part of this supplementary group ID.

For OpenShift deployment, this value need to be specified in the values.yaml file.

Find the allowed supplementary group range:

```
oc get project dct-services -o yaml
```

A response should appear as follows:

```
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  annotations:
    openshift.io/description: ""
    openshift.io/display-name: ""
    openshift.io/requester: cluster-admin
    openshift.io/sa.scc.mcs: s0:c32,c4
    openshift.io/sa.scc.supplemental-groups: 1001000000/10000
    openshift.io/sa.scc.uid-range: 1001000000/10000
  creationTimestamp: "2023-01-18T10:33:04Z"
  labels:
    kubernetes.io/metadata.name: dct-services
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/audit-version: v1.24
    pod-security.kubernetes.io/warn: restricted
    pod-security.kubernetes.io/warn-version: v1.24
  name: dct-services
  resourceVersion: "99974"
  uid: ccdd5c9f-2ce5-49b4-91a7-662e0598b63b
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
```

Copy the first value from the `openshift.io/sa.scc.supplemental-groups` line, before the slash (e.g. 1001000000). Paste this value in the values.yaml file:

```
# Define SecurityContextConstraints for the pod
podSecurityContext:
 fsGroup: 1001000000
```

6.2.1.4.2.2 Deploy DCT

Run the following command to deploy the DCT chart (where `x.0.0` should be changed to the version of DCT being installed):

```
helm install dct-services delphix-dct
```

> **delphix-dct** is the name of the folder which was extracted in the previous step. `dct-services` is the chart name which is given for this deployment.
>
> In the above directory structure, the **values.yaml** file contains the properties which are configured above. Deploy DCT using following command to use the custom values.yaml:
>
> ```
> helm install dct-services -f <path to edited values.yaml> <directory
> path of the extracted chart>
> ```

6.2.1.4.2.3 Verify deployment

All the images will be downloaded and then deployed. If some pods restarted at the startup, this is expected. After some time, a total of 9 pods will be in running status and one job pod will be in completed status.

```
oc get pods -n dct-services
```

6.2.1.4.2.4 Find API key

For the very first deployment bootstrap API key will be printed in logs, please view gateway pod logs and find for "NEWLY GENERATED API KEY". the value is the API key.

```
oc logs <gateway-pod-name> -n dct-services
```

6.2.1.5 Configure Ingress

DCT only works with HTTPS Ingress, the UI does not support HTTP.

6.2.1.5.1 Creating route

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

If SSL is terminated at this route, only then should the useSSL value in values.yaml be updated to false, so that 80 port will be exposed in proxy service and can be used to configure the route. The following screenshot shows the route that forwards requests to 80 port of proxy service:

If SSL is not terminated at the Route level, then create a PassTrough route and use 443 port of the proxy service, and configure the SSL certificate and key in the values.yaml file:

6.2.2 OpenShift authentication

6.2.2.1 Introduction

DCT uses Nginx/OpenResty as an HTTP server and a reverse proxy for the application. Using the default configuration, all connections to DCT are over HTTPS and require the user to authenticate. There are three supported methods for authentication; API keys, Username/Password, and OpenID Connect.

6.2.2.2 Enable OAuth2 authentication

By default APIKey authentication will be enabled and when DCT starts it will generate a new API key (see page 119) in logs if you want to enable openId connect authentication then follow below procedure:

Update the below properties in the **values.yaml** file and restart DCT:

```
# flag to enable api_key based authentication
apiKeyEnabled: false
# flag to enable OAuth2 based authentication
openIdEnabled: true
# URL of the discovery endpoint as defined by the OpenId Connect Discovery
specification. This needs to be set if 'openIdEnabled' is set to true
openIdServerUrl: https://delphix.okta.com/oauth2/default/.well-known/oauth-
authorization-server
# OAuth2 jwt claim name that should be used as client_id
jwtClaimForClientId: sub
# OAuth2 jwt claim name that should be used as client_name
jwtClaimForClientName: sub
```

6.2.3 DCT logs for OpenShift

All DCT containers log to **stdout** and **stderr**, so that their logs are processed by OpenShift. To view container level logs running on the OpenShift cluster, use this command:

```
oc logs <pod_name> -n dct-services
```

Log aggregators can be configured to read from **stdout** and **stderr** for all of the pods as per the requirements.

6.2.4 Admin topics for OpenShift

-
-

6.2.4.1 Deployment upgrade for OpenShift

This page covers the upgrade process for DCT deployments on Kubernetes.

> In DCT 12.0.0 and above, one new service (container) has been introduced named **Masking**.
>
> The memory and CPU requirement for this newly added service (container) is 512Mi and 500m, respectively.
>
> This container creates a new persistent volume of 1GB dynamically. If you are using pre-existing static persistent volumes for other existing pods (i.e. gateway and database), similarly to these pods, create a new persistent volume claim for the masking pod, then update the `maskingPvcName` property in the values.yaml with the name of static persistent volume claim.

> For customizing the **masking** service resources (i.e. requests and limits) and properties, please refer to the values.yaml file in the Helm chart of the current release.

> Starting from DCT 15.0.0 we have introduced one new service (container) named **virtualization-app**.
>
> The memory and CPU requirement for this newly added service (container) is 512Mi and 600m, respectively.
>
> This container creates a new persistent volume of 1GB dynamically. If you are using pre-existing static persistent volumes for other existing pods (i.e. gateway and database), similarly to these pods, create a new persistent volume claim for the virtualization-app pod, then update the `virtualizationAppPvcName` property in the values.yaml with the name of the static persistent volume claim.
>
> For customizing **virtualization-app** service resources (i.e. request and limits) and properties, please refer values.yaml file in the Helm chart of thre current release.

Create a new folder called **dct-x.0.0** where `x.0.0` should be changed to the version of DCT being installed (e.g. if on 5.0.2, it would be 6.0.0).

```
mkdir dct-x.0.0
```

Update the added repos with the following, which will update the local HELM charts metadata (index.yaml):

```
helm repo update
```

Pull the HELM charts:

> This command will download a file named **delphix-dct-x.0.0.tgz** in the folder dct-x.0.0.

```
cd dct-x.0.0
helm pull dct-services/delphix-dct --version x.0.0
```

The downloaded file is then extracted using the following command (where `x.0.0` should be changed to the version of DCT being installed):

```
tar -xvf delphix-dct-x.0.0.tgz
```

Which will extract into the following directory structure:

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

Create a new values.yaml inside to the dct-x.0.0 folder (e.g. parallel to delphix-dct folder), use sample values.yaml given below. Copy the configured properties from previous version to this newly created values.yaml[85].

The values.yaml file contains only modified values from the previous version of deployment.

Update the username and password in values.yaml. It can be obtained from https://download.delphix.com[86]. Here are some notes in regards to this step in the process:

- This username and password update in values.yaml is only required if the user using Delphix provided a Docker Registry directly in the deployment (i.e. values.yaml).

Username and password update in values.yaml is required only if username AWS and temporary password (12 hours expiry) is used.

OFFLINE MODE

Offline mode is supported for scenarios including:

- Clusters that are air-gapped and cannot access the Delphix Docker registry.

- Instances where Delphix Docker images must be pushed to an organization's internal registry for security scans or to comply with other security policies.

For these scenarios, a jumpbox machine with access to both the Delphix Docker registry or download site[87] and the organization's internal registry is required. This jumpbox must also meet the installation prerequisites.

---

85 https://delphixdocs.atlassian.net/wiki/download/attachments/243532256/values.yaml?
   api=v2&cacheVersion=1&modificationDate=1728405184817&version=1
86 https://download.delphix.com/
87 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

1. Get the Docker images

The two ways to get the Delphix Docker images are:

- Pull the images from Delphix Docker registry.

- Download and load the images from tarball.

2. Pull the images from Delphix Docker registry

- Login into the jumpbox machine.

- Pull the Docker images from the Delphix Docker registry.

```
$ docker pull dct.download.delphix.com/delphix-dct:nginx-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:app-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:data-bookmarks-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:delphix-data-
library-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:graphql-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:ui-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:jobs-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:postgres-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:virtualization-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:masking-x.0.0
$ docker pull dct.download.delphix.com/delphix-dct:virtualization-app-
x.0.0
```

- Re-tag all above Docker images. Below is a sample re-tag example for one image:

```
$ docker tag dct.download.delphix.com/delphix-dct:nginx-x.0.0 <local
registry url>:nginx-x.0.0
```

Note: Re-tag all remaining DCT images in the same format.

- Push re-tagged images into the internal registry. Below is a ample push example for one image:

```
$ docker push <local registry url>:nginx-x.0.0
```

3. Download and load the images from tarball

If you do not have direct access to Delphix HELM and Docker repositories, download the HELM charts and Docker images packages as delphix-dct-x.0.0.tar.gz[88], which is uploaded on the download site[89].

- Login into the jumpbox machine.

---

88 https://download.delphix.com/file/12747/delphix-dct-9.0.0.tar.gz
89 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

- Download the HELM chart and Docker images packages as delphix-dct-x.0.0.tar.gz[90], which is uploaded on the download site[91].

- Extract the downloaded .tar.gz file to get the HELM chart and Docker images.

- Load extracted Docker images using following:

```
for image in *tar; do sudo docker load --input $image; done
```

- Re-tag all above Docker images. Sample re-tag example for one image:

```
$ docker tag registry.delphix.com/delphix-dct:nginx-x.0.0 <local
registry url>:nginx-x.0.0
```

Note: Re-tag all remaining DCT images in the same format.

- Push re-tagged images into internal registry. Sample push example for one image:

```
$ docker push <local registry url>:nginx-x.0.0
```

4. Deploy

- Update values.yaml for registry name and image credentials:

  a. `registry: <local registry url>`

  b. `username: <local registry username>`

  c. `password: <local registry password>`

- Install.

  a. Follow the instructions below.

- Run the HELM upgrade command:

```
helm upgrade -f values.yaml dct-services delphix-dct
```

6.2.4.2 Factory reset DCT for OpenShift

To clean DCT installation run following command:

---

90 https://download.delphix.com/file/12747/delphix-dct-9.0.0.tar.gz
91 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

```
helm delete dct-services
```

> ⚠️ This process will delete both services pod and database.

6.3 Appliance (OVA)

DCT now offers an on-engine deployment method. This method is distributed as a closed appliance virtual machine image, similar to the standard Continuous Data and Continuous Compliance engines, providing a streamlined way to deploy DCT directly on the engine. Use the pages below to get started.

- Appliance requirements and prerequisites (see page 106)
- Appliance installation and setup (see page 107)

6.3.1 Appliance requirements and prerequisites

6.3.1.1 Technical requirements

The DCT engine is distributed as a virtual machine image on both ESX (OVA) and AWS (AMI, not available on the AWS Marketplace or GovCloud).

The minimum requirements specific to DCT are the following:

- 8 vCPU
- 16GB RAM
- 128GB (system disk) + 50GB disk (metadata volume)

6.3.1.2 Supported platforms

6.3.1.2.1 AWS

Any AWS instance type satisfying the above requirements is acceptable.

6.3.1.2.2 ESX

- VMware Cloud
- VMware ESXi 8.0, 8.0 U1
- VMware ESXi 7.0, 7.0 U1, 7.0 U2, 7.0 U3
- VMware ESXi 8.0, 8.0 U1, 8.0 U2, 8.0 U3

6.3.2 Appliance installation and setup

The VM image installation instructions are similar to the instructions for standard engines, where an OVA file is downloaded and installed. The OVA file or link to the AMI can be found on the Delphix downloads[92] site, under the **DCT Engine** folder of the **DCT** area (not in the *Delphix Engines* area).

With a structure similar to (but with fewer options than) standard engines:

You must then use the VM console and wait for the Delphix Management Service and Delphix DCT Service (new) to come online, which can take up to 30 minutes (longer than for standard engines):

---

92 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

```
[  OK  ] Finished Execute cloud user/final scripts.
[  OK  ] Reached target Cloud-init target.
[  OK  ] Started Delphix Perfomance Statistics.
[  OK  ] Started Delphix default PostgreSQL database server.
[  OK  ] Stopped Wait for Network to be Configured.
         Stopping Wait for Network to be Configured...
         Stopping Network Service...
[  OK  ] Stopped Network Service.
         Starting Network Service...
[  OK  ] Started Network Service.
[  OK  ] Started ntp-systemd-netif.service.
         Starting Wait for Network to be Configured...
[  OK  ] Finished Wait for Network to be Configured.
[  OK  ] Started ntp-systemd-netif.service.
         Starting Hostname Service...
[  OK  ] Started Hostname Service.
[  OK  ] Started Docker Application Container Engine.
         Starting Delphix DCT service...
         Starting Delphix management service...
```

Just like standard engines, you must use sysadmin credentials to configure network, then use a web browser to access the server setup UI application.

The server setup UI is a slightly modified version of the setup of standard engines. Since it is a distinct build from standard engines, users do not have to select the type of engine to install. When the setup starts, a **Welcome** screen shows text detailing the DCT setup process:

On the next page, you must choose a password for the sysadmin user and enter an email address. While standard engines require them to choose a password for the Engine admin and optionally Masking admin, the DCT engine requires them to copy and save a DCT bootstrap API key:

This process replaces the need to generate a bootstrap API key and obtain it from the Docker-Compose/ Kubernetes logs for existing DCT form factors[93].

> 🛈 After initial setup, users will have the ability to re-generate a bootstrap API key via the server setup screen.

Users then go through the same **Time**, **Network**, **Network Security**, **Storage**, and **Registration** pages as for standard engines:

- In the Network Security page, the KeyStore/HTTPs configuration must be used to configure the TLS certificate of the inbound HTTPs interface (nginx) of the DCT engine. This replaces the need to configure an ingress[94] for Kubernetes deployments.

The Trust Store configuration can be used to configure certificate authorities that DCT should trust when verifying TLS connections to Engines or Hyperscale orchestrators, which will be connected to this DCT Engine. This replaces the manual edits to values.yaml[95] for Kubernetes deployments.

---

93 https://dct.delphix.com/docs/latest/bootstrapping-api-keys

94 https://dct.delphix.com/docs/latest/ingress-setup

95 https://dct.delphix.com/docs/latest/engines-connecting-authenticating#id-(21.0.0)Engines:connecting/ authenticating-TruststoreforHTTPS

> ℹ️ The DSP and STUNNEL SERVER sections can be ignored for DCT Engines.

- Note on the Storage page, use of Object Storage for DCT Engines is not supported. Block storage disks will be used to store both the Management service metadata (very minimal) and DCT metadata, we recommend a minimum of 50GB of storage (but the enforced minimum is lower).

Some sections (SSO/LDAP, etc.) which are not relevant to DCT Engines are not displayed. Configuration of Phone home, SAML/SSO, LDAP reside in the DCT application itself (accessible post setup) and not in the server setup.

Users then review the Summary screen and hit Submit, which configures the engine and restarts the applications.
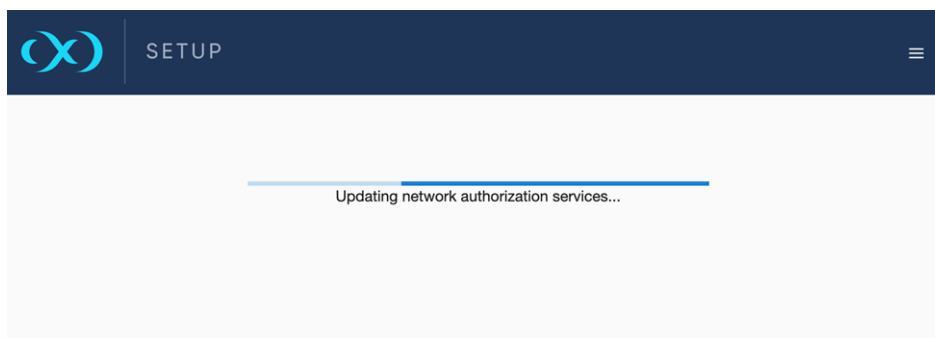
Post restart, browsers are redirected to the DCT UI application, which runs exclusively on HTTPs (HTTP redirects to HTTPs) under the **/dct** prefix. For example, https://<engine-url>/dct .
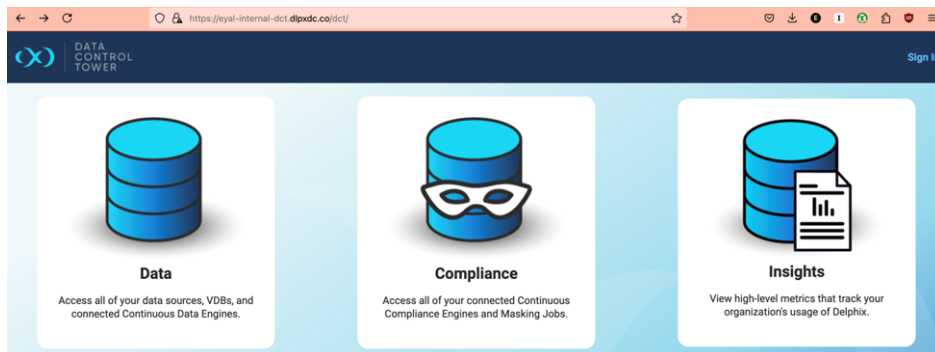
Via the Sign in and Api Key hyperlinks, users can log in using the Bootstrap API key retrieved during initial setup. Another option is to regenerate a new Bootstrap API key via the Setup app; a Hyperlink to the Setup app is shown on the DCT login screen and vice versa.
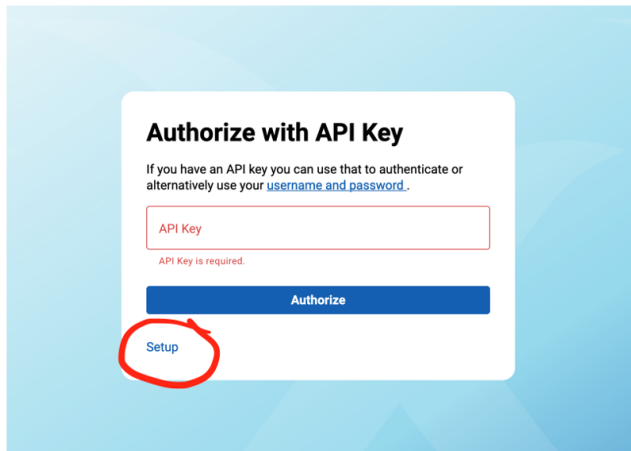
Post login, the Engine DCT UI and API experience is indistinguishable from the DCT UI experience on Docker-Compose and Kubernetes.
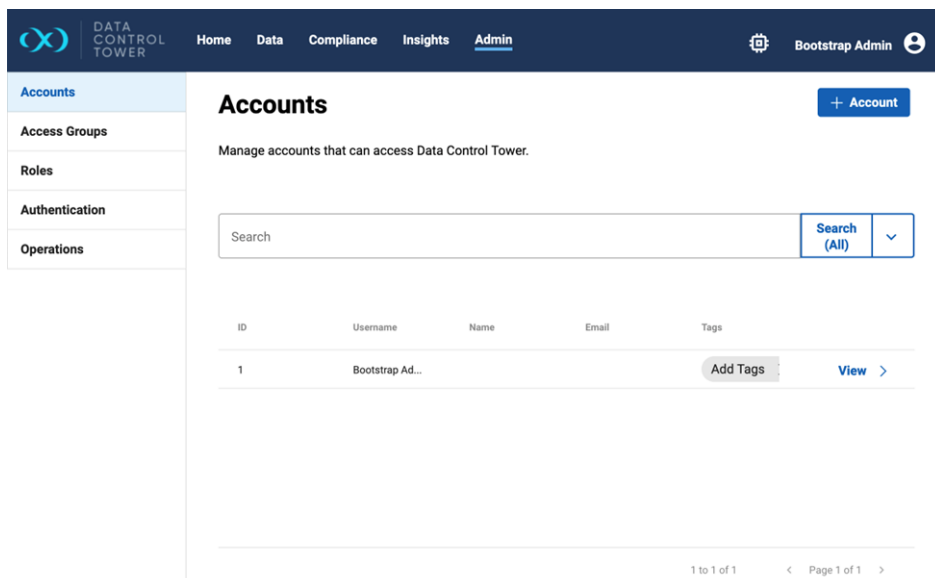
DCT accounts, roles, authentication, etc. is managed via the DCT UI, while sysadmin authentication, TrustStore/KeyStore, engine registration, network, and storage can be configured via the Setup UI, as well as regenerating a DCT bootstrap API key (in case admin access to DCT has been lost).

6.3.2.1 Limitations

The "Software version" component of the setup app is currently reporting the engine's "Appliance version" on top of which DCT is built—**not** the DCT Engine version. Multiple DCT Engines versions may (in the future) be built on top of the same appliance version; but at present, this number is not a good characterization of the DCT Engine version. There is an RFE filed to include the DCT Engine version as well.

6.4 Helm chart configuration (Kubernetes and OpenShift)

In order to support the deployment of multiple DCT instances in the same namespace or project, Helm charts must be made configurable to make resources (i.e. services, deployments, and pods) unique to that deployment. A suffix (`nameSuffix`) is provided during deployment to avoid resource name collision.

An example scenario where this might be necessary is if DCT version 15.0.0 were deployed in the `dct-services` namespace and you want to deploy another DCT instance of the same version or higher in the same namespace. The version 15.0.0 instance could have a `nameSuffix` like **v15**, while the 16.0.0 instance could have a suffix of **v16**, thus preventing a name collision.

> ℹ️ This feature works for internal and external database deployment models. If you are using an external database, then before deploying DCT with a suffix, the databases for all DCT services must be created with the same suffix:
> - appv15
> - bookmarksv15
> - data-libraryv15
> - jobsv15
> - virtualizationv15
> - maskingv15

6.4.1 Feature limitations

Deployments with a `nameSuffix` are only applicable for fresh deployments or upgrades to futures version releases of deployments with the same suffix.

6.4.2 Helm chart configuration

Deploying DCT with resource names appended with a suffix, resources must be made specific to that deployment. Use the option provided in the **vaules.yaml** to pass the `nameSuffix` to resources as follows:

```
nameSuffix:v15
```

If there is a need to deploy multiple instances of DCT in the same namespace or project, set `nameSuffix` to a string that will not conflict with the resources from each deployment.

> ℹ️ If you are enabling multiple instance deployment in the same namespace, first you must create namespace or project before the deployment of DCT and set `useExistingNamespace` to true in the values.yaml for both deployments.

> ⚠️ For multiple instance deployments in the same namespace, create a namespace or project separately to avoid the uninstallation of other DCT instances.
> If the namespace was not created manually, the helm installation will create the Namespace during the deployment of **Instance-1.** In this scenario, whenever **Instance-1** will be uninstalled, the namespace will be deleted and all other resources (like **Instance-2**) will also be removed.
> Use this command to create a namespace manually:
> ```
> $ kubectl create namespace dct-services
> ```

Once `nameSuffix` is configured in values.yaml, it gets appended to the resources of that deployment.

To deploy another instance of DCT in the same namespace, provide another `nameSuffix` string (i.e. **v16**) so the resources for the this deployment are associated with it. In the additional deployment(s), you must set the `useExistingNamespace` value to true:

```
useExistingNamespace:true
```

6.4.3 Accessing deployed DCT instances

To access deployed instances of DCT, you must create ingress routes for requests to the appropriate deployment. The ingress configuration depends on user deployment, but the following example could be used as reference.

6.4.3.1 Example

Two DCT instances are deployed with their respective `nameSuffix` configured to **v15** and **v16**. To access them, two ingress route configurations similar to the following would be needed.

Ingress route one

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: https-ingressv15
  namespace: dct-services
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  rules:
    - host: v15.dct.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: proxyv15
                port:
                  number: 443
```

Ingress route two

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```
  name: https-ingressv16
  namespace: dct-services
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  rules:
    - host: v16.dct.com
      http:
        paths:
          - path: /
            pathType: Prefix
```

After creating the ingress configurations, **v15.dct.com** and **v16.dct.com** would then be added in the host file so the deployed DCT instance can be accessed by corresponding URLs.

6.5 Engines: connecting/authenticating

6.5.1 Introduction

After DCT Authentication is complete, the HTTPS should be securely configured on DCT and able to be authenticated against. The next step is to register an engine with DCT so that it can fetch results. DCT connects to all engines over HTTPS, thus some configurations might be required to ensure it can communicate successfully.

6.5.2 TrustStore for HTTPS

6.5.2.1 For Kubernetes and OpenShift

If the CA certificate that signed the engine's HTTPS certificate is not a trusted root CA certificate present in the JDK, then custom CA certificates can be provided to DCT. If these certificates are not provided, a secure HTTPS connection cannot be established and registering the engine will fail. The insecure_ssl engine registration parameter can be used to bypass the check, however, this should not be used unless the risks are understood.

Get the public certificate of the CA that signed the engine's HTTPS certificate in PEM format. IT team help may be required to get the correct certificates. Base64 encode the certificate with:

```
cat mycertfile.pem | base64 -w 0
```

Copy the Base64 encoded value from the previous step and configure in values.yaml file under truststoreCertificates section. e.g. section will look like this:

```
truststoreCertificates:
<certificate_name>.crt: <base64 encode certificate string value in single line>
```

<certificate_name> can be any logically valid string value for e.g. "engine".

All the certificates configured in truststoreCertificates section will be read and included in the trustStore which would be then used for SSL/TLS communication between DCT and Delphix Engine.

6.5.2.2 For Appliance (OVA)

If you are using the Appliance (OVA) deployment method, follow the server setup instructions outlined in TrustStore settings[96].

6.5.3 Authentication with the Delphix Engine

All authentication with the Delphix Engine is done using the username and password of an Engine Administrator. There are two methods for storing these credentials with DCT:

1. **Stored and encrypted on DCT itself**: Credentials are saved securely within DCT.

2. **Retrieved from a password vault**: Credentials are fetched from an external vault. Currently, only the HashiCorp vault is supported.

Fetching the credentials from a vault is recommended for enhanced security.

6.5.3.1 Adding a Compliance Engine

When adding a compliance engine, it is necessary to add two types of users:

1. **Masking Administrator**
2. **Engine Administrator**

If an Engine Administrator is not provided, no performance metrics can be collected by DCT.

6.5.4 HashiCorp vault

There are two high-level steps to configuring a HashiCorp vault. The first is to set up authentication with the vault and register the vault. The second is to tell DCT how to get the specific engine credentials needed from that registered vault. A single vault can be used for multiple different Delphix Engines.

6.5.4.1 Vault authentication and registration

First, DCT needs to be able to authenticate with the vault. DCT supports the Token[97], AppRole[98], and TLS Certificates[99] authentication methods. This is done by passing a command to the HashiCorp CLI[100]. It is recommended to first ensure that successful authentication is done and one can retrieve the credentials with the HashiCorp CLI directly to ensure the correct commands are passed to DCT.

---

96 https://cd.delphix.com/docs/latest/truststore-settings
97 https://www.vaultproject.io/docs/auth/token
98 https://www.vaultproject.io/docs/auth/approle
99 https://www.vaultproject.io/docs/auth/cert
100 https://www.vaultproject.io/docs/commands

Adding a vault to DCT is done through API calls to the /v2/management/vaults/hashicorp endpoint. All authentication methods requires the location of the vault is provided through the env_variables property in the POST body like so:

```
"env_variables": {
    "VAULT_ADDR": "https://10.119.132.40:8200"
  }
```

6.5.4.2 Token

To use the token authentication method, this needs to be included as part of the env_variables field. The full example to register the vault would appear as:

```
curl --location --request POST 'https://<hostname>/dct/v2/management/vaults/
hashicorp' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>' \
--data-raw '{
   "env_variables": {
       "VAULT_TOKEN": "<your token>"
       "VAULT_ADDR": "https://10.119.132.40:8200"
   }
}'
```

A response should be received similar to the lines below:

```
{
   "id": 2,
   "env_variables": {
       "VAULT_TOKEN": "<your token>"
       "VAULT_ADDR": "https://10.119.132.40:8200"
   }
}
```

Note the id of the vault, this will be needed in the next step to register the engine.

6.5.4.3 AppRole

To use the AppRole authentication method, this needs to be included as part the login_command_args field, as shown below.

```
"login_command_args":
       [ "write", "auth/approle/login", "role_id=1", "secret_id=123"]
```

The full example to register the vault would appear as:

```
curl --location --request POST 'https://<hostname>/dct/v2/management/vaults/
hashicorp' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>' \
--data-raw '{
    "env_variables": {
        "VAULT_ADDR": "https://10.119.132.40:8200"
    },
    "login_command_args":
        [ "write", "auth/approle/login", "role_id=1", "secret_id=123"]
}'
```

A response should be received similar to the lines below:

```
{
    "id": 2,
    "env_variables": {
        "VAULT_TOKEN": "<your token>"
        "VAULT_ADDR": "https://10.119.132.40:8200"
    }
}
```

6.5.5 TLS certificates

The configuration of mutual TLS authentication requires an additional step. This feature currently is NOT supported for Kubernetes deployment of DCT. This will be covered in later releases.

6.5.5.1 Retrieving engine credentials

Once DCT can authenticate with the vault, it needs to know how to fetch the relevant engine credentials. When registering an engine, the user will need to provide the HashiCorp CLI commands through the `hashicorp_vault_username_command_args` and `hashicorp_vault_password_command_args` parameters.

The relevant part of the engine registration payload will look like the following:

```
'{
    "hashicorp_vault_id": 1
    "hashicorp_vault_username_command_args": ["kv", "get", "-field=username",  "kv-
v2/delphix-engine-secrets/engineUser"]
,
    "hashicorp_vault_password_command_args": ["kv", "get", "-field=password",  "kv-
v2/delphix-engine-secrets/engineUser"]
}'
```

The hashicorp_vault_id will be the ID that was returned as part of the previous step. Note that the exact paths to fetch the username and password will vary depending on the exact configuration of the vault.

6.6 Accounts: connecting/authenticating

There are 5 supported methods for authentication; **API keys**, **Username/Password**, **LDAP/Active Directory**, **SAML/SSO**, and **OpenID Connect**. These authentication methods are detailed on the corresponding pages in this section.

> 📄 DCT uses Nginx/OpenResty[101] as an HTTP server and a reverse proxy for the application. Using the default configuration, all connections to DCT are over HTTPS and require the user to authenticate. The Nginx/OpenResty configuration files can be edited via /etc/config bind mounts, for the proxy container to customize the HTTP server and change options (such as TLS versions).

- API keys (see page 119)
- Username/password (see page 121)
- LDAP/Active Directory (see page 125)
- SAML/SSO (see page 133)

6.6.1 API keys

API keys are the default method to authenticate with DCT. This is done by including the key in the HTTP Authorization request header[102] with type `apk` .

> ℹ️ API keys are long-live tokens and as a result, do not automatically expire in the future. They remain valid until they are deleted or destroyed from DCT.

---

101 https://openresty.org/en/
102 https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization

A cURL example using an example key of
`1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3` would appear as:

```
curl --header 'Authorization: apk
1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3'
```

cURL (like web browsers and other HTTP clients) will not connect to DCT over HTTPS unless a valid TLS certificate has been configured for the Nginx server. If this configuration step[103] has not been performed yet and the risk is comprehended, you may disable the check in the HTTP client. For instance, this can be done with cURL using the `--insecure` flag.

> 🗐 The cURL version must be 7.43 or higher.

6.6.1.1 Create and manage API Keys

The initial API key created should be used to create a new admin secure key. This is done by creating a new Account entity and setting the `generate_api_key`. The "username" attribute should be the desired name to uniquely identify the account.

```
curl --location --request POST 'https://<hostname>/docs/v2/management/accounts' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk
1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3' \
--data-raw '{
   "username": "secure-key",
   "generate_api_key": true
}'
```

> 🗐 If the cURL version being used is below 7.43, replace the `--data-raw` option with `--data`.

A response should be received similar to the lines below:

```
{
   "id": 2,
   "token": "2.vCfC0MnpySYZLshuxap2aZ7xqBKAnQvV7hFnobe7xuNlHS9AF2NQnV9XXw4UyET6"
   "username":"secure-key"
```

---

[103] https://delphixdocs.atlassian.net/wiki/spaces/DCT/pages/1114361/Replace+HTTPS+certificate+for+DCT

```
    }
```

Now that the new and secure API key is created, the old one must be deleted for security reasons since the key appeared in the logs. To do this make the following request:

```
curl --location --request DELETE 'https://<hostname>/dct/v2/management/api-clients/
<id>' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk
2.vCfC0MnpySYZLshuxap2aZ7xqBKAnQvV7hFnobe7xuNlHS9AF2NQnV9XXw4UyET6'
```

The id referenced above is the numeric id of the Account. It is the integer before the period in the token. For example, the id of

`1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3` is 1.

Finally, to list all of the current Accounts, make the following request:

```
curl --location --request GET 'https://<hostname>/dct/v2/management/accounts/' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>'
```

6.6.1.2 API key expiration

Support has been implemented to rotate the API key for another user account. This action is only allowed if the user has the necessary permissions to rotate/regenerate the API key for other accounts. A running DCT instance with VDBs is required.

The following existing API endpoints are utilized:

Regenerate

- **API Endpoint**: `/management/accounts/{id}/api-key-reset`
- **Inputs**: User account ID
- **Outputs**: New API key

Delete Api Key

- **API Endpoint**: `/management/accounts/{id}/api-key-delete`
- **Inputs**: User account ID

6.6.2 Username/password

When creating an account, a username and password combination can be associated with the account (whether an API Key was generated for the account or not). To do so, specify the "username" and "password" properties in the API request, for example:

```
curl -k --location --request POST 'https://<hostname>/v2/management/accounts' \
  --header 'Content-Type: application/json' \
  --header 'Accept: application/json' \
  --header 'Authorization: apk
1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3' \
  --data-raw '{
    "username": "some-username",
    "password": "some-password",
    "generate_api_key": false
    "is_admin": true
}'
```

> The **is_admin** property will create the account with admin privileges. Remove this property to create an account without admin privileges.

The username and password combination can then be used to login via the UI, or to fetch a temporary access token valid for 24 hours. To do so, call the 'login' API endpoint:

```
curl -k --location --request POST 'https://<hostname>/v2/login' \
  --header 'Content-Type: application/json' \
  --header 'Accept: application/json'  \
  --data-raw '{
    "username": "some-username",
    "password": "some-password"
}'
```

A response should be received similar to the lines below:

```
{
    "access_token":"eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJhcGlndy1zZXJ2aWNlcy1hcHAiLCJzdWIi
OiI4IiwiZXhwIjoxNjYyNTUyMzI3LCJpYXQiOjE2NjI0NjU5MjcsInVzZXJuYW1lIjoic29tZS11c2VybmFt
ZSJ9.Cx_hGU9noyWS6mtK6gjsA85FTgJRQgyJizR5t_akNps",
    "token_type":"Bearer",
    "expires_in":86400
}
```

The access token can be used as HTTP Authorization request header[104] with type **Bearer**.

> This access token should not be confused with ApiKeys. These tokens are short lived tokens and expire at the end of 24 hours.

---

[104] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization

A cURL example using the access token retrieved above would appear as:

```
curl --header 'Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJhcGlndy1zZXJ2aWNlcy1hcHAiLCJzdWIiOiI4IiwiZXhwIjoxNjYy
NTUyMzI3LCJpYXQiOjE2NjI0NjU5MjcsInVzZXJuYW1lIjoic29tZS11c2VybmFtZSJ9.Cx_hGU9noyWS6mtK
6gjsA85FTgJRQgyJizR5t_akNps'
```

The password for an account can be updated with the **change_password** API endpoint, passing in both the old and new passwords, such as in this example:

```
curl -k --location --request POST '<hostname>/v2/management/accounts/3/
change_password \
  --header 'Content-Type: application/json' \
  --header 'Accept: application/json'  \
  --header 'Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJhcGlndy1zZXJ2aWNlcy1hcHAiLCJzdWIiOiI4IiwiZXhwIjoxNjYy
NTUyMzI3LCJpYXQiOjE2NjI0NjU5MjcsInVzZXJuYW1lIjoic29tZS11c2VybmFtZSJ9.Cx_hGU9noyWS6mtK
6gjsA85FTgJRQgyJizR5t_akNps' \
  --data-raw '{
      "old_password":  "some-password",
    "new_password": "new-password"
}'
```

Following security best practices, the password is not stored on DCT and cannot be retrieved. If the password has been lost, an account with admin privilege can reset the password for a particular account. It is recommended to change the password reset by an admin account on the first login, or with the **change_password** API, as described above.

```
curl -k --location --request POST '<hostname>/v2/management/accounts/2/
password_reset' \
  --header 'Content-Type: application/json' \
  --header 'Accept: application/json'  \
  --header 'Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJhcGlndy1zZXJ2aWNlcy1hcHAiLCJzdWIiOiI4IiwiZXhwIjoxNjYy
NTUyMzI3LCJpYXQiOjE2NjI0NjU5MjcsInVzZXJuYW1lIjoic29tZS11c2VybmFtZSJ9.Cx_hGU9noyWS6mtK
6gjsA85FTgJRQgyJizR5t_akNps' \
  --data-raw '{
    "new_password": "new-password"
}'
```

In the above example, the admin is resetting the password of an account with id **2** to "new-password".

6.6.2.1 Password policies

The password policy feature allows users to enable and customize the password policy enforced for local username/password authentication (does not apply to LDAP/Active Directory or SAML/SSO based authentication).

6.6.2.2 Understanding password policies

The password policy is a set of requirements that local passwords must satisfy:

Property	Default value	Description
min_length	15	A password must be longer than this length.
reuse_disallow_limit	2	The user should not reuse old passwords. This tells the number of last used passwords disallowed to be reused as the new passwords.
uppercase_letter	true	A password must have at least one capital letter.
lowercase_letter	true	A password must have at least one lower case letter.
digit	true	A password must have at least one digit
special_character	true	A password must have at least one special character, such as #, $, !
disallow_username_as_password	true	A password should not be the same as the user name.
maximum_password_attempts	5	The number of allowed attempts for incorrect password, after which the account gets locked.

6.6.2.3 Changing the password policy

To change the current password policy, call the password policy API endpoint, as shown in the example below:

```
curl --location --request PATCH 'https://<hostname>/v2/management/accounts/password-
policies' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>' \
--data-raw '{
    "enabled": true,
    "maximum_password_attempts": 2,
    "min_length": 5,
    "reuse_disallow_limit": 3,
    "digit": true,
    "uppercase_letter": true,
    "lowercase_letter": true,
    "special_character": true,
    "disallow_username_as_password": true
}'
```

Changing the password policy does not affect existing passwords.

6.6.2.4 Disabling local username/password authentication

Username/password authentication (with passwords locally in DCT) can be disabled for individual accounts by not setting or unsetting their password property, or across the DCT instance using the global properties API. Disable username/password authentication to force authentication to use an alternate authentication method (LDAP/Active Directory, SAML/SSO, etc.) as shown in this example:

```
curl --location --request PATCH 'https://<hostname>/v2/management/properties' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>' \
--data-raw '{"disable_username_password": true}'
```

6.6.3 LDAP/Active Directory

6.6.3.1 Configuration

LDAP/Active directory can be used to authenticate login requests, and optionally to retrieve additional information about accounts, thereafter referred to as LDAP Search.

6.6.3.1.1 Configuring authentication

The following attributes must be set to configure LDAP/Active Directory authentication.

Property Name	Description
enabled	Whether the LDAP/Active Directory feature is enabled.
auto_create_users	Whether DCT must automatically create account records for successful authentication attempts using a username which does not match any accounts.
	If this is disabled, an administrator must create a DCT account with an ldap_principal attribute matching the value from the LDAP/Active Directory server prior to the first login attempt.
	If this is enabled, any user with valid credentials in the LDAP/Active Directory server can authenticate to DCT, by default with an empty authorization set (i.e not being able to view any data or perform any action).
hostname	The host name or IP address of the LDAP/Active Directory server.
port	Port of the LDAP/Active Directory server. This is usually 389 for non SSL, and 636 for SSL.
enable_ssl	Whether the connection to the LDAP/Active Directory server must be performed over SSL. It is highly advised to use SSL. Without SSL, communication between DCT and the LDAP/Active server can be intercepted.
insecure_ssl, unsafe_ssl_hostname_check, truststore_filename, truststore_password	The SSL protocol requires the LDAP/Active Directory server to expose a certificate signed by a Certificate Authority (CA) trusted by the JDK which is running DCT. Refer to the dedicated section below to see how to configure an Active Directory/LDAP server of which certificate is not recognized.

Property Name	Description
[domains].msad_domain_name	Microsoft Active Directory only: The DNS name of a domain in the same forest as the accounts which login. DCT will append the msad_domain_name to the username provided at login to form a user principal name (UPN). Example: if the msad_domain_name is http://mycompany.co and a user logs in with username john, DCT will perform an LDAP request to the Active Directory server to authenticate john@mycompany.co[105].
[domains].username_pattern	If the LDAP server is not Microsoft Active Directory, the username_pattern is used to create a DN string for user authentication. The pattern argument {0} is replaced with the username at runtime. Example: If the username_pattern is **uid={0},ou=People** and a user logs in with username john, DCT will perform an LDAP request with DN **uid=john,ou=People**.

The LDAP/Active Directory Integration can be configured both via DCT UI and API. The below image shows an example of how the configuration can be set in the UI as a way to Authenticate users, auto create new users, as well as map group attributes for authorization within the DCT Access Control system.

---

[105] mailto:john@mycompany.co

Edit LDAP Settings

☑ Enabled

☑ Auto-create Users

Hostname
activedirectory.acme.com

Port
636

Domains

+

MSAD Domain Name
acme.com

Username Pattern

Search Base
CN=Users,DC=acme,DC=com

Group Attribute
department

Email Attribute
mail   +   🗑

First Name Attribute
givenName

Last Name Attribute
sn

Object Class Attribute
person

Search Attribute
sAMAccountName

Cancel   **Save**

The following example requests enable LDAP authentication over SSL with an Active Directory server at address **activedirectory.company.co** using the **us.company.co** domain:

```
curl --location --request PUT 'https://<hostname>/v2/management/ldap-config' \
```

```
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>' \
--data-raw '{
 "enabled": true,
 "auto_create_users": true,
 "hostname": "activedirectory.company.co",
 "enable_ssl": true,
 "port": 636,
 "domains":[{
    "msad_domain_name":"us.company.co"
 }]
}'
```

6.6.3.1.2 Validating the configuration

Updating the LDAP/Active Directory configuration does not guarantee that the provided values are correct, as validating those requires a user to authenticate to DCT. This can be achieved with the ldap-config/validate API endpoints, using the credentials valid for the LDAP/Active Directory server. When provided with a username/password combination, the ldap-config/validate API endpoint will authenticate with the LDAP server. If the response status code is 200, the configuration is correct. Otherwise, the response code will be 400, and the response body will provide information to resolve the configuration problems. For example:

```
curl --location --request POST 'https://<hostname>/v2/management/ldap-config/
validate' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>' \
--data-raw '{
 "username": "<ldap-username>",
 "password": "<ldap-password>"
}'
```

Because of a defect in version 3.0.0 of DCT, the above request might fail with a response similar to:

```
search failed for john.doe with search base null' ,search attribute
'null'
```

This indicates that authentication works, and search (see below) is not configured.

6.6.3.1.3 Login

One the configuration has been updated, accounts can login (via the UI or API) using the same UI form/API endpoint they would be using for the local username/password authentication feature. For example:

```
curl -k --location --request POST 'https://<hostname>/v2/login' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--data-raw '{
 "username": "<ldap-username>",
 "password": "<ldap-password>"
}'
```

When LDAP/Active directory is enabled, DCT first attempts to validate passwords with the LDAP/Active Directory server, and falls back to local password authentication in case of failure. Enabling LDAP/Active directory is thus a non disruptive operation for existing accounts.

In order to force a transition to LDAP/Active Directory only password authentication, the DCT administrator must either update the account records to remove the password, or disable local password authentication entirely.

6.6.3.2 Configure LDAP/Active Directory groups

In addition to being an authentication method, the LDAP/Active Directory integration can optionally also be used to retrieve additional attributes about the accounts authenticating: first name, last name, email address and group membership.

DCT only supports retrieving groups which are exposed as an attribute of the LDAP/Active Directory user record. DCT can not fetch groups membership from group records at the LDAP/Active Directory, and thus also does not support nested groups.

Group memberships are retrieved at authentication time, using the account credentials. DCT does not need credentials of an LDAP/Active Directory administrator, but will only be able to retrieve group memberships if LDAP/Active Directory users have the right to read the corresponding attribute.

This can be enabled by setting additional arguments to the domain API object.

search_base	The Context name in which to search. Being specific enables faster LDAP search.
	To construct the search_base DN string according to your LDAP/Active Directory server, using an LDAP browser, navigate to a user, and then construct the search_base DN in reverse order from the User, up the folder hierarchy. For example:
	If a User DN is:
	CN=some-user-id,CN=Users,DC=mycompany,DC=co
	The corresponding search base might be:
	CN=Users,DC=mycompany,DC=co
email_attr	Name of the attribute in the LDAP/Active Directory server containing email addresses.
	Example: **mail**
last_name_attr	Name of the attribute in the LDAP/Active Directory server containing last names
	Example: **sn**
first_name_attr	Name of the attribute in the LDAP/Active Directory server containing first names
	Example: **givenName**
group_attr	Name of the attribute in the LDAP/Active Directory server containing group(s) membership. This can be a multi-valued attribute.
	Example: **memberOf**
search_attr	Name of the attribute in the LDAP/Active Directory server of which value corresponds to the username provided to the DCT login requests.
	For Active Directory, this is usually **sAMAccountName**.
	Example: If the search base is **CN=Users,DC=mycompany,DC=co** and the search_attr is **principalName**, DCT will search for a record with a principalName matching the username provided to the login request under the **CN=Users,DC=mycompany,DC=co** sub tree.

object_class_attr	Restricts search to records with an objectClass matching this value. Example: **person**

6.6.3.2.1 Active Directory example

The following requests enable LDAP authentication over SSL with an Active Directory server at address **activedirectory.company.co**, using the **us.company.co** domain, and configures optional attributes to retrieve first name, last name, email address, and group membership from the users sub-tree.

```
curl --location --request PUT 'https://<hostname>/v2/management/ldap-config' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>' \
--data-raw '{
 "enabled": true,
 "auto_create_users": true,
 "hostname": "activedirectory.mycompany.co",
 "enable_ssl": true,
 "port": 636,
 "domains":[{
   "msad_domain_name":"mycompany.co",
   "search_base":"CN=Users,DC=mycompany,DC=co",
   "email_attr": "mail",
   "first_name_attr": "givenName",
   "last_name_attr": "sn",
   "group_attr": "memberOf",
   "object_class_attr":"person",
   "search_attr": "sAMAccountName"
 }]
}'
```

With the above config, when a user logs in with username John, DCT will:

1. Authenticate with the Active Directory server using the user principal name john@mycompany.co[106] and supplied password.

2. Search in the **CN=Users,DC=mycompany,DC=co** sub tree a record with **objectClass=person** and **sAMAccountName=john**.

3. Create or update a DCT Account record with the attributes extracted from the Active Directory server.

4. For each group membership found in the `memberOf` of the Active Directory server, an account tag is created with **key=login_groups** and value is the group name. These tags are protected (i.e cannot be modified within DCT) and can be securely used to control access groups membership.

---

[106] mailto:john@mycompany.co

As explained above, the **ldap-config/validate** API endpoint can be used to validate that each of the attributes corresponding to LDAP/Active Directory attributes.

6.6.4 SAML/SSO

6.6.4.1 General points

- The SAML 2.0 protocol allows DCT to delegate authentication to a SAML 2.0 compatible Identity Provider (Active Directory Federation Services, Azure Active Directory, PingFederate, Okta, OneLogin, etc.) and only applies to web browser-based interaction. It cannot be used for API access (scripting, integration).
- Setting up SAML/SSO requires configuration changes both in the Identity Provider and DCT for trust to be established across both products.
- When using SAML/SSO, DCT will uniquely identify accounts by email address. With that, make sure records at the identity provider are configured with a unique email address.
- DCT supports automatic account creation (or "just in time" account provisioning) when using SAML/SSO. When automatic account creation is enabled, accounts are created automatically when users login for the first time.
- DCT allows group membership to be retrieved from the Identity Provider, which can be used to control access control authorization within DCT via DCT Access Groups. Using Identity Provider group membership allows DCT authorization to be managed per account group and guarantees that authorizations in DCT reflect the organization structure, which is expressed by group membership of the identity provider.
- SAML/SSO is not mutually exclusive with other authentication methods, so enabling SAML/SSO is not disruptive (accounts configured with local password or LDAP/Active Directory authentication can still authenticate).
    - In order to switch to SAML/SSO exclusively as an authentication method for web browser interaction, perform the SAML/SSO configuration steps below and disable LDAP/Active Directory and Username/Password authentication. API Key-based authentication cannot be entirely disabled and only administrators can create accounts with API keys.

6.6.4.2 Identity provider setup

Require that an administrator of the Identity provider used by your organization sets up a SAML 2.0 integration with DCT (an integration is sometimes called a Relying party trust, or an application).

The exact instructions are product specific, but the following input values must be provided:

Name	Alternate names (depending on the product)	Value
Single Sign-on URL	• SAML Assertion Consumer Service • ACS • Recipient URL • Destination URL • Relying party SAML 2.0 SSO • Service URL • Reply URL	• Container-based installs: https://\<dct-hostname\>/v2/saml/SSO • Engine-based installs: https://\<dct-hostname\>/dct/v2/saml/SSO
Audience URI	• SP Entity ID • Relying Party trust identifier	Any value can be selected, as long as the same value is set in the Identify Provider configuration and DCT configuration. We recommend: https://\<dct-hostname\>
Binding	• POST	
Protocol		SAML 2.0 WebSSO protocol

The identity provider must be configured to include the email address as `NameId` attribute, and DCT will use the email attribute as a unique identifier for users when connecting via SAML/SSO.

6.6.4.3 DCT SAML/SSO setup

Once the configuration has been performed at the Identity provider, use the saml-config API endpoint to configure DCT accordingly. Copy the metadata from the Identity Provider using a web browser and provide it directly to DCT.

The Identity provider (IDP) metadata is a standardized XML document providing the SAML Service Provider (DCT) with the necessary information to verify the validity of incoming login requests and initiate a SAML/SSO login flow.

If `auto_create_users` is enabled, DCT will create accounts automatically when they login with SAML/SSO for the first time. If this is disabled, an administrator must create a DCT account with an email attribute matching the value from the SAML/SSO Identity provider before they can login. When `auto_create_users` is enabled, any user configured to authenticate via the Identity provider can authenticate to DCT, by default with an empty authorization set (i.e not being able to view any data or perform any action).

Without network access, provide the IDP metadata directly:

```
curl --location --request PUT 'https://<hostname>/v2/management/saml-config' \
```

```
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>' \
--data-raw '{
    "enabled": true,
    "auto_create_users": true,
    "metadata": "<json-escaped-idp-metata-xml-blob>",
}'
```

> 🗏 The IDP metadata must be JSON escaped. On a terminal with ./jq[107] installed, this can be achieved with the following command: `jq --slurp --raw-input <<< 'xml-metadata-here'`

6.6.4.4 Login

The SAML 2.0 protocol defines two login procedures: The Service Provider initiated flow starts by having users point their web browser to `https://<dct-hostname>/v2/saml/login` to login, while the Identity provider-initiated flow starts at the Identity provider (details specific to Identity provider vendor). DCT supports both flows. The SAML/SSO authentication method is not intended for API interaction and cannot be used with the Swagger UI.

After successful authentication, the web browser is redirected to the UI landing page. Use the navigation bar at the top to go to the desired page. The session expires 24 hours after login.

6.6.4.5 Troubleshooting

There was an issue in SAML authentication: The assertion cannot be used before <timestamp>.

- The above error message, which is accompanied by **com.coveo.saml.SamlException**: The assertion cannot be used before <timestamp> error in the application logs, indicates that DCT was not able to validate the timestamp of the authentication provided by the Identity Provider. This is usually due to the system clock of the machine running DCT being incorrectly configured. Consider using NTP to maintain the machine's clock up to date.

There was an error fetching data.

- The above error message indicates that the current account does not have permission to view the data displayed on the page. Remember that, while DCT creates accounts automatically upon login when auto_create_users is enabled, by default accounts are created without any authorization and thus cannot see any data. Review the section below to see how SAML/SSO group membership can be assigned automatically at account creation.

---

[107] https://stedolan.github.io/jq/

6.6.4.6 Attributes mapping

As explained above, the only required attribute in the SAML Response (the message sent by the Identity Provider to DCT during login) is the `NameId` attribute which must be configured to a unique email address.

In addition to this, DCT allows for first name, last name, and group membership attributes to be included. The first and last names attributes will be stored as properties of the account object. For each group membership found in the SAML response attribute, an account tag is created with `key=login_groups` and value is the group name. These tags are protected (i.e cannot be modified within DCT) and can be securely used to control access groups membership.

In other to enable these optional attributes, update the Identity provider configuration to include them in the SAML response, and use the **saml-config** API endpoint to configure DCT with the name of the attributes configured in the Identity provider:

```
curl --location --request PUT 'https://&lt;hostname>/v2/management/saml-config' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>' \
--data-raw '{
  "enabled": true,
  "auto_create_users": true,
  "metadata": "<json-escaped-idp-metata-xml-blob>",
  "first_name_attr": "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/
givenname",
  "last_name_attr": "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/
surname",
  "group_attr": "http://schemas.xmlsoap.org/claims/Group"
}'
```

With the above configuration, and a SAML Response as the following produced by the Identity Provider during login:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<saml2:Assertion ID="id979239831676038211157180516" IssueInstant="2022-12-01T10:07:
12.856Z" Version="2.0"
    xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
    <saml2:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">http:/
/www.idp-demo.com/exk1fupjwz1YcMo290h8</saml2:Issuer>
    <saml2:Subject>
        <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified">john.doe@company.co</saml2:NameID>
        <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml2:SubjectConfirmationData NotOnOrAfter="2022-12-01T10:12:12.857Z"
 Recipient="https://localhost/v2/saml/SSO"/>
        </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2:Conditions NotBefore="2022-12-01T10:02:12.857Z" NotOnOrAfter="2022-12-0
1T10:12:12.857Z">
        <saml2:AudienceRestriction>
            <saml2:Audience>https://dct-demo.delphix.com</saml2:Audience>
        </saml2:AudienceRestriction>
    </saml2:Conditions>
    <saml2:AuthnStatement AuthnInstant="2022-12-01T10:05:07.916Z" SessionIndex="id
1669889232855.2084756273">
        <saml2:AuthnContext>
            <saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Pas
swordProtectedTransport</saml2:AuthnContextClassRef>
        </saml2:AuthnContext>
    </saml2:AuthnStatement>
    <saml2:AttributeStatement>
        <saml2:Attribute Name="http://schemas.xmlsoap.org/ws/2005/05/identity/
claims/givenname" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:unspecified">
            <saml2:AttributeValue
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs
:string">John
            </saml2:AttributeValue>
        </saml2:Attribute>
        <saml2:Attribute Name="http://schemas.xmlsoap.org/ws/2005/05/identity/
claims/surname" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:unspecified">
            <saml2:AttributeValue
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs
:string">Doe
            </saml2:AttributeValue>
        </saml2:Attribute>
        <saml2:Attribute Name="http://schemas.xmlsoap.org/claims/Group"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
            <saml2:AttributeValue
```

```
                    xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs
:string">Dev-Team
            </saml2:AttributeValue>
            <saml2:AttributeValue
                    xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs
:string">QA
            </saml2:AttributeValue>
    </saml2:AttributeStatement>
</saml2:Assertion>
```

Would automatically create or update a DCT account with the following properties:

```
{
    "id": 94,
    "username": "john.doe@company.co",
    "firstName": "John",
    "lastName": "Doe",
    "email": "john.doe@company.co",
    "tags": [
        {
            "key": "login_groups",
            "value": "Dev-Team"
        },
        {
            "key": "login_groups",
            "value": "QA"
        }
    ]
}
```

6.7 Replace HTTPS certificate for DCT

For DCT Appliance (OVA) deployments, use the setup UI for network settings and the pages in the Configuring network settings[108] section.

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

---

[108] https://cd.delphix.com/docs/latest/configuring-network-settings

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

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

- To generate the Base64 encoded value of the certificate:
  ```
  cat my_cert_file.pem | base64 -w 0
  ```
- To generate the Base64 encoded value of the key:
  ```
  cat my_private_key.key | base64 -w 0
  ```

Generating a new TLS certificate and private key could require the assistance of your Security or IT departments. A new key pair (public and private key) will need to be created, in addition to a certificate signing request (CSR) for that key pair. Your IT department should be able to determine the correct certificate authority (CA) to sign the CSR and produce the new certificate. The common name of the certificate should match the fully qualified domain name (FQDN) of the host, as well as the FQDN as a Subject Alternative Name (SAN).

- After changing the *crt* and *key* values in values.yaml file. Run the HELM upgrade command:

  ```
  helm upgrade -f values.yaml dct-services delphix-dct
  ```

- After upgrading the helm chart, restart the proxy pod to pick the new certificates:

  ```
  kubectl rollout restart deployment proxy -n dct-services
  ```

6.8 External database support

6.8.1 Overview

ⓘ   This feature is not available for DCT Appliance (OVA) deployments.

DCT uses a PostgreSQL database to store all the persistent data powering the application (engines, VDBS, compliance jobs, accounts, permissions, etc.). By default, a PostgreSQL container image is packaged along with the DCT application and deployed along other pods onto the Kubernetes cluster, storing its persistent data into the `gwdatabase-data` persistent volume.

Alternatively, DCT can be configured to use an external PostgreSQL database instead, to which DCT connects over TCP and can then run anywhere (typically outside of the Kubernetes cluster).

> ⚠️ External database support is only available for Kubernetes and OpenShift deployments. Docker Compose installations are not eligible.

6.8.2 Requirements

- **Database type:** PostgreSQL
- **Min supported version:** 12.16
- **Max supported version:** 14.10

6.8.3 Setup

6.8.3.1 PostgreSQL database setup

The following databases must be created prior to connecting DCT: **app, jobs, data-library, bookmarks, masking, and virtualization**.

> ⚠️ **masking** is a newly added database for the masking service and is applicable only for release 12.0.0 and above.
> **virtualization** is a newly added database for virtualization-app service and is applicable only for release 15.0.0 and above

> ⚠️ The `max_connections` setting for database server must be greater than or equal to 200.

> ⚠️ Quotes must be used to create or connect to the **data-library** databases, since its name contains a hyphen (-). Creating the database with a different name (such as **data_library)** will NOT work.

A PostgreSQL user must be created for DCT, with either the admin privilege or ALL privilege on the above databases. The following SQL script exemplifies the creation of required databases, granting privileges to a pre-existing `dct_user` (role) user.

```
CREATE DATABASE "bookmarks";
GRANT ALL PRIVILEGES ON DATABASE "bookmarks" TO dct_user;
```

```
CREATE DATABASE "data-library";
GRANT ALL PRIVILEGES ON DATABASE "data-library" TO dct_user;

CREATE DATABASE "jobs";
GRANT ALL PRIVILEGES ON DATABASE "jobs" TO dct_user;

CREATE DATABASE "app";
GRANT ALL PRIVILEGES ON DATABASE "app" TO dct_user;

CREATE DATABASE "masking";
GRANT ALL PRIVILEGES ON DATABASE "masking" TO dct_user;

CREATE DATABASE "virtualization";
GRANT ALL PRIVILEGES ON DATABASE "virtualization" TO dct_user;
```

6.8.3.2 DCT setup

Edit the **values.yaml** file to set `useExternalDatabase` to true and also provide the `dbHost`, `dbPort`, `dbUser`, and `dbPassword`, then run the following.

```
helm upgrade -f values.yaml dct-services delphix-dct
```

⚠️ Previous versions of DCT required the aforementioned properties to be base64 encoded, but the values must be plain text values, as shown in the following excerpt:

```
useExternalDatabase: true
dbHost: "database-host.company.co"
dbPort: 5432
dbUser: "dct_user"
dbPassword: "dct_user_password"
```

⚠️ If the password is stored as exemplified above in the values.yaml file, make sure to store this file in a secure location. Alternatively, set the value using the `--set` option flag in the HELM upgrade command, as shown in the following excerpt:

```
helm upgrade -f values.yaml --set dbPassword=dct_user_password dct-services
delphix-dct
```

- After upgrading the HELM chart, restart the pods to pick the changes:

```
kubectl rollout restart deployment data-library -n dct-services
kubectl rollout restart deployment data-bookmarks -n dct-services
kubectl rollout restart deployment jobs -n dct-services
kubectl rollout restart deployment gateway -n dct-services
kubectl rollout restart deployment masking -n dct-services
kubectl rollout restart deployment virtualization-app -n dct-services
```

6.8.4 Backup and recovery

When using an external database, the `gwdatabase-data` persistent volume (created at default by DCT) to store database data is not used. Backing up and restoring the external database is not managed by DCT. Frequent or continuous backups are required, otherwise, DCT data will be lost.

The `gateway-data`, `masking-data`, and `virtualization-app-data` persistent volume must still be backed up, because it contains an encryption key, which is used to encrypt sensitive data at the application-level before being sent to the database.

A backup of the external database cannot be restored successfully without a corresponding backup of `gateway-data`, `masking-data`, and `virtualization-app-data`, as DCT would not be able to decrypt some of the data in the database.

The encryption key in `gateway-data`, `masking-data`, and `virtualization-app-data` does not change after having been initially created, so backups of it do not need to be scheduled at the same time as database backups.

6.8.5 External database migration or upgrade

The external database can be migrated to a different host and/or upgraded to a different version at any time, as long as the version requirements above are met. If the database is unavailable for a period of time, the DCT application will temporarily fail (internal server errors on all API calls), but will recover automatically without the need for a restart. However, in case of planned maintenance, upgrade, or migration, the following procedure should be followed:

1. Stop (shutdown) the DCT application.

2. Upgrade or migrate the database.

3. If necessary, set the updated database properties in values.yaml, and run `helm upgrade`.

4. Start the DCT application.

6.8.6 DCT upgrade

Before upgrading to a new DCT version, review the documentation to identify if the external database version is compatible. If the external database version is unknown, call the `metadata-database` API endpoint to get the information.

```
curl -k --location --request GET 'https://<dct-server>/v3/management/metadata-
database' \
--header 'Accept: application/json' \
--header 'Authorization: apk <api-key>'

{
  "external": false,
  "version": "14.9",
  "database_product_name": "PostgreSQL",
  "major_version": 14,
  "minor_version": 9,
  "min_supported_major_version": 12,
  "min_supported_minor_version": 16,
  "max_supported_major_version": 14,
  "max_supported_minor_version": 9,
  "compatible": true
}
```

If the version of the external database is not compatible with the requirements of the DCT version being upgraded to, follow the instructions in the **External database migration or upgrade** section above before upgrading.

> The selected PostgreSQL version to upgrade to must be compatible with both the currently running DCT version *and* the upgrade version.

After the upgrade, verify if the external database is compatible with DCT by inspecting the `compatible` property of the `metadata-database` API endpoint (as shown above).

6.9 DCT data backup, recovery, and migration

> This method is only applicable for **Kubernetes** and **OpenShift**.
> - For Kubernetes, use the **kubectl** command prefix.
> - For OpenShift, use the **oc** command prefix.

> ℹ️ For DCT Appliance (OVA) deployments, these instructions do not apply. Instead, use Hypervisor-provided compatibilities for these actions, as discussed further in the Backup solution implementation[109] page.

The following directions will guide you through a Data Control Tower (DCT) backup, recovery, and migration. They can be used for a "lift and shift" or a "blue-green" deployment. However, the backup and recovery steps are standardized, and can be applied in one-off scenarios as well.

This page refers to the two DCT servers as **initial** (source) and **destination** (target).

6.9.1 Example deployment scenarios

An example of a "lift and shift" deployment could be:

1. Backup the running *initial* server pre-upgrade.
2. Upgrade the *initial* server to the desired version and confirm functionality.
3. Backup the *initial* server post-upgrade.
4. Install a new *destination* server as the same *initial* server's post-upgrade version.
5. Restore the *initial* post-upgrade backup to the *destination* server.
6. Restart the *destination* services and confirm functionality.
7. Shutdown the *initial* server or maintain it for further testing.

An example of a "blue-green" deployment could be:

1. Backup the running *initial* server.
2. Install a new *destination* server with the same *initial* server's version.
3. Restore the *initial* backup to the *destination* server.
4. Restart the *destination* services and confirm functionality.
5. Upgrade the *destination* server to the newer version and confirm functionality.
6. Redirect traffic to the *destination* server.
7. Shutdown the *initial* server or maintain it for a future blue-green deployment.

6.9.2 Prerequisites

1. The *initial* (source) DCT server is up and running.
   a. Referred to as `svr_source` in any CLI commands.
2. The *destination* (target) DCT server is installed in a separate Kubernetes cluster.

---

[109] https://cd.delphix.com/docs/latest/backup-solution-implementation

a. It is the same version as the *initial* DCT server when the backup is taken.

b. Referred to as `svr_target` in any CLI commands.

3. Ability to share backup files from *initial* to *destination* environments.

4. Sufficient access to perform various `kubectl` commands on both the initial and destination clusters.

6.9.3 Backup instructions

Ensure the *initial* DCT server is running, to take a successful backup. Then, run the following commands:

```
kubectl cp <srv_source-gateway-pod>:/data gateway_data --namespace <srv_source-
namespace>
```

```
kubectl cp <srv_source-masking-pod>:/data masking_data --namespace <srv_source-
namespace>
```

```
kubectl cp <srv_source-virtualization-app-pod>:/data virtualization_app_data --
namespace <srv_source-namespace>
```

```
kubectl exec -it <srv_source-database-pod> --namespace <srv_source-namespace> --
pg_dumpall -U postgres > postgres_db_all.sql
```

This will create four files: `gateway_data`, `masking_data`, `virtualization_app_data`, and `postgres_db_all.sql`:

- `gateway_data` is the gateway pod's persistent volume containing encryption keys and various other configuration information.
- `masking_data` is the masking pod's persistent volume containing various configuration information.
- `virtualization_app_data` is the virtualization-app pod's persistent volume containing various configuration information.
- `postgres_db_all.sql` is a complete database backup.

6.9.4 Restore instructions

Ensure the *destination* DCT server is running to restore successfully. In addition, make the `postgres_db_all.sql`, `gateway_data`, `masking_data`, and `virtualization_app_data` files available to the *destination* cluster in the subsequent steps.

Then, run the following commands:

```
kubectl cp gateway_data <srv_target-namespace>/<srv_target-gateway-pod>:/data
```

```
kubectl cp masking_data <srv_target-namespace>/<srv_target-masking-pod>:/data
```

```
kubectl cp virtualization_app_data <srv_target-namespace>/<srv_target-virtualization-
app-pod>:/data
```

```
kubectl cp postgres_db_all.sql <srv_target-namespace>/<srv_target-database-pod>:/tmp
```

```
for i in app bookmarks data-library jobs masking virtualization
{
    kubectl exec -it <srv_target-database-pod> --namespace <srv_target-namespace> --
psql -U postgres -c "drop database \"$i\" with (FORCE)"
}
```

```
kubectl exec -it <srv_target-database-pod> --namespace <srv_target-namespace> -- psql
-U postgres -f /tmp/postgres_db_all.sql
```

Finally, delete and restart the DCT pods:

```
for i in `kubectl get pods --namespace <srv_target-namespace> | awk '{print $1}' |
grep -v jobs-cleaner | egrep "gateway|data-library|jobs|data-bookmarks|masking|
virtualization-app"`
{
    kubectl delete pod $i -n <srv_target-namespace>
}
```

> After deleting the pods, Kubernetes will automatically recreate them and absorb the new
> database backup, and gateway volume data.

6.9.5 Additional environment configuration

The HELM chart's `values.yaml` contains information specific to your environment, such as certificates, hostname, or resource limits. You can update this information before or after the migration process. The standard installation and configuration process can be followed to update these values.

6.10 Exporting DCT logs to Splunk

6.10.1 Overview

> ℹ️ This feature is not available for DCT Appliance (OVA) deployments.

This article provides some tips for configuring DCT (running on Kubernetes) to send logs to Splunk and extract useful information in Splunk.

6.10.2 Setting up a Splunk instance

Authenticate with Splunk via the web portal and install the third-party Monitoring Kubernetes[110] app directly via the Splunk UI, then enable HTTP Event Collector in Splunk and save the HTTP Event Collector token for future use.

6.10.3 Enable Splunk log forwarding

Once the Splunk instance is setup, follow the instructions to install Splunk logic in the Kubernetes cluster to forward logs to Splunk. This blog post[111] is a useful resources to understand the log collection and configuration options.

```
git clone https://github.com/splunk/splunk-connect-for-kubernetes.git
cd splunk-connect-for-kubernetes/helm-chart/splunk-connect-for-kubernetes
edit values.yaml
```

Edit values.yaml, at the minimum the host property (hostname of the Splunk collector) and token (of the HTTP Even Collector) must be set.

---

110 https://splunkbase.splunk.com/app/3743
111 https://faun.pub/logging-in-kubernetes-using-splunk-c2785948fdc0

```
global:
 logLevel: info
 splunk:
   hec:
     # host is required and should be provided by user
     host: <insert-splunk-http-event-collector-hostname-here>
     # port to HEC, optional, default 8088
     port:
     # token is required and should be provided by user
     token: <insert-token-here>
```

Install the helm chart and and after a few minutes DCT logs will be visible in Splunk.

```
helm install splunk-connect-for-kubernetes . -f values.yaml --set splunk-kubernetes-
logging.fullnameOverride=splunk-logging
```
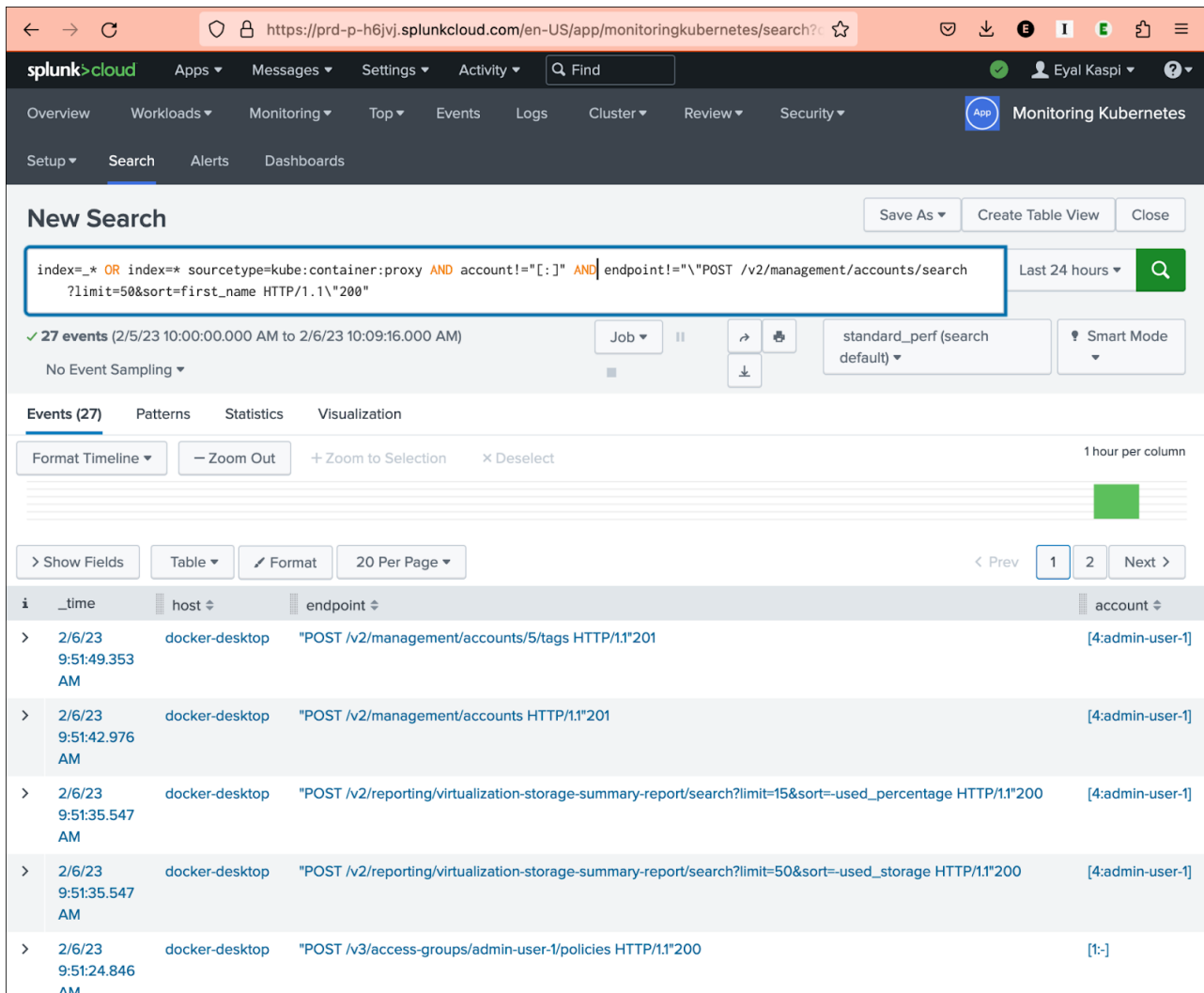
6.10.4 Search for events in Splunk

In the Splunk Cloud UI, via the "Monitoring Kubernetes" App, you can "search" for data sent by Kubernetes, as exemplified in the screenshot below. The bootstrap API key can be found as shown.

The example screenshot below shows a search for \"nginx\", with use of the "extract new fields" wizard on the bottom left, which has Splunk parse the Nginx access logs. A regexp is used to name some of the fields like ipaddress, endpoint, accountid, etc. The example runs a search to return API requests associated with accountid.

6.11 Generating a support bundle

> ℹ️ For DCT Appliance (OVA) deployments, use the setup UI for support logs in the Creating support logs[112] page.

6.11.1 Find the `collect_bundle.sh` script

- The support bundle tar file is available on the downloads site[113].

---

112 https://cd.delphix.com/docs/latest/creating-support-logs
113 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

- Once the file is downloaded, untar the file to find the script.

```
dlpxuser@delphix:~/test$ tar -xzvf dct-support-bundle-1.0.1.tar.gz
x ./
x ./collect_bundle.sh
x ./README
x ./VERSION
```

6.11.2 Execute the `collect_bundle.sh` script when DCT is running in Kubernetes

- Transfer the `collect_bundle.sh` script to the machine where you have permissions to execute `kubectl` commands against the DCT pods.

⚠️ You must install the bash shell to generate a DCT support bundle, if it is not already.

- Execute the `collect_bundle.sh` script, which assumes a Kubernetes deployment by default. The script may need to run with `sudo` if root permissions are needed to run the `kubectl` commands.
- If the namespace is not the default "dct-services", use the `-n` flag and pass the correct namespace.

```
dlpxuser@delphix:~/test/tools/support-scripts/$ ./collect_bundle.sh -n
<custom_namespace>
....
DCT support bundle collection started at Thu Jun 22 12:35:05 EDT 2023
Collecting logs from all DCT containers...
....
```

6.11.3 Execute the `collect_bundle.sh` script when DCT is running in Docker-Compose

- Transfer the `collect_bundle.sh` script to the machine where you have permissions to execute `docker` commands against the DCT Docker-Compose application.

⚠️ You must install the bash shell to generate a DCT support bundle, if it is not already.

- Execute the `collect_bundle.sh` script with the `-d` parameter. The script may need to run with `sudo` if root permissions are needed to run the `docker` commands.

```
dlpxuser@delphix:~/test/tools/support-scripts/$ ./collect_bundle.sh -d
...
DCT support bundle collection started at Thu Jun 22 12:35:05 EDT 2023
Collecting logs from all DCT containers...
....
```

6.11.4 Find the generated support bundle tar file

The resulting support bundle will be located at `dct-support-****.tar.gz`, inside the current directory.

```
dlpxuser@delphix:~/test$ ls -ltr
total 316
-rw-r--r-- 1 65436 staff 104189 Feb 17 08:52 dct-support-<current_timestamp>.tar.gz
```

The support bundle tar file contains the following information:

- DCT logs for all of the containers.
- A java heap dump, .hprof, if one exists.
- A java thread dump and memory stats.
- The output of **docker stats**, if running in Docker-Compose.
- The output of **cpuinfo**, **meminfo**, and **mpstat** for each container, if running in Kubernetes.
- The output of **kubectl get pods -o json** for each container, if running in Kubernetes.

ⓘ
- The collect_bundle.sh generates a support bundle from a DCT engine running in Docker or Kubernetes.
- The resulting support bundle will be at `./dct-support-****.tar.gz` inside the container.
- The user must have privileges or permission to execute the **docker** or **kubectl**, commands in order to generate the support bundle.

7 Data governance

7.1 DCT administration

DCT delivers a management layer on top of all connected Delphix engines through surfacing object inventories, instrumenting all common Delphix operations, delivering a business metadata layer with tagging, and using those tags to drive attribute-based access control. This provides the ability for administrators to deliver a highly curated and secure Delphix experience for automation and end-users.

This section contains configurations handled under the **Admin page** in the DCT interface.

7.1.1 Operations in DCT
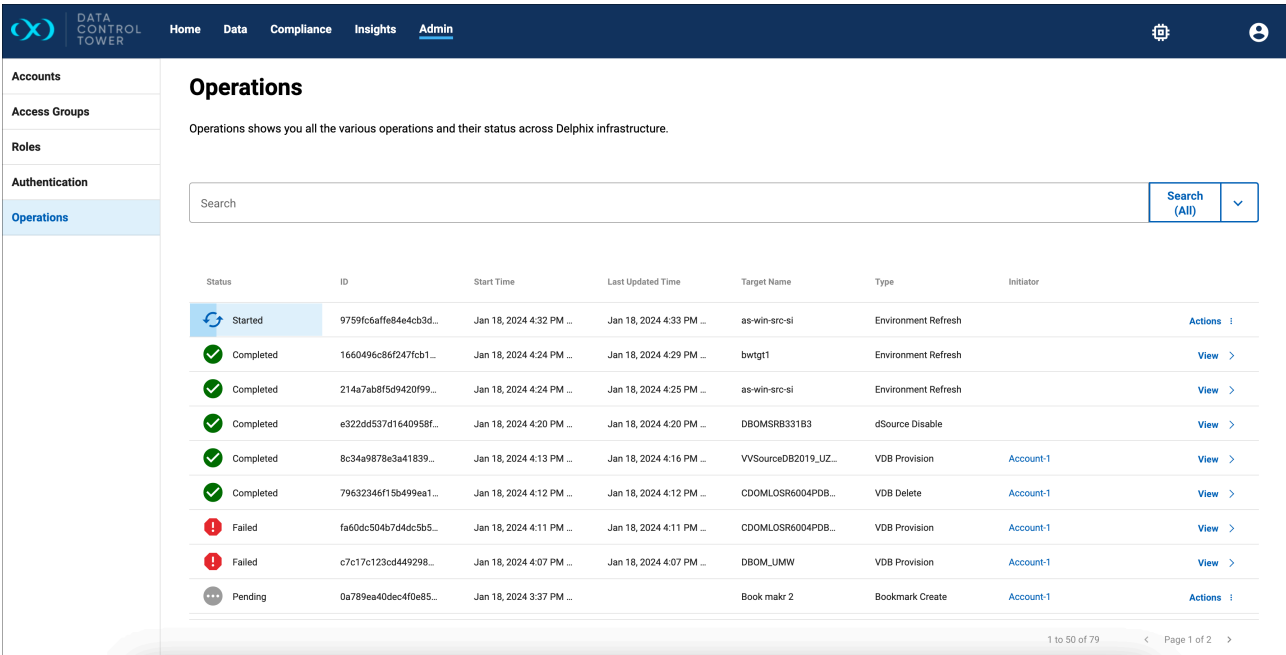
The Operations page is available under the **Admin** menu, showing a comprehensive list of all the DCT Jobs across Delphix infrastructures. It only shows operations that the user has access to.

The Operations Monitoring feature allows users, such as Database Administrators (DBA), Developers, CISOs, and Database Managers, to monitor system operations related to data virtualization and masking. Users can view all operations and their statuses (completed, in progress, or error) on the Operations page.

This page will display operations based on the user's **read permission** on the associated object. Click on a specific operation to view additional details, including the related object name, the operation type, the current status, a progress indicator (if in progress), and any additional metadata related to the operation.

> The Operations page is accessible from **Administration** > **Operations** or the **View All** button on the Operations Monitor Bar, which is described below.

Relevant details for the operation are shown, such as status with a graphical progress bar for running jobs, type of job, target id (refers to the id of the object on which the operation has been performed), start time, last updated time, etc. In versions 10.0.0 and above, a column shows the initiator of the operation.

The capture **above** represents an **admin user** view of the Operations page.

The capture **below** represents a **non-admin user** view of the same page.
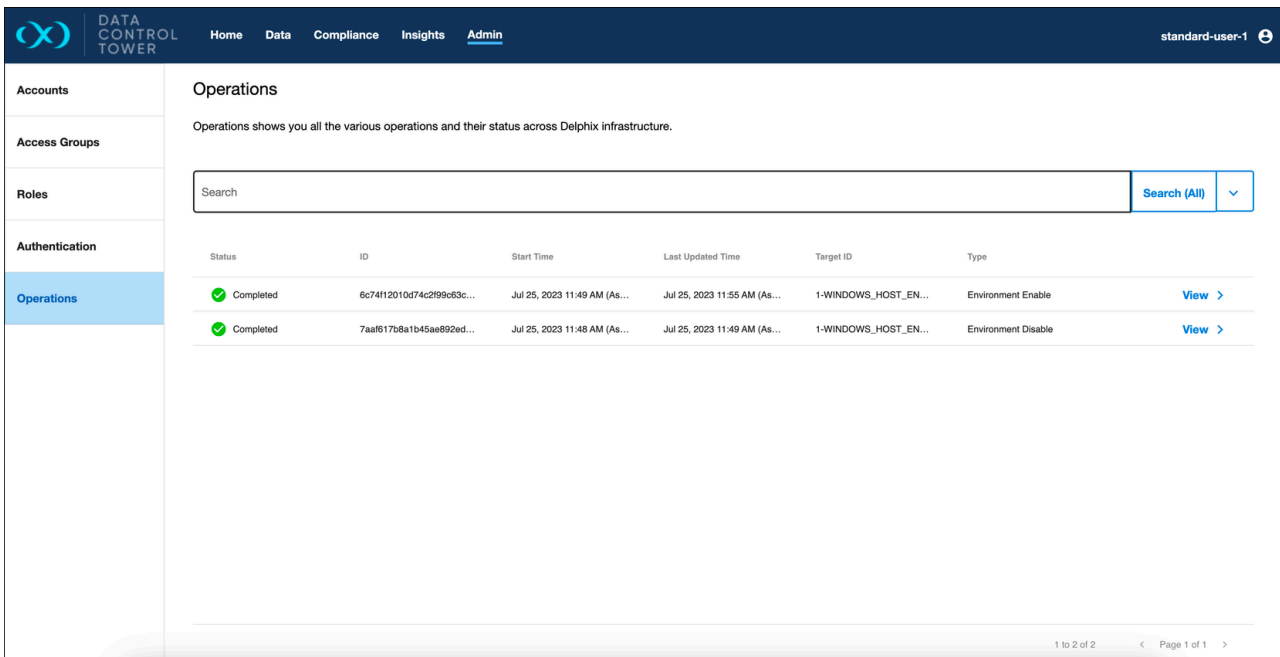
Selecting a job will navigate to the Operations details page, which shows further details for that job, in addition to a progress bar. The details include target id, error, and warning logs. Continuous Data operations are the jobs initiated by the engine, as shown in the action sidebar.

The system will execute operations either manually by users or by automated tasks. If an operation encounters an error, the error details can be viewed on the Operation Details page. Notifications based on operation status changes will be covered in future brief and are not in the scope of this product brief.
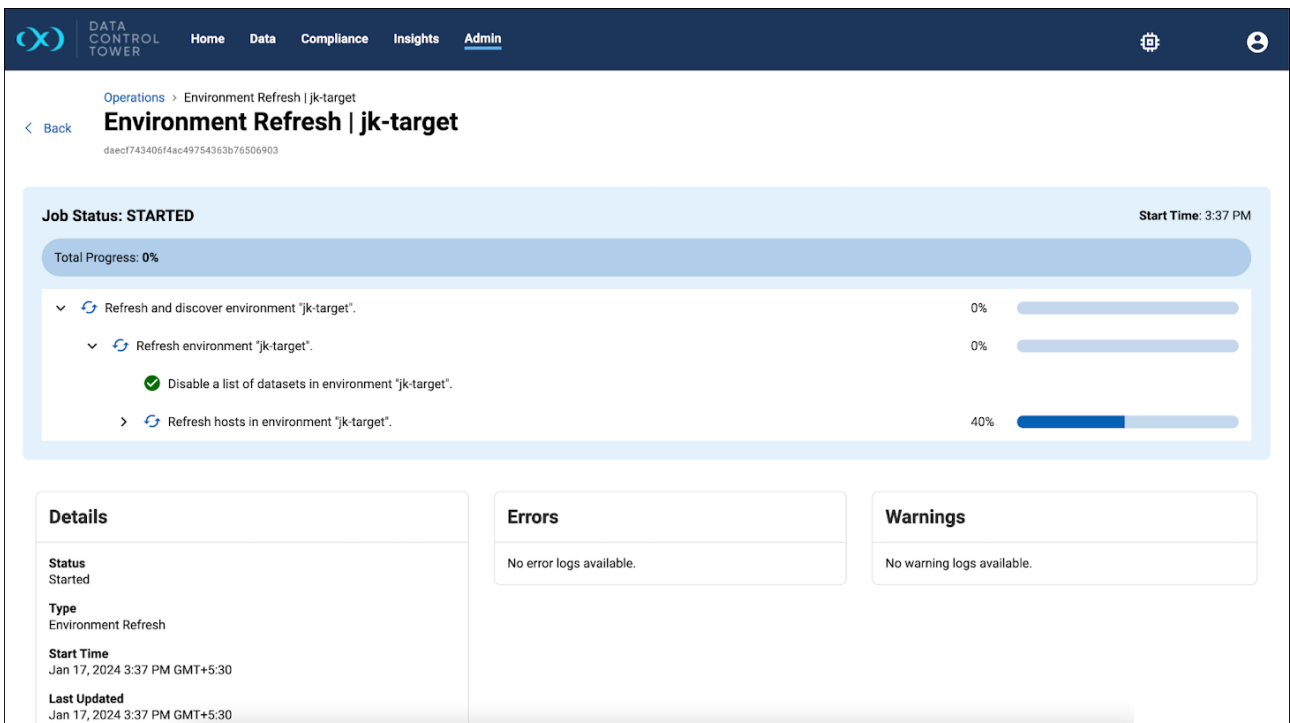
7.1.1.1 Operations Monitor Bar

The Operations Monitor Bar appears at the bottom of pages to display user-initiated operations, with those completed being auto-dismissed. Operations with errors or in progress remain until the user dismisses them from the bar. In-progress operations display the operation type and object name. By default, the Operations Monitoring Bar toggle button will be available on all pages of the application.

When logging into the application, the Operations Monitoring Bar will be hidden by default. There will be two triggers for showing it:

- Click on the icon (Operations Monitor Bar toggle button) in the top right.
- Initiate an action like VDB refresh or compliance operation execution.

The bar has a **View All** button that navigates to the full Operations page and an actions menu, represented by a vertical "more" icon, that displays available actions for that operation. Visibility of the bar can be toggled on from the "memory" icon in the top right corner.

The Operations Monitoring Bar will show the following:

- Operations initiated by the user.
- Operations initiated during the current session.
    - Initially the Operations Monitoring Bar will always be empty upon login.
- Operations marked as **Monitor** by the user in the full Operations page.
- Operations currently running and failed.

7.1.1.1.1 Additional notes

- Close and reopen the Operations Monitoring Bar as needed by using the toggle button located in the top right corner (within the top banner).
- The successfully completed operations will auto dismiss. In other words, when any operation completes, the status changes from RUNNING to COMPLETED and it will be auto-dismissed/auto-removed from the Operations Monitor Bar. For any other status, such as CANCELED, ABANDONED, FAILED, SUSPENDED, TIMEOUT etc., auto dismiss will not occur.
- Click the three-dot menu icon on any operation in the Operations Monitor Bar to **dismiss** and remove any operation from the Operations Monitor Bar by selecting this action from the
- When logging out and logging in again, the Operations Monitoring Bar will revert to being empty. The application will not retain the state of the Operations Monitoring Bar between user sessions.
- The **View All** button will navigate users to the Operations page.
- The **View Details** button will navigate users to the Details page of the specific operation.

7.1.2 Tags

7.1.2.1 Tags management

DCT powers data governance with tags. These key-value pairs can be used to associate any business-level data with any Delphix object, to drive greater intelligence in automation, administrative workflows, data access, and reporting. Advanced search for tags is available.

Tags are individual attributes on every object exposed in DCT; from VDBs, to compliance jobs, and even users. There are no limits on tag count per object and character limits are set for flexibility to enable robust grouping.

7.1.2.2 Administrative tagging

Tags can be managed from the UI by selecting "View Tags" for a particular object on its global list page. The below example shows the tag configuration screen for a dSource "AGDatabaseSQL2016" and multiple tags have been added to characterize that particular object:

DCT tags enable complex searching to enable intelligent reports. A demonstration using the above example dSource and using expression-based search to filter dSources with the {App Team: Alpha} tag.
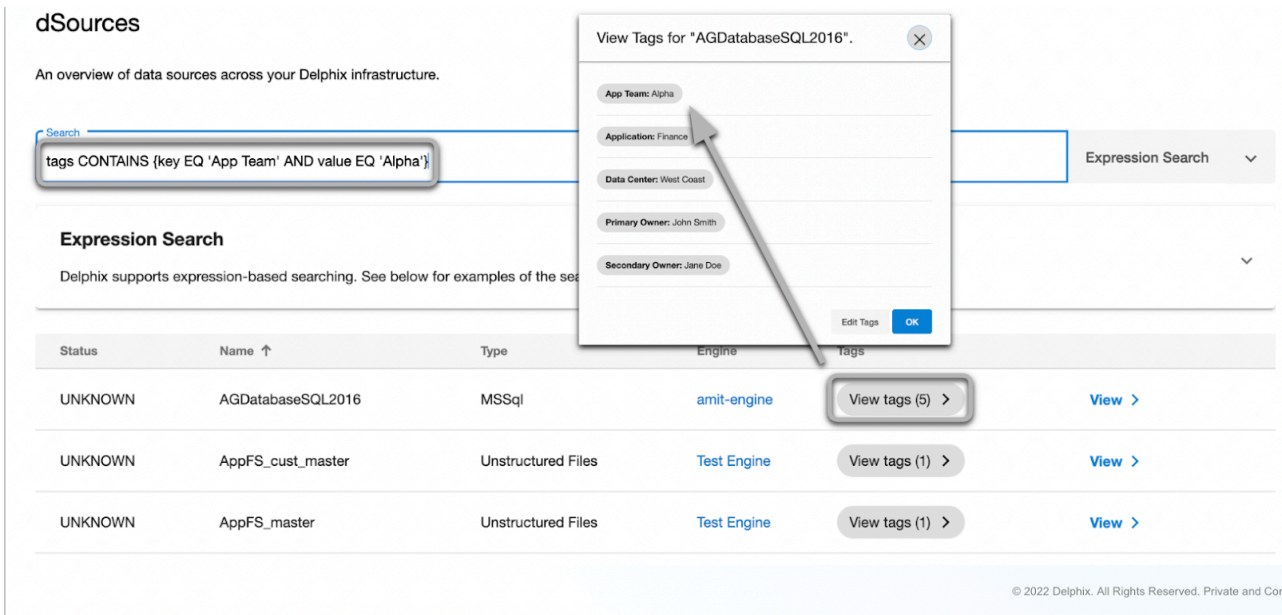
7.1.2.3 Tags powering attribute-based Access Control

Tags also power the DCT permissions system for both Accounts (users) and Role Scopes (object entitlements). The below example shows an Access Group (Alpha Team) with the Accounts tab on display. Notice that the accounts tab has {App Team: Alpha} under "tag mapping", which automatically attributes any users with the {App Team: Alpha} tag.
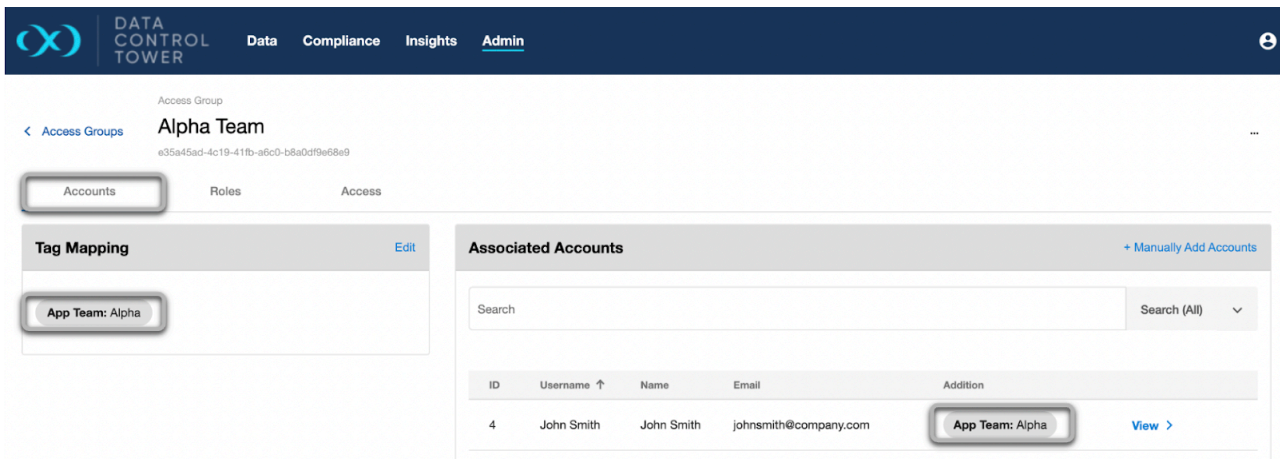
The same goes for Scoped Roles under the "Roles" tab. The Alpha Team role has been mapped to the {App Team: Alpha} tag and all dSources with that same tag are automatically attributed.
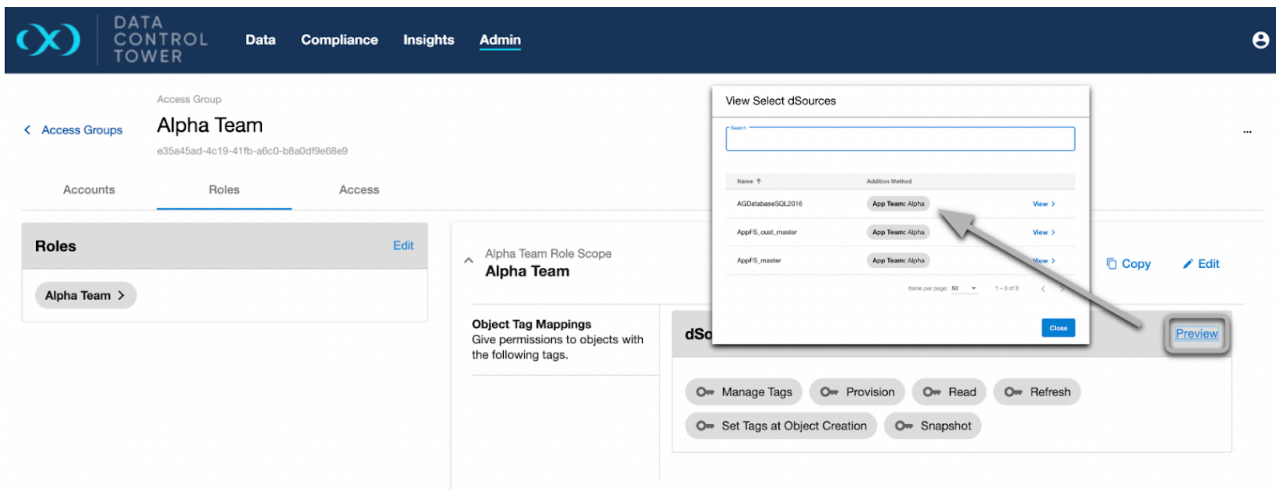
7.1.3 Access Control

Data Control Tower (DCT) fundamentally changes how application teams are governed across the Delphix Platform to ease expansion and management burden. Previously, Delphix administrators were focused on managing individual user-level access on each engine. This made it difficult as teams increased their data set requirements. This inevitably led to more time managing engine access and not rolling out test data management (TDM) practices. Now with DCT, all users are managed and access their data sets through a centralized server. This makes it easier for administrators to manage the Delphix Platform and application teams to utilize the self-service capabilities.

To take advantage of DCT's new capabilities, Delphix administrators will implement a centralized Attribute Based Access Control (ABAC) model. This is performed by consolidating permission management from the engines to DCT, implementing Access Group policies, and assigning Object tags. The flexibility of this approach ensures your company's required security model can be maintained or even further refined.

The below picture attempts to show the shift in access models. In the original Engine Model, the engines were isolated from one another. No access control mechanisms were shared between Engines. In the DCT Model, Delphix administrators will manage applications teams directly through DCT. Those application teams will log directly into DCT. Only administrators will log into the Engines for advanced usage.
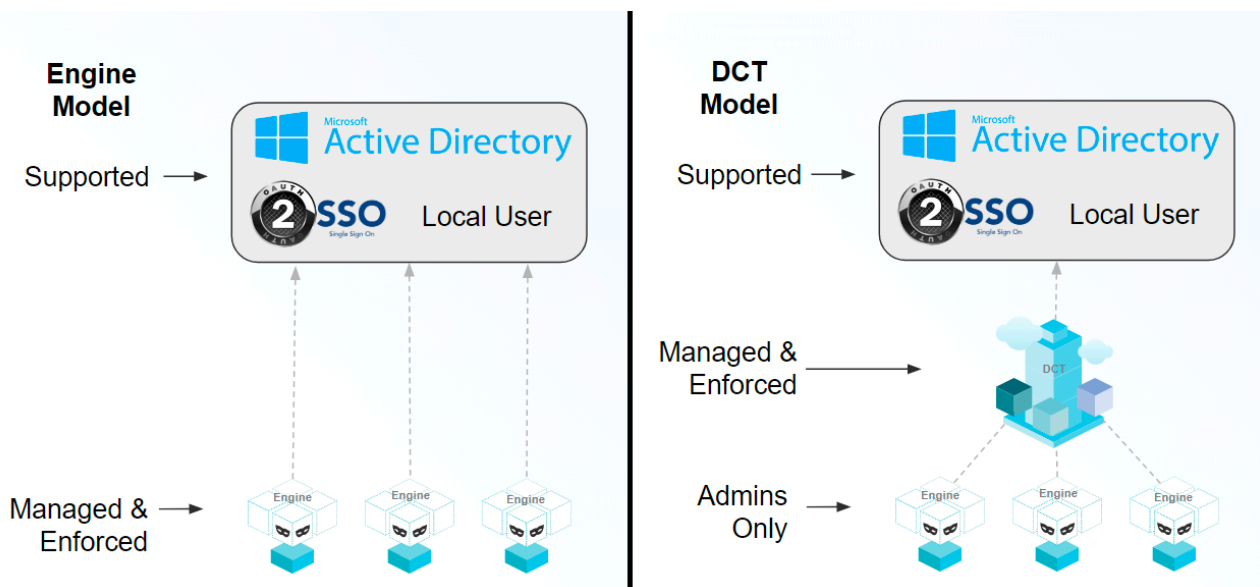
7.1.3.1 Access model overview

Data Control Tower implements a model that you might find in other types of software called Attribute Based Access Control (ABAC). This model is incredibly flexible but requires detailed configuration to perfect your use cases. In our model, there are four entity types which are defined below. Understand each entity as they are the foundational blocks of DCT's ABAC model.

Entity	Description	Managed By
Accounts (aka Users)	A single or shared user who can authenticate with DCT (UI or API).	Create manually or via Identity Provider (IdP), such as SSO or LDAP. Accounts are independent of Delphix Engines.
Access Groups	A collection of accounts that share one or more characteristics, such as a Team or Permission set. Equivalent to an Active Directory group.	Manually created. Populated manually or via the 'login_groups' tag.
Roles and Permissions	The collection of read, write, and delete permissions forms a reusable, named role.	Some roles are provided out of the box, but Admins can build their own from the available permissions. Individual permissions are immutable.
Objects	Units, such as VDBs, Bookmarks, and Environments, that are managed across the Delphix Platform.	Automatically identified by DCT from the connected engines. Assigned to Roles via various models. The CD and CC Engines supply these objects.

Each entity is linked to another through manual or automated assignment. A manual (or direct) assignment is a good approach for early implementations. However, it can be challenging to maintain as teams grow. As an alternative, Tagging is suggested as it performs automatic assignments based on your custom configuration. The below diagram shows how each entity is linked together. The directions below start with Accounts creation to Access Groups with Role assignments and finish with Object mappings.
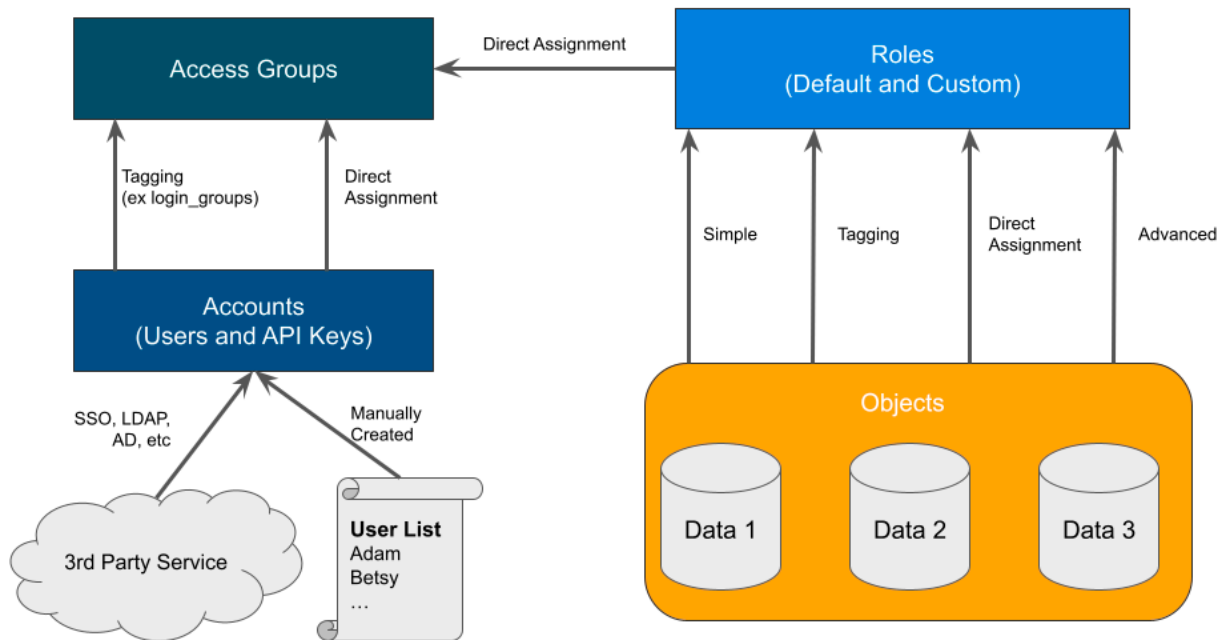
Understanding your team structure is imperative to identify the best access model. Usually, organizations have existing groupings defined in their Identify Provider (IdP). These groups are typically organized in one of two ways (a) a team dedicated towards a central goal (such as a product development team) or (b) a group of individuals with similar permissions (such as Security Administrators). Understanding the purpose of each group should be a guide in how the Roles and Permissions are designed. For example, the Alpha product development team might have full permission to manage existing VDBs and create new bookmarks for their team's "Alpha" objects. On the other hand, Security Admins might have sweeping read and disable access across the entire platform to ensure compliancy. Iterating through each Access Group and designing custom, but re-useable roles, based on the Principle of Least Privilege[114], will produce a streamlined rollout.

7.1.3.2 Accounts: Manual, LDAP/AD, or SSO/SAML

Goal: Import or create user accounts. Complete either the Manual or LDAP/SOO configuration.

---

114 https://en.wikipedia.org/wiki/Principle_of_least_privilege
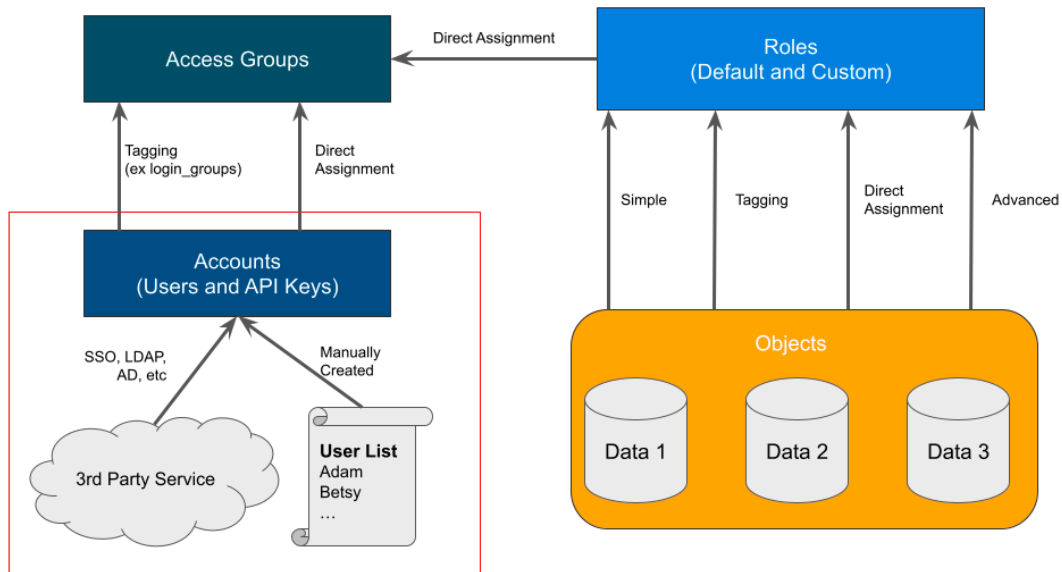
Manual (User List)

Navigate to **Admin** > **Accounts**, click the "+ Account" button, and complete the form.
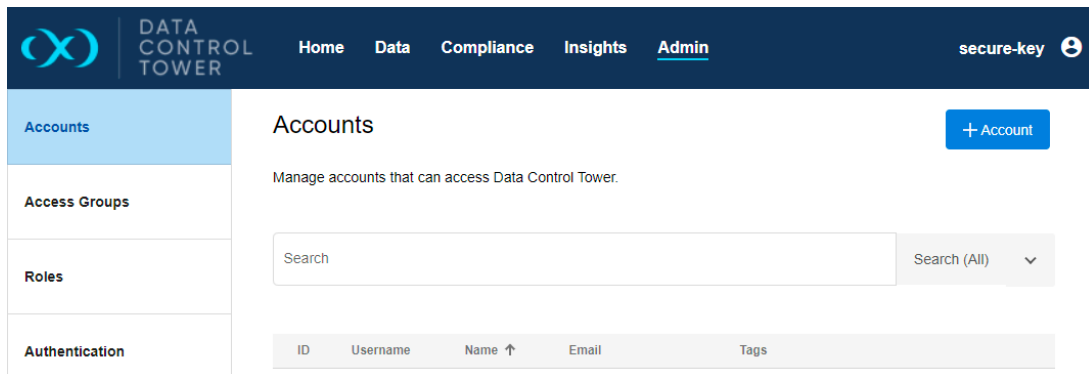
Manual accounts are great for testing user access or providing a service account. Take note of the checkboxes by which you want this user to access DCT.
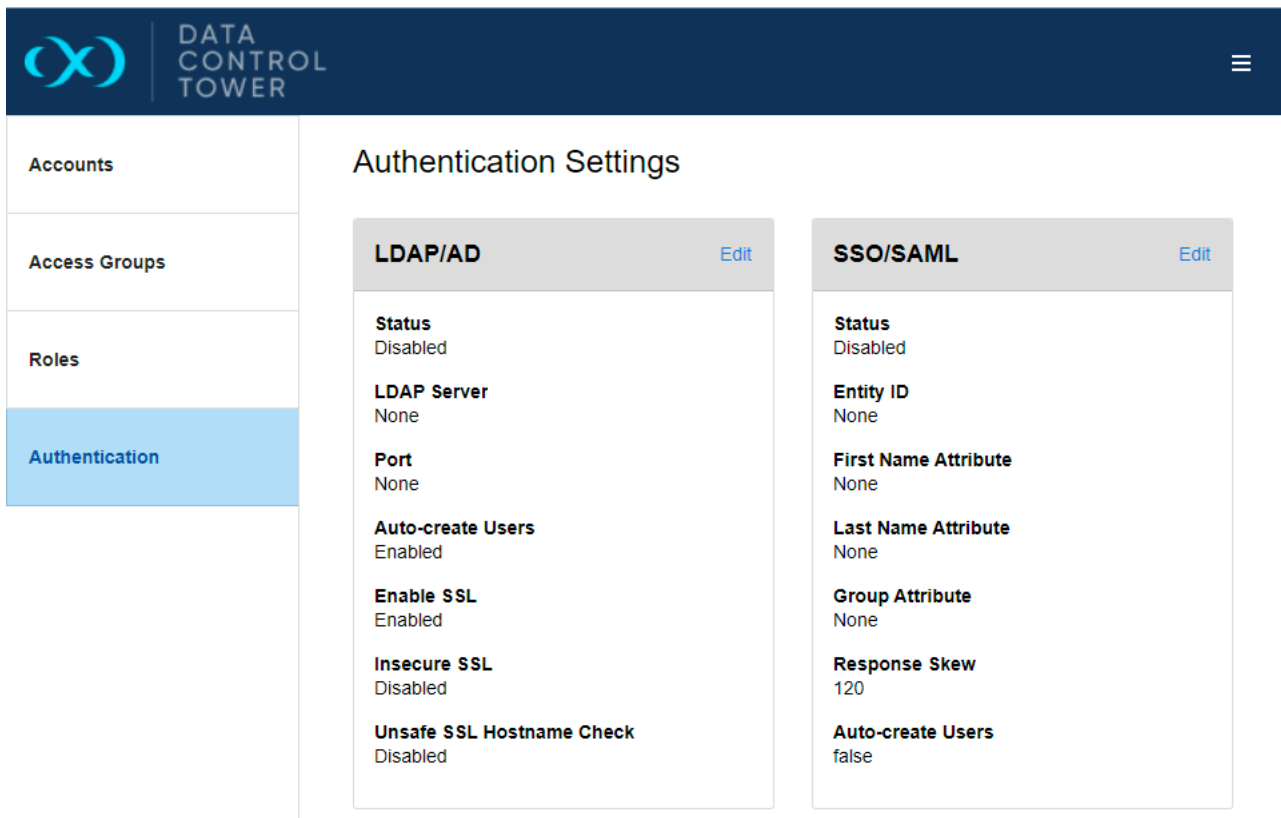
When you have specified all required values, select the "Create Account" button. By default, this user will have no permissions.

LDAP/Active Directory or SSO/SAML (3rd Party Service)

Navigate to **Admin** > **Authentication**, click "Edit" for either LDAP/AD or SSO/SAML, and complete the form. Ensure "Auto-create Users` is enabled. It can be disabled at any time.

If you need guidance on how to configure, follow the directions here:

- LDAP/AD Directions
- SSO/SAML Directions

Once configured, Accounts will be automatically created when a user successfully logs in.

> This is functionally different from the old Engine model. Previously, the Account was created manually before they could log in.

Recommended: LDAP/Active Directory Domains

It is highly recommended that we also configure group membership during this stage. By defining the metadata attributes in the option Domain fields, DCT can automatically assign users to Access Groups. If configured correctly, you will see an automatically generated `login_groups` tag on recently logged-in accounts. If an Account does not have the tag, then (a) the Domain configuration is invalid, or (b) they should re-login. The `login_groups` tag is the only tag that cannot be specified on an Account manually.

LDAP/Active Directory Domain Groups Directions
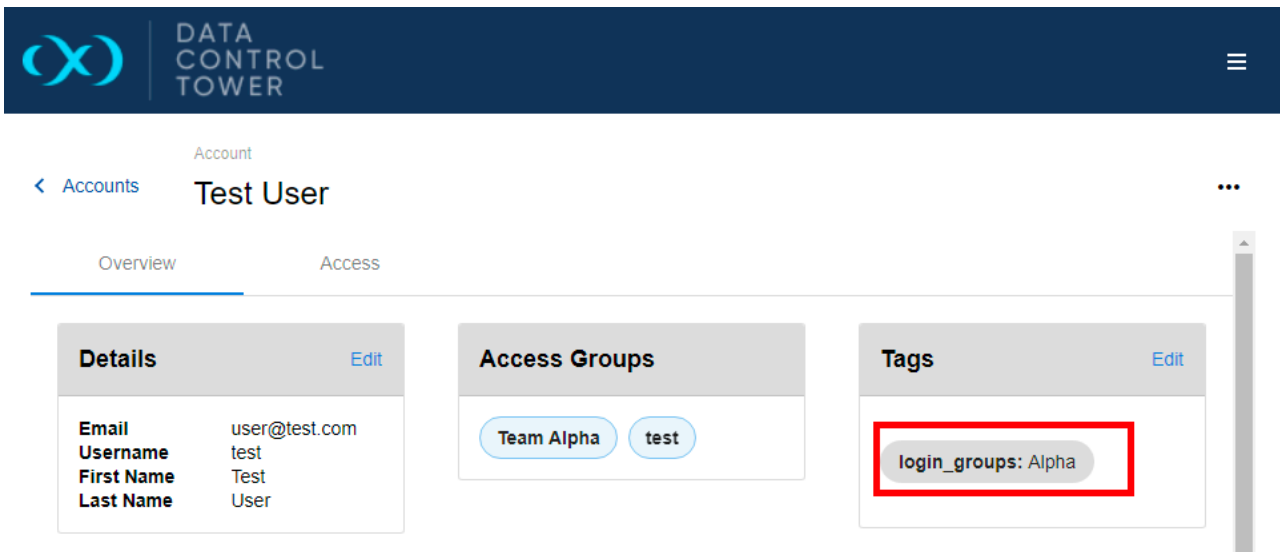
Example Account with the `login_groups` **tag.**

Please test these new accounts out by logging in on another browser. By default, these accounts will not have any permissions and not see anything. In the following steps, we will give them access. In addition, if configured, we'll take advantage of `login_groups` or a custom tag.

7.1.3.3 Access Groups: Creation and account assignment

7.1.3.3.1 Goal: Create an Access Group and assign Accounts directly or through Tags.
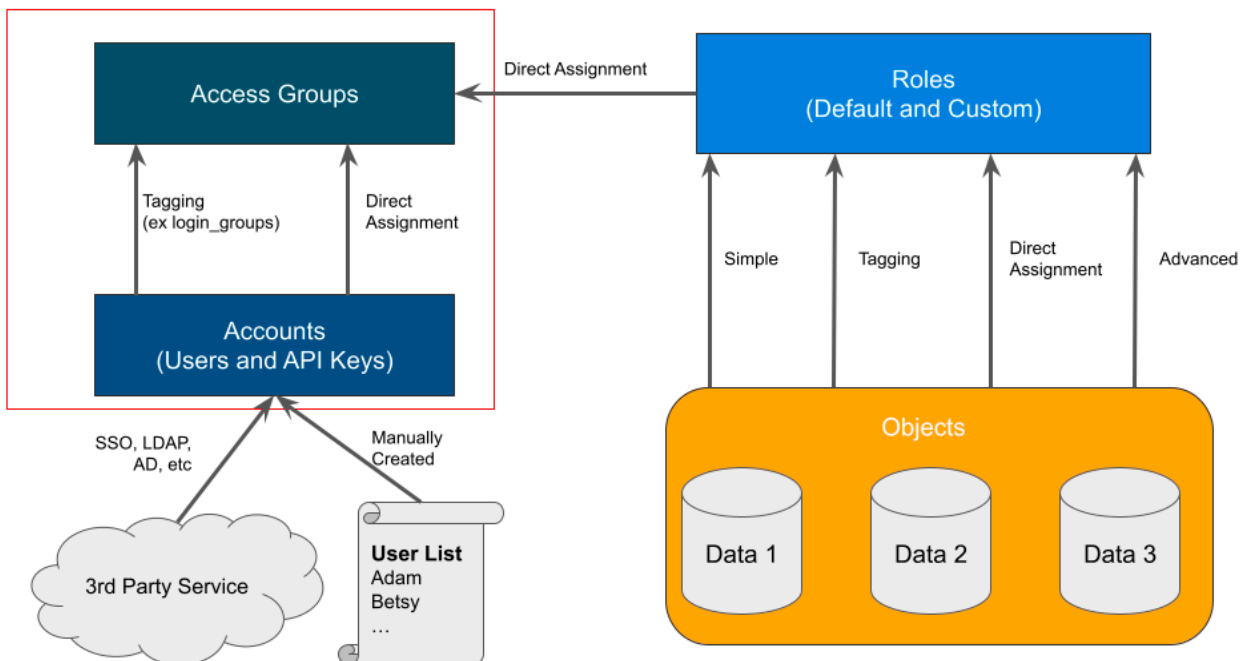
Access Group Creation

Next, navigate to the **Admin** > **Access** Groups tab, select the "+ Access Group" button, and complete the presented form. As described previously, these groups are based on existing teams or users with similar access. If you successfully configured the Active Directory's Domain Groups, you can specify the `login_groups` tag and value here. Or specify a custom Tag, such as "Team: Alpha".

You can also select Roles if you already know which should be applied. Otherwise, ignore it.

Submit the form once you are happy with your new group.

> Unlike an Account, you can specify the `login_groups` tag on an Access Group shown in the picture above.

On completion, you will be presented with a page similar to the one below. Unfortunately, it's empty. Let's add some associated Accounts now.

Manual (Direct) Assignment

Select the "+ Manually Add Accounts" button, select the desired Account, and then "Add Account". Immediately, you'll see it presented in the Associated Accounts list.

This is a good solution for quick management. However, it can be cumbersome as usage grows. Therefore, we recommend tags!

Tag Assignment

First, navigate to the **Admin** > **Accounts** tab and select an existing Account. (Feel free to create another one!) Once selected, add a custom Tag such as 'Team: Alpha". If one already exists on the Account, such as "login_groups", remember it.
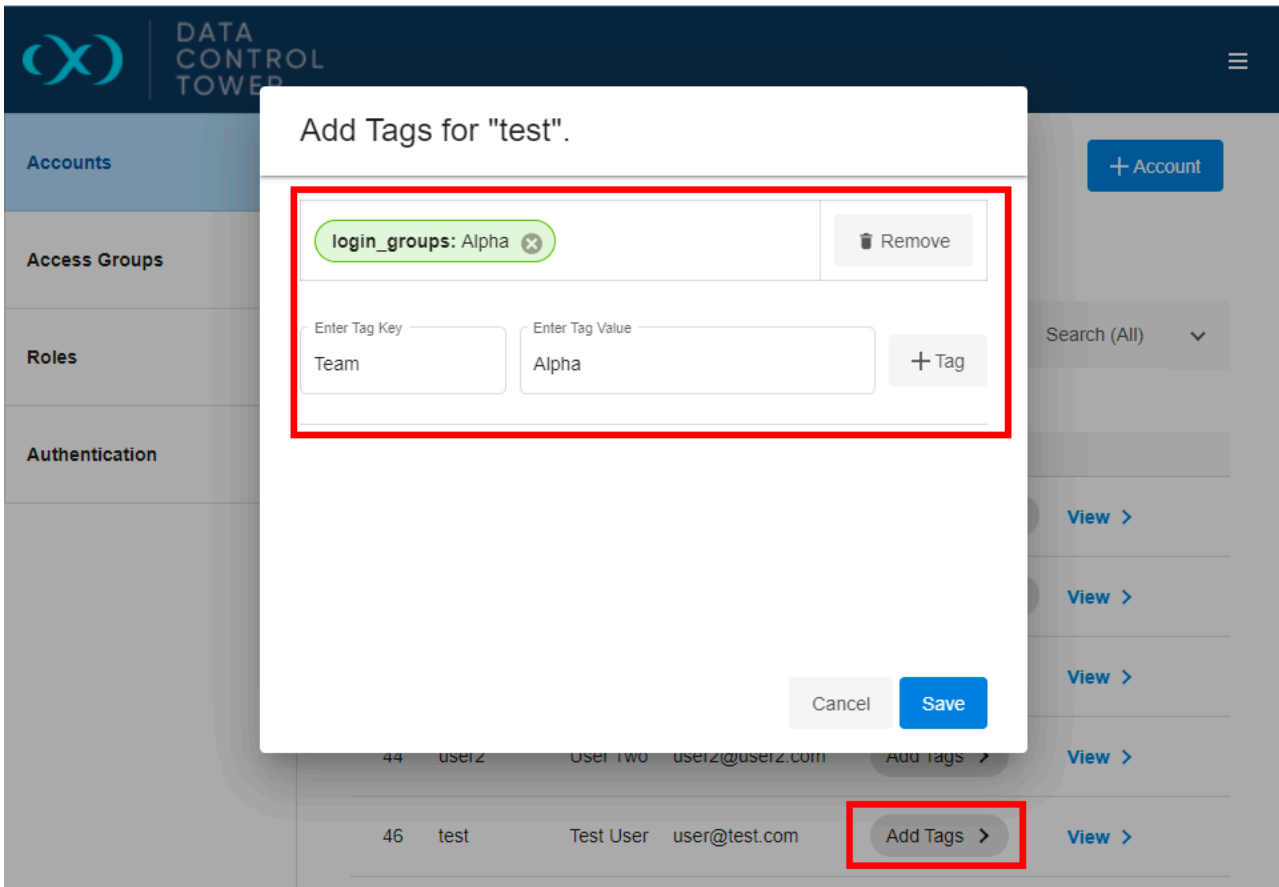
Next, navigate back to the Access Group, select the Tag Mapping's "Edit" button, and specify that same Key: Value pair. It might look similar to the below picture.
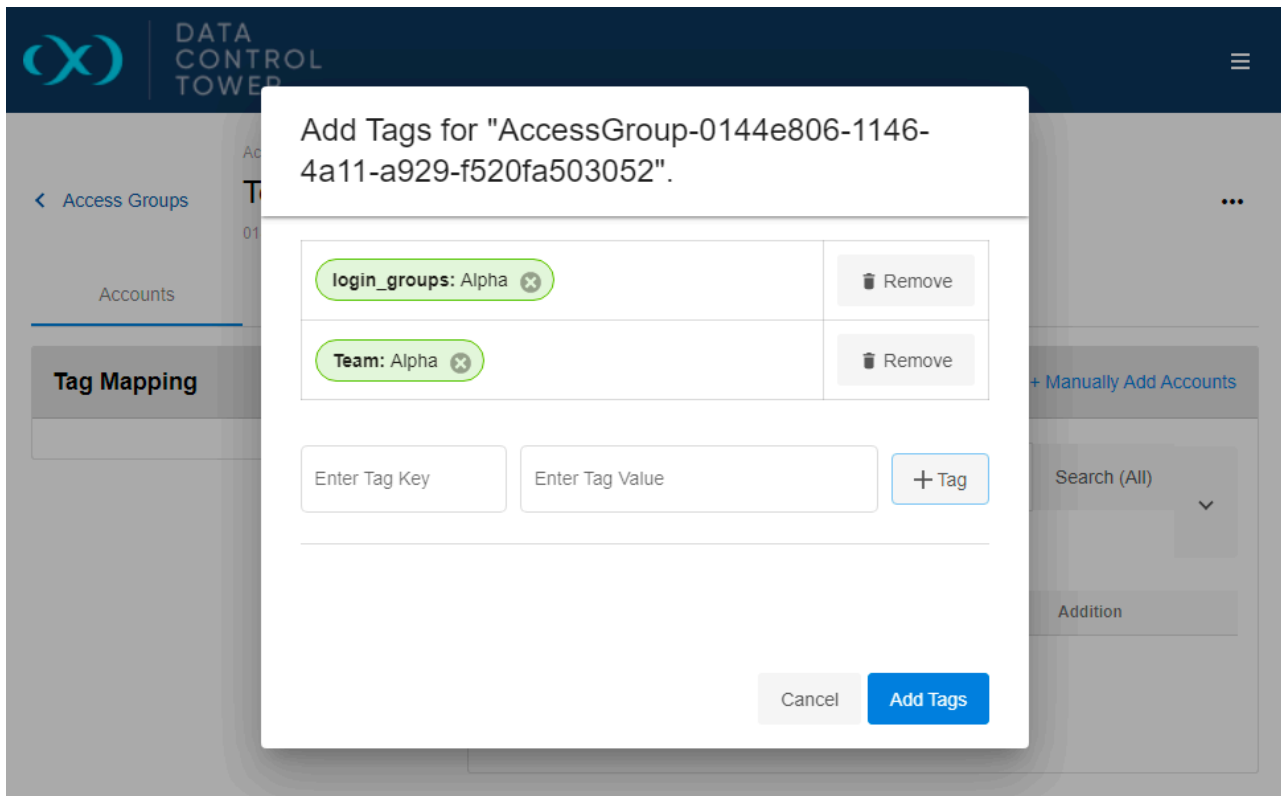
In this example, the "Team: Alpha" and "login_groups: Alpha" were added through the Access Group's Tag Mapping widget. If configured successfully, your Access Group might look similar to the below picture. If you remove the Access Group or Account's tag, you will see Account automatically removed from this listing.

> The "login_groups" tag functions identically to a custom tag within the Access Group. Again, the only difference is that it's automatically assigned to the Account.

 This section taught us how to organize Accounts into different groups. This allows us to keep permission sets separated. Feel free to experiment with new Access Groups, Tags, and Accounts. If you still need additional pointers, review our Access Groups Documentation (see page 164).

7.1.3.4 Roles: Creation and assignment
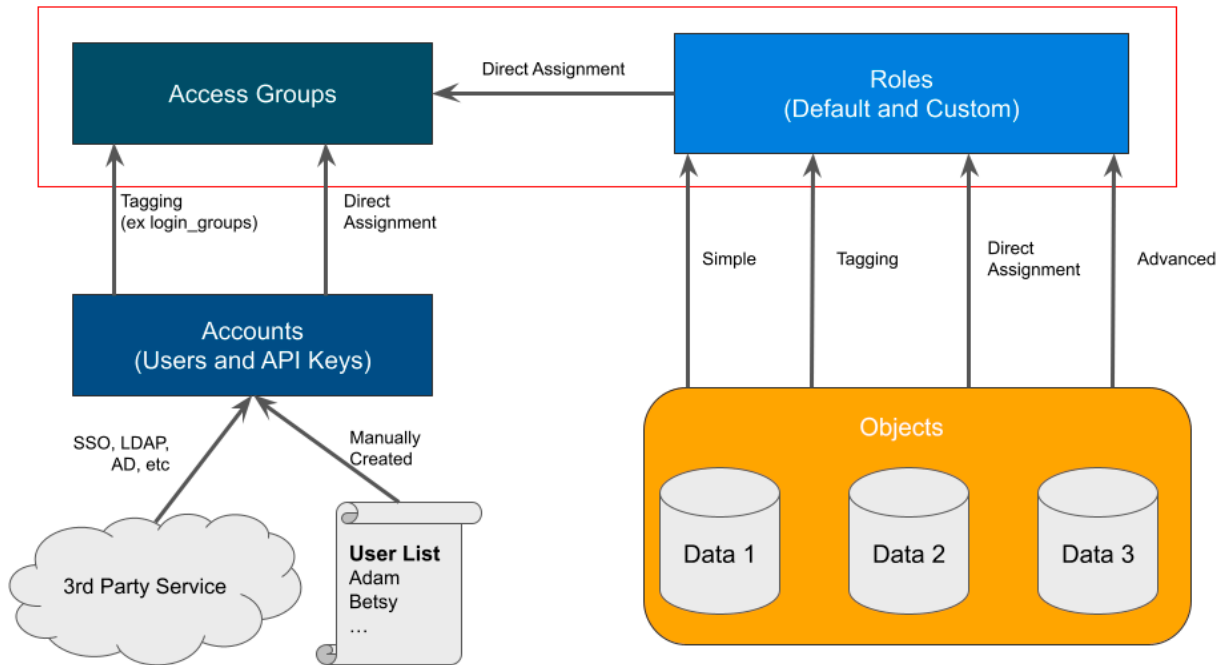
Role Investigation and Creation

Navigate to the **Admin** > **Roles** tab. Here we see a list of DCT's default Roles. Each role has its selection of Permissions, such as Read VDB, Delete Bookmarks, Modify dSources, etc. Select "View" on the "devops" role to see its permissions.
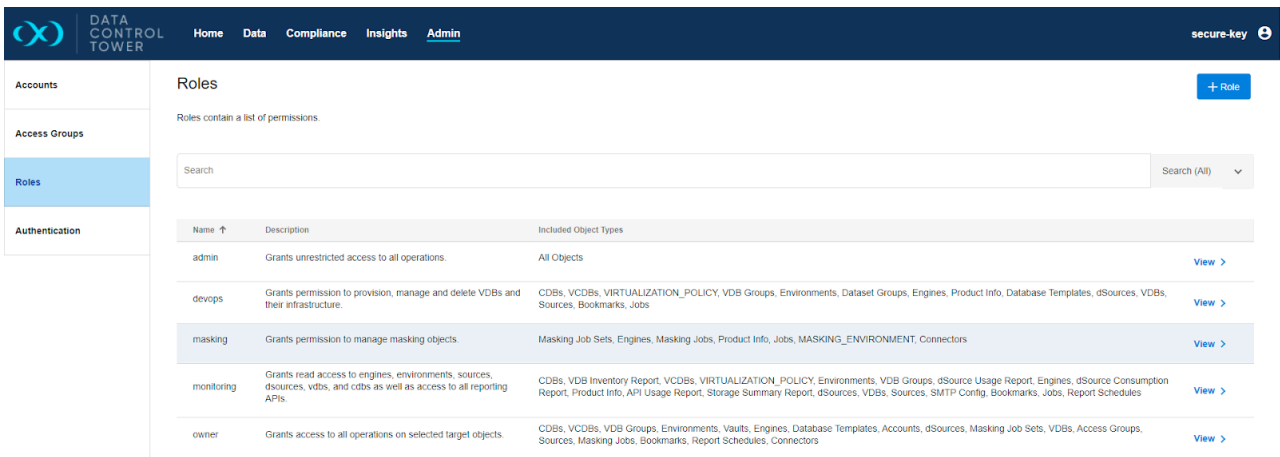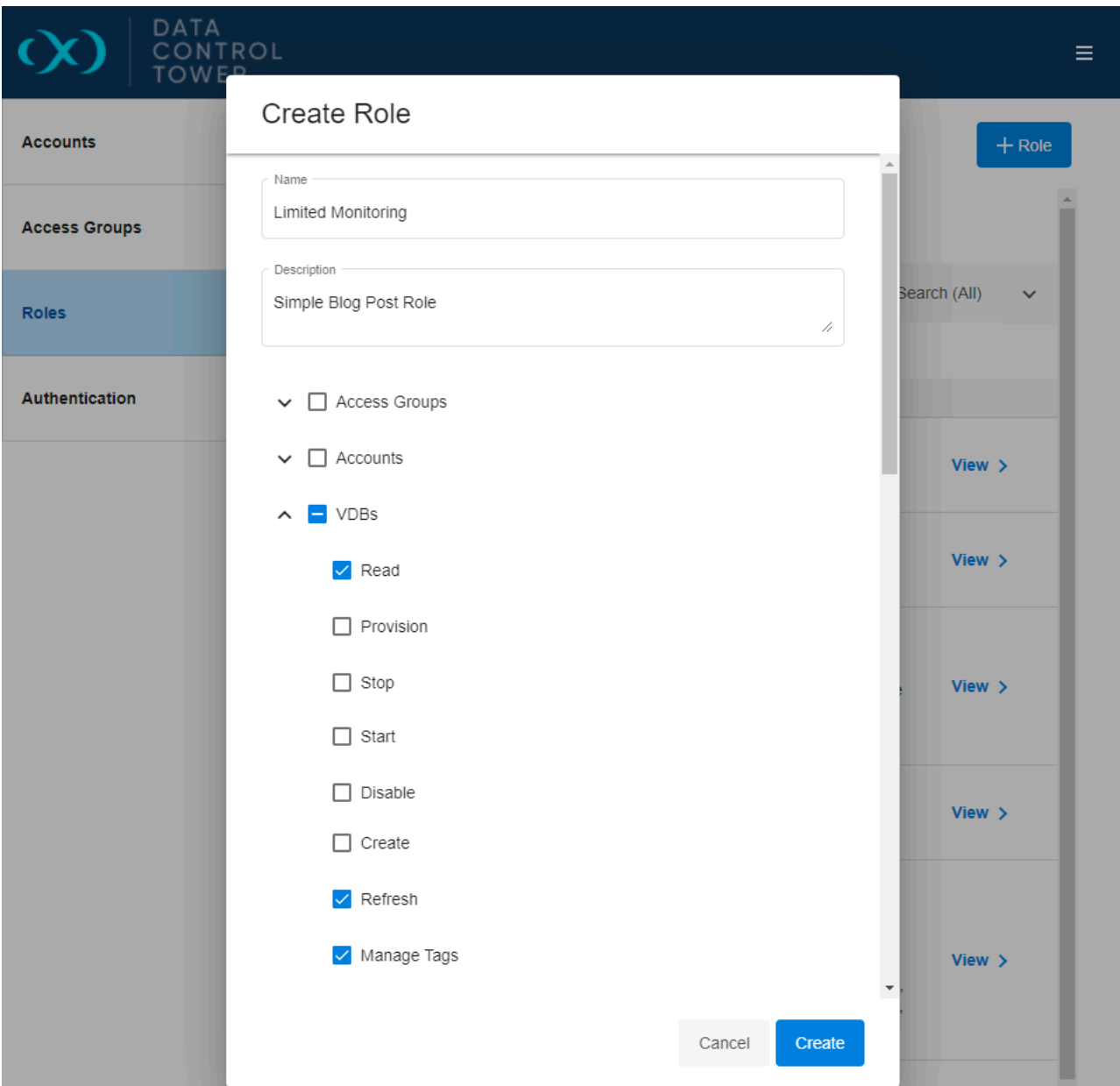
On the left-hand side, you can see a description, the Access Groups it's currently a part of, and any assigned Tags. On the right-hand side, is the complete list of permissions. For example, you can see here that the "devops" role has "Manage Tags" and "Read" permissions on the CDBs objects. These various permissions make up the Role's identity.

> ℹ️ DCT's default roles are immutable.

Role Creation

Now we understand what it's composed of, let's create one. Navigate back to the Admin > Roles tab and select the "+ Role" button. Give the Role a custom name, sample description, and add all the permissions you want. In my simple example, I gave it the "VDBs > Read, Refresh, and Manage Tags" permissions. If you need to grant permission for the entire category, select the header checkbox, such as "Access Groups" or "Bookmarks". If you only want a portion of that Object group, then click the little arrow icon to open up the complete set of options and select the targeted permissions.

Once happy with your selection, click "Create". You can modify your Permissions further on the presented page.

Role Assignment

Roles, by themselves, provide no access. You must first assign them to an Access Group and a set of Objects before their permissions are applied to an Account. Let's do the first part now. Navigate back to the **Admin** > **Access** Groups tab and "View" your previously created Access Group. Select the "Roles" subtab and then "Edit" within the Roles widget.
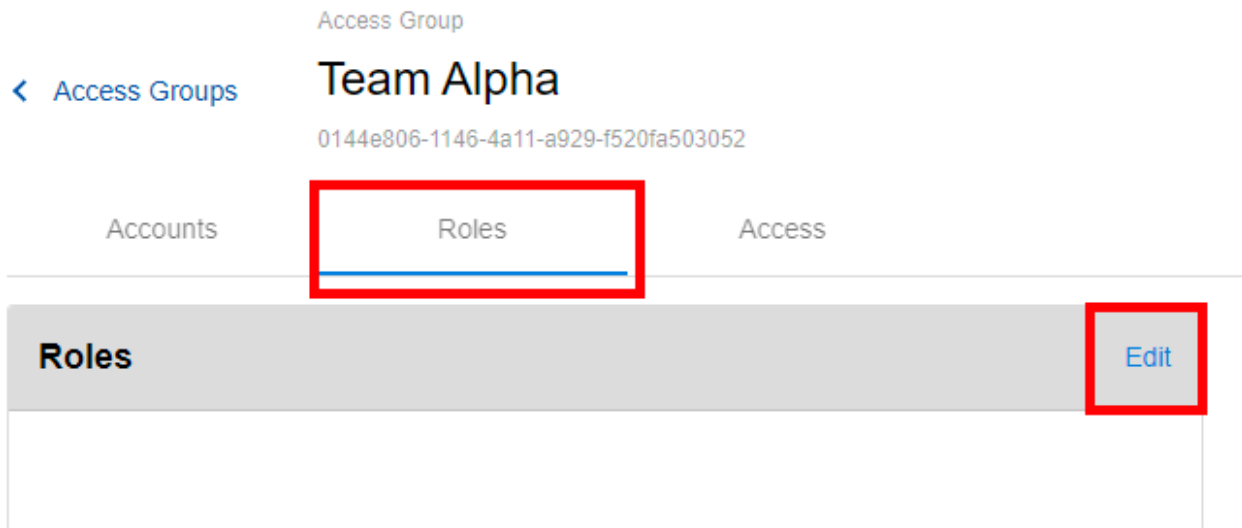
Now you can assign default Roles, such as "devops", and your newly created Role. You might recall that Role assignment was also possible during Access Group creation. On Save, your Access Group might look like the following.
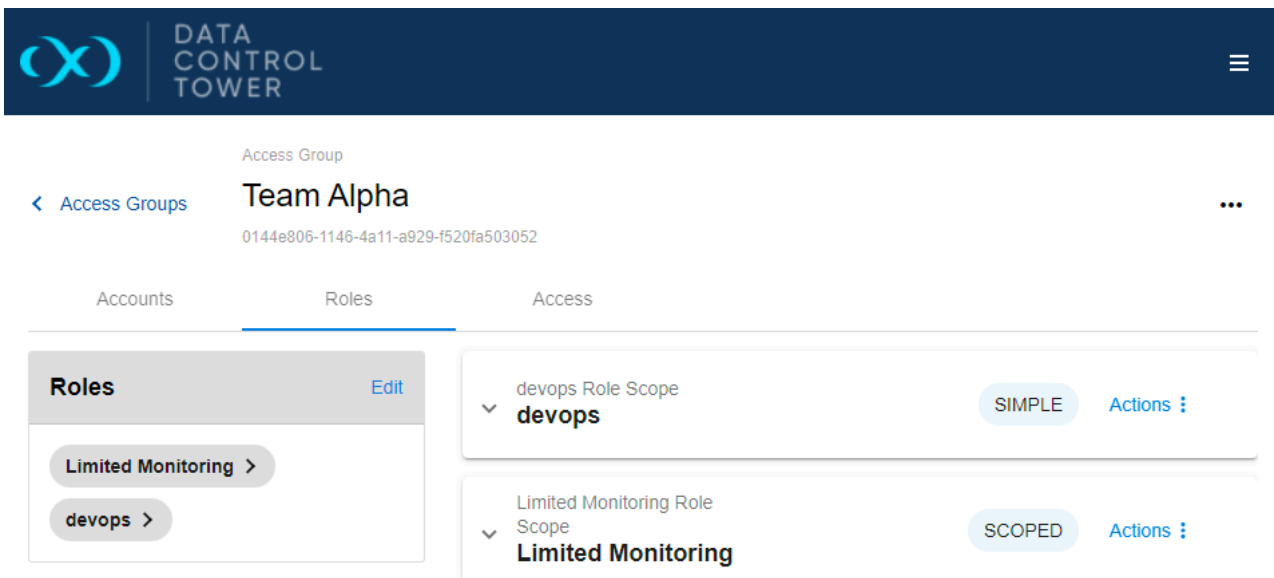
Immediately on assignment, all users within the Access Group will now have the permissions assigned to them through these roles. (Since you are currently an Admin user, you must log in as your test Account user.) However, you might notice that this user has full access to every object on DCT. The following section will define the Role scoping modes and refine the Account Object access.

7.1.3.5 Objects: Refine permission to targeted objects
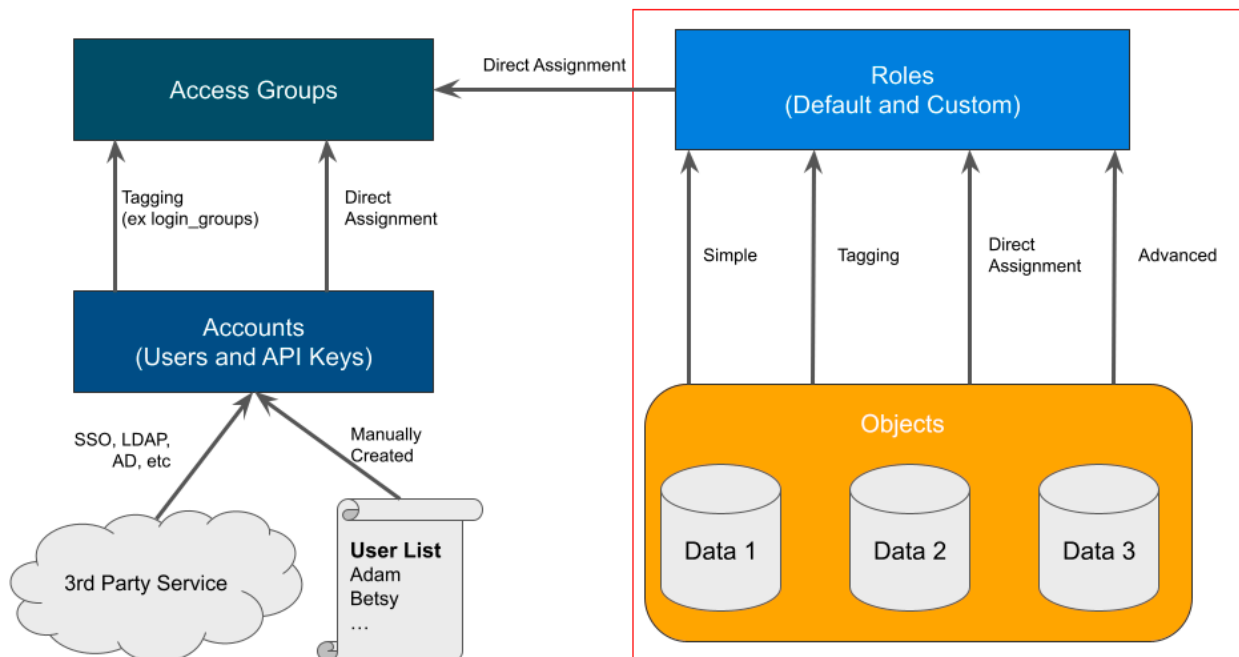
Every Access Group's Role has its own set of Objects to which the permissions are applied. In the previous section, we defined the permissions, and now we select the Objects. Objects are assigned in three different modes. They are listed below with their method of application:

1. Simple - All Objects within DCT.

2. Scoped
    a. Tags - Objects with matching Tags.
    b. Direct - Objects manually assigned.

3. Advanced Scoped - Objects are assigned directly on the permission action (such as Read Bookmark, Edit Bookmark, or Delete Bookmark) using Tags or Direct Assignment.

We will work through the first two in this post, Simple and Scoped. Advanced is easier to comprehend afterward and a solid self-lead challenge. Before diving into this section, we recommend that your DCT server has a handful of objects, such as Bookmarks or VDBs.

Simple

If you have been following the post steadily, you should have two Roles assigned to your Access Group. In my example I have "devops" and "Limited Monitoring". Both are given the "Simple" mode by default. We can see the breadth at which this Role governs by selecting anywhere on its row and then the "Preview" button on the right-hand side.

In this example, we select the "Preview" list for Bookmarks. It displays every Bookmark this role has access to and the method to which they are applied. If we wanted to validate, we could log in as a user on this Access Group and verify the permissions are applied. However, this is an easier way for Administrators to confirm without switching logins. Because this is a "Simple" scope, every object is available, so this view is not particularly intriguing. In the next part, we'll refine our Role.

> ℹ️ If you do not see any objects listed in the "Preview" widget, that object might not be available to DCT. This could be because (a) engines are not connected, (b) the DCT-only object (such as Bookmarks or VDB Groups) is not created, or (c) permissions are being enforced correctly.

Scoped - Direct

Let's change the mode to "Scoped" and target a subset of VDBs. On the Access Group > Roles tab, select the **Action** > **Edit** button of your chosen Role.
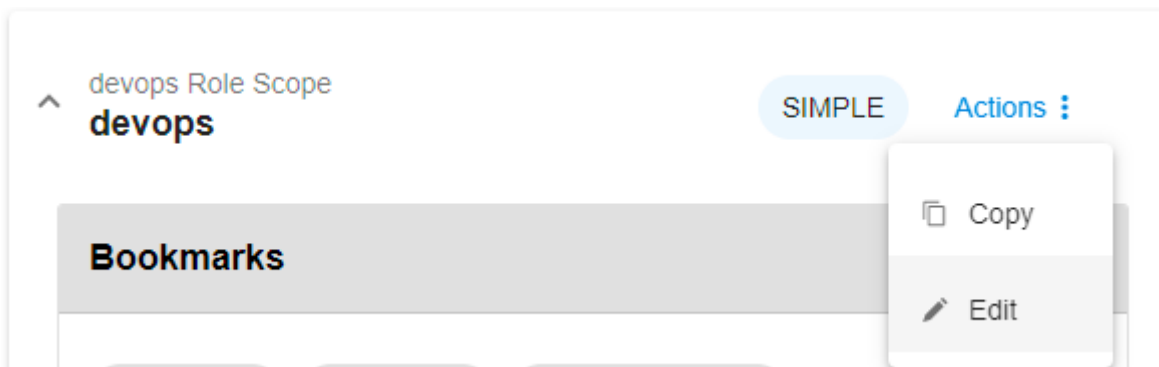
A new wizard will appear with the Simple, Scoped, and Advanced Scoped options. Change the Role's mode from "Simple" to "Scoped". Skip the "Add Tag Mappings" for now and select "Next" to move to "Add Objects". You will be presented with a long list of the objects available to DCT. This is where you can manually assign specific DCT objects.

Edit devops Role Scope   ✕

○ Scope Mode

○ Add Tag Mappings

● Add Objects

Add Objects

Manually select objects to include in addition to the objects being added dynamically from tag mappings.

Bookmarks	○ No Manually Added VDBs
CDBs	◉ Manually add VDBs
Database Templates	▬ **Name**
	☑ Mysql_MASK
Dataset Groups	☑ MySQL_DEV
dSources	☑ Mysql_QA
Engines	☐ Postgres_MASK
Environments	☐ Postgres_DEV
	☐ Postgres_QA
Sources	☐ AppFS_MASK
VCDBs	☐ AppFS_DEV
VDBs	☐ AppFS_QA

Cancel   Back   Next   **Submit**

Scroll down the Object type list and select VDBs. Next, choose the "Manually add VDBs" radio button and, on the right-hand list, select a couple of VDBs. Feel free to add other available Objects too. When happy with your selection, press the "Submit" button. This set of actions should change the Role's "Simple" mode to "Scoped" mode. Let's verify by, again, opening the Role's row, scrolling to your chosen Object Type, and selecting the VDBs' "Preview" button.

In my example, the same three VDBs I selected during permission configuration are shown here. If you want to verify manually, log in as your other test user and confirm.

ℹ️ Any other Roles or Access Groups assigned to this user might affect its visibility. So if you do this test, ensure it's not accidentally pulling in another permission set.

Scoped - Tags

Direct Assignment is a solid strategy for early onboarding and one-off requirements. However, as we expand our consumption of Delphix, I suggest leveraging the Tagging mechanism to assign permissions quickly. Similar to the Account & Access Group's "login_groups" tag, we can assign tags to Objects and Roles to immediately grant or restrict access. This is the recommended approach for a robust production implementation.

Before jumping back into a Role, navigate to the top-level Data >VDBs tab. (If you don't have any VDBs, select another tab with available objects.) Here identify a test object and select the "Add Tags" button.

In this form, we assign a simple Key-Value pair. This pair helps govern access and maintain the organization of the Delphix Platform. I've selected the "Team: Alpha" and "Environment-Dev" pairs in my example. Repeat the process for a couple of other objects using similar or different Key-Value pairs. As I explained earlier in this post, we can define and create an organizational structure in many ways. If you prefer other pairings, please experiment, such as with Geography, Age, DB Type, or Importance.

Next, let's take advantage of the created tags in the Access Model. Navigate back to your test Access Group, select the Roles tab, and edit the Role we modified previously. Because "Scoped" is already chosen, press the "Next" button, but this time stop on the "Add Tag Mappings" view. Similar to your Object's tag assignment, specify one or two of the same Key-Value pairs here.

Edit devops Role Scope   ✕

○ Scope Mode

● **Add Tag Mappings**   **Add Tag Mappings**

○ Add Objects   Select tags that will dynamically add objects that are assigned the mapped tag. In Advanced
                                       Mode, you can select tags for each permission included in the role.

                                       ┌──┬─────────────────┐
                                       │ ⬤ Team: Alpha ⊗   │ 🗑 Remove   │
                                       └──┴─────────────────┘

                                       ┌────────────────────────┬────────────────────────┬──────────┐
                                       │ Enter Tag Key   │ Enter Tag Value   │ + Tag   │
                                       └────────────────────────┴────────────────────────┴──────────┘

                                                        Cancel   Back   **Next**   Submit

In my example, this process will assign the Objects with the chosen "Team: Alpha" tag to this "devops" Role.
Thus, granting the set of permissions defined by "devops". Finally, we can verify again by completing the
form and previewing the Role.

| Compliance | Insights | **Admin** |

View Select VDBs

Search

Name ↑	Addition Method	
AppFS_cust_DEV	Team: Alpha	View >
AppFS_cust_MASK	Team: Alpha	View >
AppFS_cust_QA	Team: Alpha	View >
MySQL_DEV	Manual	View >
Mysql_MASK	Manual	View >
Mysql_QA	Manual	View >
OracleDEV_CHT	Team: Alpha	View >

Items per page: 50 ▼   1 – 9 of 9   ‹ ›

Close

In my example, we can see a mix of objects assigned to this role through Tags and Manual (direct) assignments.

At this point, challenge yourself by adding and removing tags to different Roles and Objects to understand the flexibility of the ABAC model. If you need a deeper dive into Tags, read our documentation here. (see page 155)

7.1.4 VDB templates

> For additional detail on VDB templates, visit the "Configuration Settings for Oracle VDBs" article in the Continuous Data Engine documentation.

DCT has implemented a global VDB template system to centrally manage and apply VDB templates for any and all VDB provisioning workloads. This feature works as an extension of the local VDB template system on Continuous Data Engines as a means of enforcing VDB configuration standards and policies uniformly.

DCT Admins have the choice of either importing pre-existing VDB templates from a local engine or creating net-new templates from within DCT.

7.1.4.1 Creating templates

Users can create Database Templates directly via DCT, which can then be used on VDBs across their engines. The DCT API interface for creating templates is equivalent to that of on-engines, requiring a name and sourceType, and optionally taking in a description and the list of config parameters. Here's a sample CURL command:

```
curl -X 'POST' \
 '<https://<APPLIANCE_ADDRESS>/v2/database-templates'> \
 -H 'accept: application/json' \
 -H 'Authorization: <API_KEY>' \
 -H 'Content-Type: application/json' \
 -d '{
 "name": "vdb-config-template-1",
 "source_type": "OracleVirtualSource"
 "parameters": {"config1": "value1", "config2": "value2"}'}
```

This will result in a new DCT **DatabaseTemplate** object, which can then be viewed using the **List/Get/Search** APIs.

7.1.4.2 Importing templates

Unlike many other Delphix objects, DCT is not automatically pulling in all the Database Templates from registered engines and creating DCT objects out of them. It is often the case that users have already made arrangements and have copies of their templates across their engines. DCT does not blindly import the templates to avoid generating duplicates, leading users to consolidating and clean up. Instead, DCT provides an import API that can be used to selectively choose which engines they wish to import their templates from, along with an API to undo imports. The import workflow has a couple of things to be aware of:

- The user cannot be selective of which individual templates to import from an engine. The import API will pull ALL templates from that engine.
- Import is allowed only one time per Engine. After an initial import, subsequent imports will be blocked, and it is assumed that a user will use the DCT APIs to create more templates.
- In the event that an import was done on accident or no longer desired, the undo import API can be called to delete all the imported templates from the selected engine. This will result in the removal of all DCT Database Templates that were created as a result of the import.
- If an imported template is later used on a VDB running on a different engine than where it was originally imported from, then the undo import flow is also prohibited, as DCT can no longer safely delete a template that is in use elsewhere.

Import templates from the engine:

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v2/database-templates/import' \
  -H 'accept: */*' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "engine_id": "3"
}'
```

Undo the imported templates from engine:

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v2/database-templates/undo-import' \
  -H 'accept: */*' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "engine_id": "3"
}'
```

7.1.4.3 Using templates

DCT Database Templates can be used by specifying the template_id property at VDB provisioning time, or by updating the template_id on an existing VDB. In either case, DCT will deploy the template to the respective engine and bind the template with the VDB. When a DCT Database Template currently in use is updated or deleted, those changes are propagated to the respective VDBs and engines.

> If a VDB has the same parameter called out in both VDB template and individual setting, the value specified in the template will take precedence. The individual parameter value will only be used if the VDB template is removed.

Updating a VDB to use template_id:

```
curl -X 'PATCH' \
  'https://<APPLIANCE_ADDRESS>/v2/vdbs/1-ORACLE_DB_CONTAINER-1' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "template_id": "319db966-961c-4977-a444-14d337aa3276"
}'
```

> Updates to a VDB template will propagate to all associated VDBs.

7.1.5 API metering

7.1.5.1 API metering instructions

DCT employs a per API consumption model, which requires API metering and periodic reporting to Delphix Customer Success. To support reporting of API consumption, DCT offers an API consumption reporting endpoint called, "api-usage-report". This report will provide a list of all unique API endpoints and how often they were used over the specified time period sorted by API and method.

Required inputs

- File type: CSV or JSON (CSV file types are compatible with most spreadsheet-style software like Excel or Google Sheets)
- Start/end date: The default start date is "when DCT was installed" and the default end date is the "time when the report was generated".

7.1.5.1.1 Example cURL call

```
curl --location --request GET 'https://[Inser_DCT_Server]/v2/reporting/api-usage-
report/?end_date=2022-06-14T09:00-04:00&start_date=2022-06-01T00:00Z' \
--header 'Content-Type: application/json' \
--header 'Accept: text/csv' \
--header 'Authorization: apk  1.xxxxxxxx'
```

Example output

```
api_endpoint,api_method,api_count
"/v2/management/api-clients",GET,2
/v2/management/engines,GET,1
"/v2/management/engines/search",POST,1
"/v2/reporting/api-usage-report",GET,2
```

7.1.6 Client telemetry

DCT provides complete flexibility to clients on how to attribute their API calls. DCT captures the value provided in an optional HTTP header (`X-Dct-Client-Name`) and standard, mandatory HTTP header (`User-Agent`) for the purpose of attributing an API call. These values are stored as `client_name` and

`user_agent` in the backend, and can be queried in the report. Below are some examples of how this can be used.

Example one

Clients can view the report grouped on the basis of client name and API classification by providing a `group_by` query parameter.

```
curl --location 'https://[Inser_DCT_Server]/v3/reporting/api-usage-report?
group_by=client_name%2Ckind' \
--header 'Authorization: <api_key>' \
--header 'Accept: text/csv'

api_endpoint,api_method,api_count,kind,client_name,user_agent,dct_version
,,2,management,client-1,,
,,5,management,client-2,,
,,20,management,client-3,,
```

The `group_by` parameter supports any combination of properties from `api_endpoint`, `api_method`, `kind`, `client_name`, `user_agent`, and `dct_version`. All properties function as their name describes, where `kind` corresponds to API classification.

Example two

Clients filter the records for a list of particular client names, DCT versions, user agents, or classifications, by providing corresponding query parameters.

1. Filter the API calls by client names.

```
curl --location 'https://[Inser_DCT_Server]/v3/reporting/api-usage-report?
group_by=client_name%2Ckind&client_name=client1%2Cclient2' \
--header 'Authorization: <api_key>' \
--header 'Authorization: apk <api key>
```

2. Filter the API calls by API classification.

```
curl --location 'https://[Inser_DCT_Server]/v3/reporting/api-usage-report?
group_by=client_name%2Ckind&api_metric_kind=automation' \
--header 'Authorization: <api_key>' \
--header 'Authorization: apk <api key>'
```

> The API query parameters for this report are dynamic; depending on the number of records in the backend and the granularity of the response requested, API response can be too large to be handled by DCT. DCT can run out of memory and eventually crash if that is the case. To prevent this from happening, this report has the maximum limit set to 10,000 records in the API

response. Thus, it is strongly recommended to always 'limit' the scope of the usage response by filtering records on the basis of `start_date` and `end_date` query parameters, or on the basis of client names and/or user agents.

7.2 Central governance workflows

Data Control Tower delivers the management layer for all connected Delphix engines by virtue of its converged architecture. As such, DCT has the ability to simplify everyday administration of common engine admin tasks. This section will go over various how DCT exposes object relationships and reports on meaningful use patterns under insights.

7.2.1 Governing engines (Continuous Data)

7.2.1.1 Infrastructure

DCT provides a near real-time list of all connected Continuous Data engines and lists them in an aggregate view. From the below screen, Delphix administrators can easily view and manage their engine connections.

Administrators can manage engine connects via the "Connect Engine" button on the top right corner. By clicking this button, the below dialogue will appear, asking for connection details.

> DCT will access the engine as a registered user and, as detailed in the **Deployment** section, requires both a username and password as well as admin-level access to the engine.

An option is available to configure DCT so that it automatically creates predefined tags on discovered objects ingested from the engine. This can be done in the **Tags** step of the **Connect Engine** wizard. Checking the box will make VDBs, dSources, and vCDBs automatically get tagged with the name of their Dataset Group on the engine.

Checking the second box will include the engine name as a tag on ingested objects. Finally, checking the last box allows users to set custom tags on ingested objects. Tags will be applied on VDBs, dSources, vCDBs, and Infrastructure Connections.

ⓘ   Auto-tagging can modify object access. Review permissions carefully after auto-tagging objects.

All existing and new objects from then on will automatically get the configured tags, which are no different from ordinary tags that can be added to objects. The auto-tagging configuration can later be updated in the Data Engine Details page by going to the **Action** menu and selecting **Auto-tagging Configuration**. This will open up a window with a checkbox to enable or disable auto-tagging.

Disabling auto-tagging will not remove any of the tags that were previously added. Instead, new ingested objects will not have tags automatically applied. In order to delete these tags, this must first be disabled and the tags will need to be deleted manually on all affected objects.

Since auto-tagging can have an impact on access control, non-admins will require special permissions to configure auto-tagging.

- To configure auto-tagging for pre-defined tags when registering an engine for the first time, the account must have a role with the **Configure pre-defined auto-tagging at registration** permission.
- To update the auto-tagging configuration for pre-defined tags on a pre-registered engine, the account must have the **Configure pre-defined auto-tagging** on the ENGINE object they wish to update.

Configuring custom tags also requires another level of permissions.

- To configure auto-tagging with custom tags when registering an engine for the first time, the account must have a role with the **Configure custom auto-tagging at registration** permission.
- To update the auto-tagging configuration's custom tags on a pre-registered engine, the account must have the **Configure custom auto-tagging** on the ENGINE object they wish to update.

7.2.1.2 Engine overview

Individual engine details can be seen and acted upon by clicking down on a particular engine detailed view. Once clicked, users will be sent to an "Overview" tab that provides relevant metadata related to the engine. From the Actions menu, you can also Deregister an engine.

> ℹ️ Deregistering or deleting an engine can also be done via the API with:
>
> ```
> curl --location --request DELETE '<https://<DCT Host Name>:<Port>/v3/
> management/engines/<ID or Name of the Engine to be deleted>' \
> --header 'Authorization: apk <api key for authentication>'
> ```
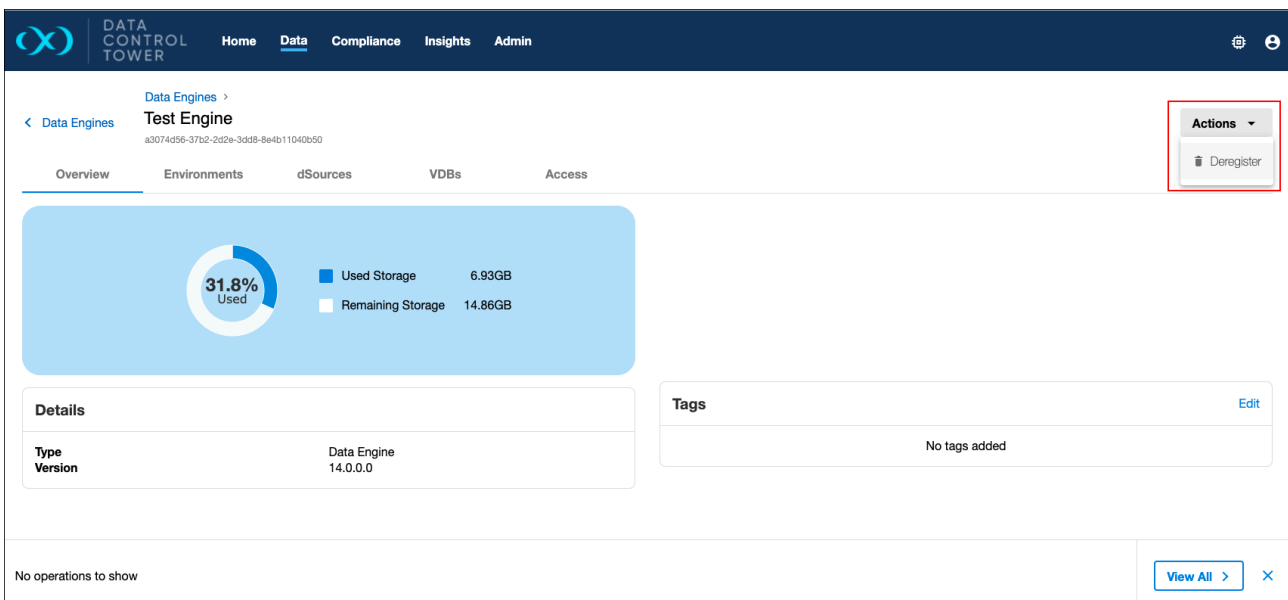

7.2.1.2.1 Engine-connected environments

The "Environments" tab presents all environment connections to that particular engine.

7.2.1.2.2   Local dSources

The "dSources" tab presents all dSources associated with the selected engine. Clicking the "View" button will link the user directly to the associated dSource page.

7.2.1.2.3   Local VDBs

The "VDBs" tab presents all VDBs associated with the selected engine. Clicking the "View" button will link the user directly to the associated VDB page to take action.
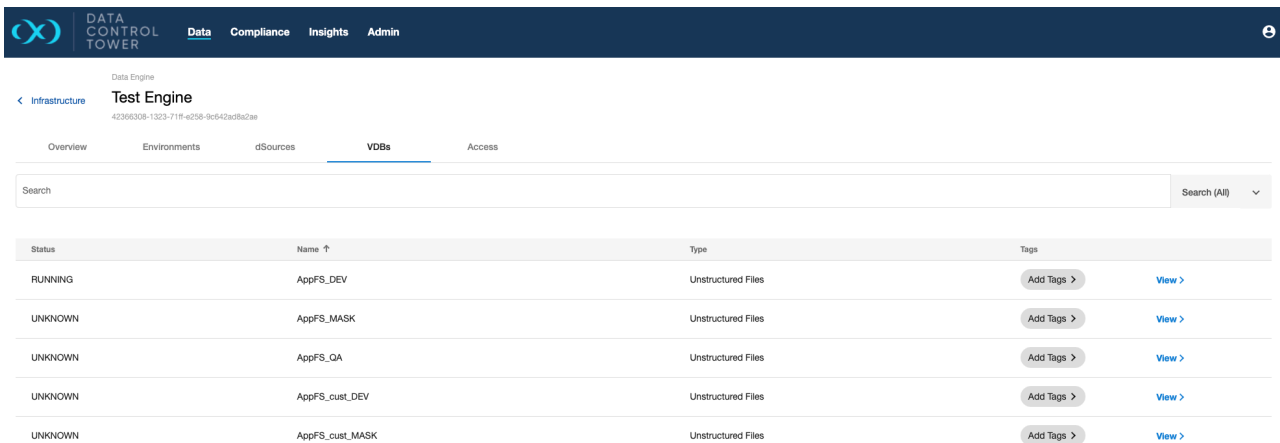
7.2.1.2.4 Engine-based Operations access

Users are able to audit which users have access to this particular engine, what access group they belong to, and the associated permissions that each user has on this engine. Admins are able to click on the "View" button to access further details under the access control screen related to that specific user.
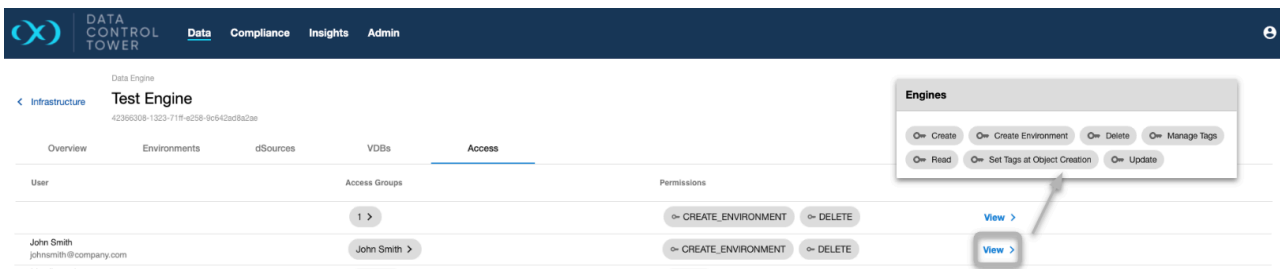
7.2.1.3 Infrastructure connection wizard

In the DCT interface, select the **Home** tab on the top navigation bar to see the Infrastructure Connections landing view, which hosts the list of connections. In DCT versions before 9.0.0, these connections were listed under the Environments section of the Data tab. Infrastructure Connections are the DCT equivalent of an environment. The **+ Infrastructure Connections** button will launch the wizard.
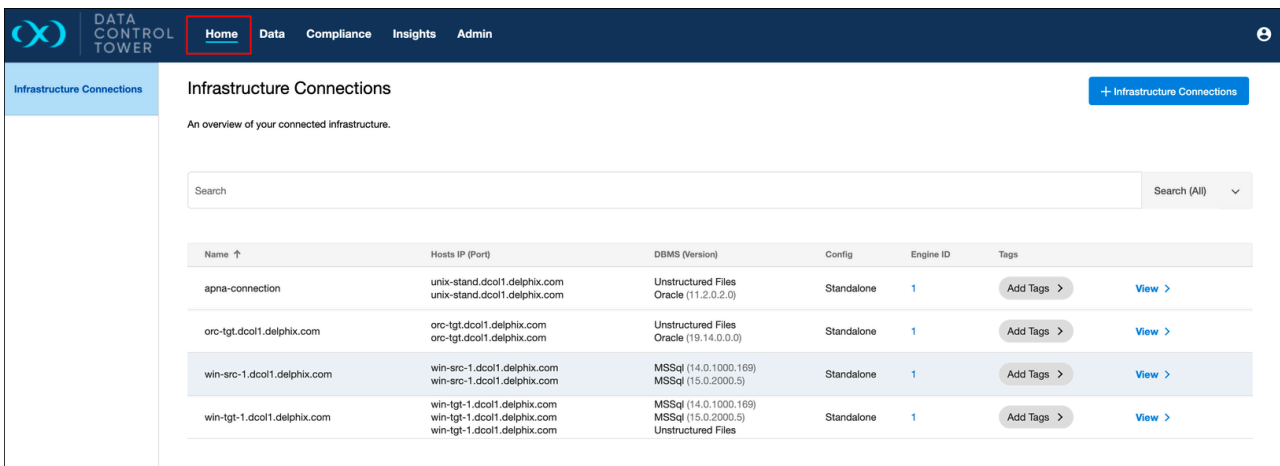
The following connections are available:

- UNIX environment
    - Standalone/Cluster
- Windows environment
    - Source/Target
    - Standalone/Cluster

7.2.1.3.1 Infrastructure connection wizard steps

1. **Connection Name**
   Enter the **Connection Name** in the corresponding field and select the **Associated Engine** from the dropdown.

2. **Host & Server**
   Select the **Host OS**, **Server Type**, and **Host Type** (applicable to Windows).

2 Unix

3 Windows

For Windows/Target/Standalone settings, a Delphix Connector download link has been added. Unlike the engine, this link makes an API call to authenticate and download the connector exe file.

3. **<u>Settings</u>**

   This step includes various connection setting options from basic items like the **Host/IP Address** and **SSH Port** to advanced items like **Discover SAP ASE**, **Provide my own JDK**, and **Set NFS**. It includes a **Validate** button to help confirm the environment user and prevents access to the next step if the credentials are not valid.

Create Infrastructure Connection   ✕

Host & Server

A Connection is a host or cluster with which Delphix will communicate.

Toolkit Path

Settings

Java Development Kit

Summary

☐ Provide my own JDK

Delphix will automatically provide a default JDK unless you elect to provide your own by checking 'Provide my own JDK'

Advanced Options

☑ Discover SAP ASE

Login Settings

◉ Username and Password

○ Password Vault

○ Kerberos

ASE DB Username

ASE DB Password

☐ Set Network File System Addresses(NFS)

Setting NFS addresses is not normally needed. If the host has multiple IP addresses, these can be added for NFS traffic by selecting this option to add additional host addresses.

☐ Set Delphix Session Protocol Options(DSP)

This option does not need to be selected unless you wish to configure Server/Client authentication for Delphix Session Protocol communication

Description

Cancel   Back   Next   Submit

A wide range of **Login Settings** are available for the OS user and to **Discover SAP ASE** (if applicable), like username/password, username/public key, PasswordVault, or Kerberos.

    a.   **HashiCorp**, **CyberArk**, and **Azure** PasswordVaults are supported.

    b.   The Kerberos login option only shows up when the user selects a Kerberos enabled engine as the target for environment creation.

Create Infrastructure Connection   ✕

Connection Name

Settings

Host & Server

A Connection is a host or cluster with which Delphix will communicate.

Host/IP Address

Settings

SSH Port

22

Summary

Login Settings

◉ Username and Password

○ Username and Public Key

○ Password Vault

○ Kerberos

OS Username

OS Password

Validate

Toolkit Path

Java Development Kit

Cancel   Back   Next   Submit

4. **Java Development Kit** (only applicable if selected in **Settings**)
   Set the custom JDK path in the corresponding text field.

Create Infrastructure Connection ✕

- Connection Name
- Host & Server
- Settings
- **Java Development Kit**
- NFS Addresses
- DSP Options
- Summary

Java Development Kit

The current Java Development Kit that is installed:
Default JDK

Java Development Kit (JDK) Path

path/to/jdk

Provide the full (absolute) path to the root of the JDK.

Cancel   Back   Next   Submit

5. **NFS** (only applicable if selected in **Settings**)
   Set NFS addresses in the corresponding text field (comma separated).

Create Infrastructure Connection ✕

- Connection Name
- Host & Server
- Settings
- Java Development Kit
- **NFS Addresses**
- DSP Options
- Summary

NFS Addresses

Set Network File System (NFS) Addresses

NFS Addresses

host1, host2.com, 1.12.21.32

Cancel   Back   Next   Submit

6. **DSP** (only applicable if selected in **Settings**)
   Set DSP configurations in this step.

Create Infrastructure Connection   ✕

DSP Options

These fields have no effect unless client and server authorization for remote connections is enabled. These can be enabled in the Network Security section of the associated engine's Setup application dashboard.

- Connection Name
- Host & Server
- Settings
- Java Development Kit
- NFS Addresses
- **DSP Options** ●
- Summary

DSP KeyStore Path
DSP config1

DSP KeyStore Password
•••••••••

DSP KeyStore Alias
DSP config2

DSP TrustStore Path
DSP config3

DSP TrustStore Password
•••••••••

Cancel   Back   **Next**   Submit

7. <u>**Summary**</u>

Shows a comprehensive summary of the selected options in the previous configuration steps. Shows the type of login being used for both the OS user and SAP ASE (if applicable).

Create Infrastructure Connection   ✕

Summary

Review the configuration for this Infrastructure Connection

- Connection Name
- Host & Server
- Settings
- Java Development Kit
- NFS Addresses
- DSP Options
- **Summary** ●

Host

Host OS
Unix/Linux

Server Type
Standalone

Connection Name
My IC

Associated Engine
sj-2903.dcol1

Host/IP Address
unix-stand.dcol1.delphix.com

NFS Addresses
host1, host2.com, 1.12.21.32

SSH Port
22

Toolkit Path
/work

Java Development Kit (JDK) Path
path/to/jdk

DSP KeyStore Path
DSP config1

DSP KeyStore Alias
DSP config2

DSP TrustStore Path
DSP config3

Discover SAP ASE

User

Login Type
Password authentication used

OS Username
sybase

Cancel   Back   Next   **Submit**

7.2.2 Governing dSources

7.2.2.1   Managing dSources

DCT provides the ability to view, search, sort, and filter all dSources within a connected Delphix ecosystem. This page can be found under the **Data** section and is used to find and act upon all dSources, if they have the appropriate access.

7.2.2.2   dSource overview

Individual dSource details can be viewed and acted upon by clicking down on a particular dSource's detailed view. Once clicked, users will be sent to an "overview" tab that provides relevant metadata related to the dSource.

7.2.2.3 Timeflow visibility

Users are able to view snapshot information by tabbing over to the "timeflow" section, which lists all available snapshots via a vertical timeline. Users are able to modify snapshot retention periods by clicking on the ellipsis located to the right of the relevant snapshot.
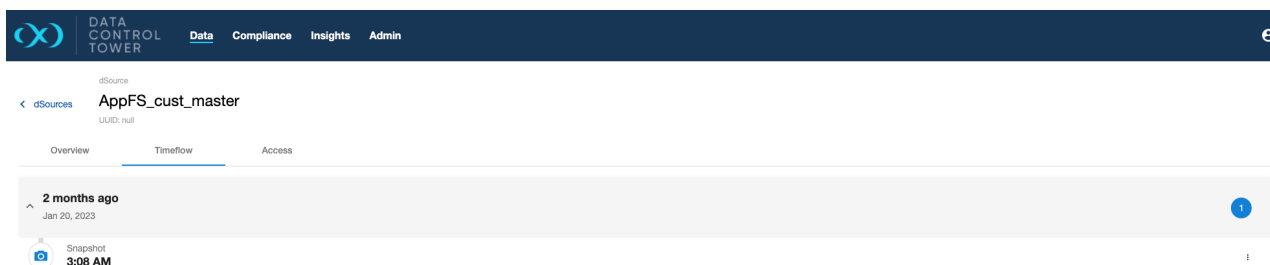
7.2.2.4 Access auditing

Users are able to audit what other users have access to a particular dSource, what access group they belong to, and the associated permissions that each user has on that particular dSource.
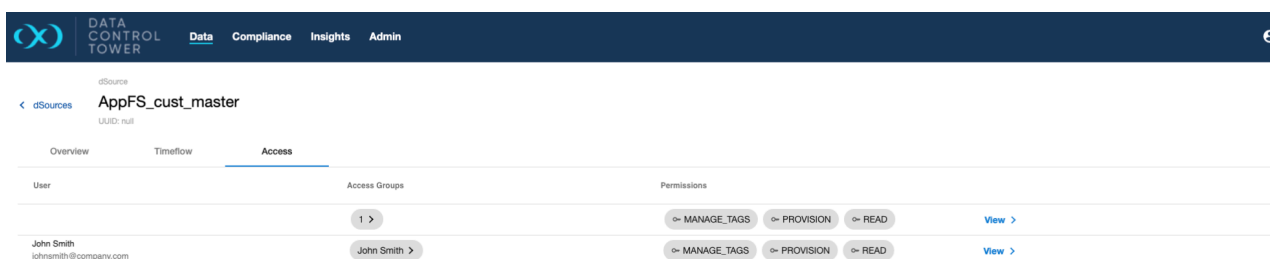
7.2.2.5 dSource linking

DCT 10.0.0 introduces the feature to link a dSource using DCT APIs. With the new API parameters exposed from DCT, you can get the dSource linking defaults and use them to connect the dSource. The API parameters are available in the API references[115] article under:

- **BaseDSourceLinkSourceParameters**
- **OracleDSourceLinkSourceParameters**
- **OracleStagingPushDSourceLinkSourceParameters**
- **AppDataDSourceLinkSourceParameters**
- **ASEDSourceLinkSourceParameters**
- **LinkDSourceResponse**
- **LinkDSourceDefaultRequest**
- **MSSQLDSourceLinkSourceParameters**
- **MSSQLDSourceStagingPushLinkSourceParameters**

---

[115] https://dct.delphix.com/docs/latest/api-references

> 📄 DCT supports APIs for linking all types of databases. DCT does not have a GUI flow for these APIs yet, which is why they are API only.

As a prerequisite, Continuous Data Engines should be registered in DCT and should have non-linked sources to be linked as a dSource.

7.2.3 Governing VDBs

DCT provides the ability to view, search, sort, and filter all VDBs within a connected Delphix ecosystem. This page can be found under the **Data** section and is used to find and act upon all VDB if they have the appropriate access.

7.2.3.1 VDB overview

Individual VDB details can be seen and acted upon by clicking down on a particular VDB detailed view. Once clicked, users will be sent to an "overview" tab that provides relevant metadata related to the VDB.

7.2.3.2 VDB active timeline

Actionable snapshots are listed on the Active Timeflow tab. From this page, users can refresh, enable, disable, start, stop, delete, and create bookmarks on the VDB.

> ℹ️ These actions are available for vCDBs as well. When performing disable or delete actions on a vCDB, it is important to stop all vPDBs first.
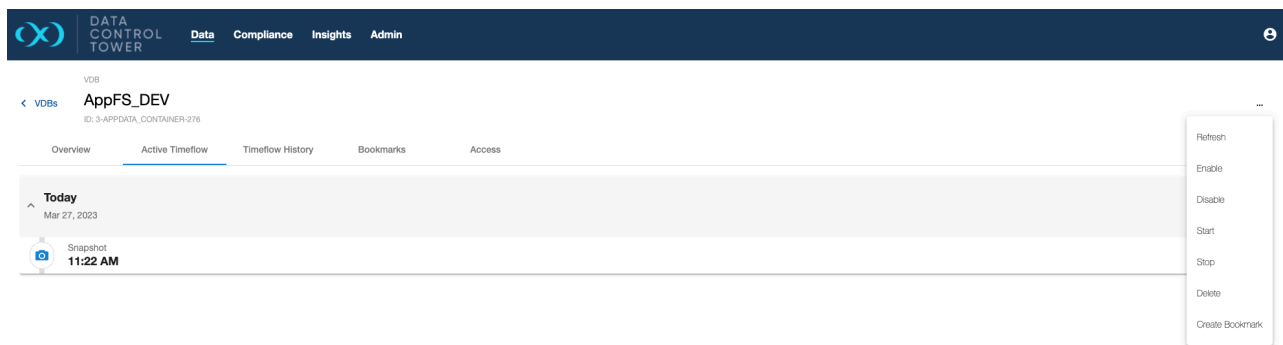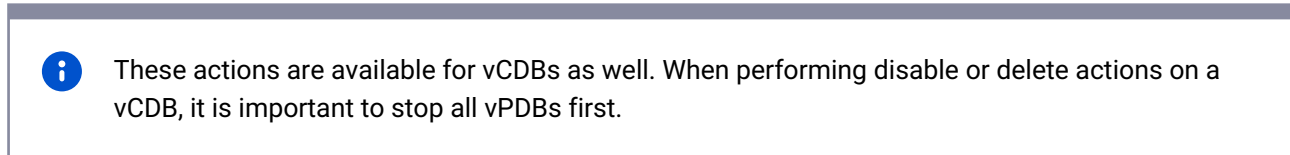
7.2.3.3 VDB timeline history

A chronological history of all non-active timelines (commonly referred to as **timeflows**) is shown under the Timeflow History tab. From this page, developers can curate their QA or Development work by renaming timeflows to match their testing history. Developers also have the ability to access old timeflow data by making a particular timeflow active. Navigate to the **Continuous Data workflows** section, then **VDB operations in the UI**. See Timeline history UI[116] for more details.

The VDB Details Timeline History tab now shows bookmarks along with the snapshots. The bookmark details will include **Name**, **Data Timestamp** or **Location**, **Log Delta**, and **Expiration**. An operation to delete the bookmark is available with the action button on the right of a bookmark line item.

---

116 https://portal.document360.io/docs/timeline-history

Data Timestamp is only shown when available for the bookmark. Data Location is only shown when the bookmark is created using a location.

Log Delta is the delta between a bookmark's data timestamp and a snapshot's timestamp when the bookmark's data timestamp is available. It can also be the delta between a bookmark's location and a snapshot's location when the bookmark is created using a location.

7.2.3.4 VDB bookmarks

A list of all bookmarks generated on the selected VDB can be found under the "Bookmarks" tab. This page provides a list of all bookmarks allowing for general organization and actions (developers can use bookmarks as a refresh or provision point from the API).

7.2.3.5 VDB access

Users are able to audit which users have access to this particular VDB, what access group they belong to, and the associated permissions that each user has on that VDB. Admins are able to click on the "View" button to access further details under the access control screen related to that specific user.
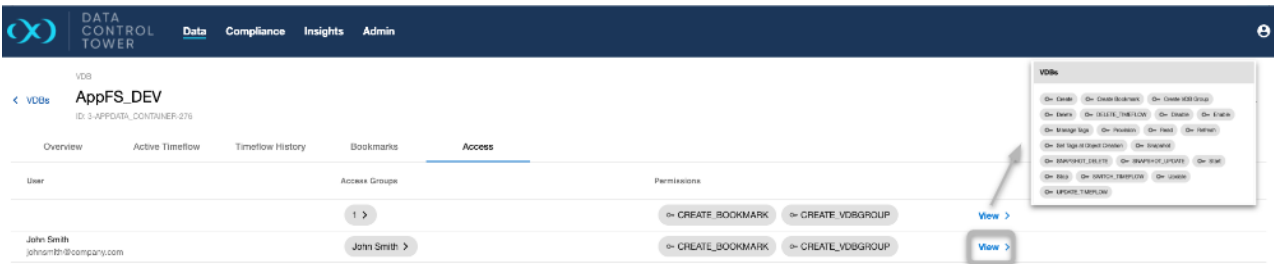
7.2.3.6 VDB templates

Importing and removing imported VDB templates from connected engines is an available action from the **VDB Config Templates** page.
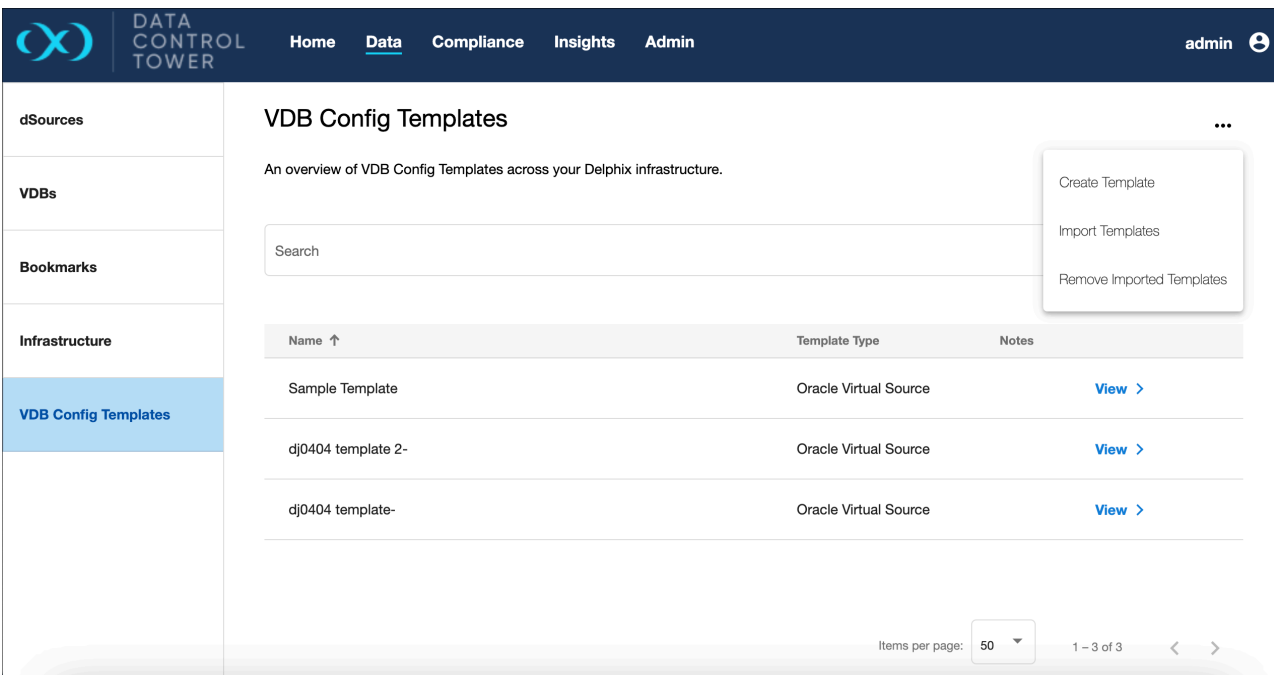
7.2.3.6.1 Import templates

To import a template, select the engines from the list in the import dialog.
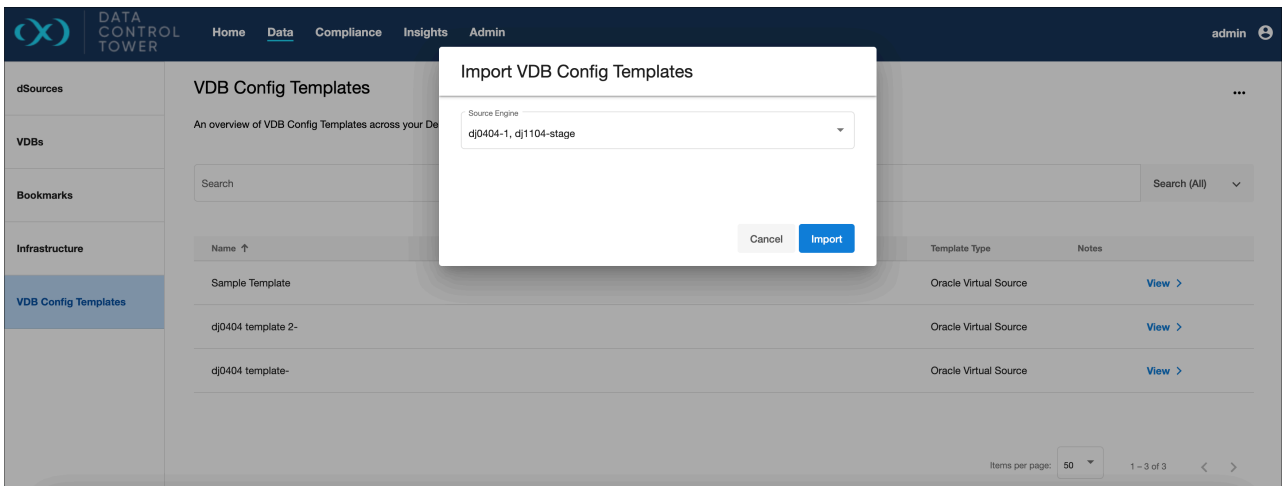
7.2.3.6.2   Remove imported templates

To remove a template, select the engines from the list in the remove dialog.
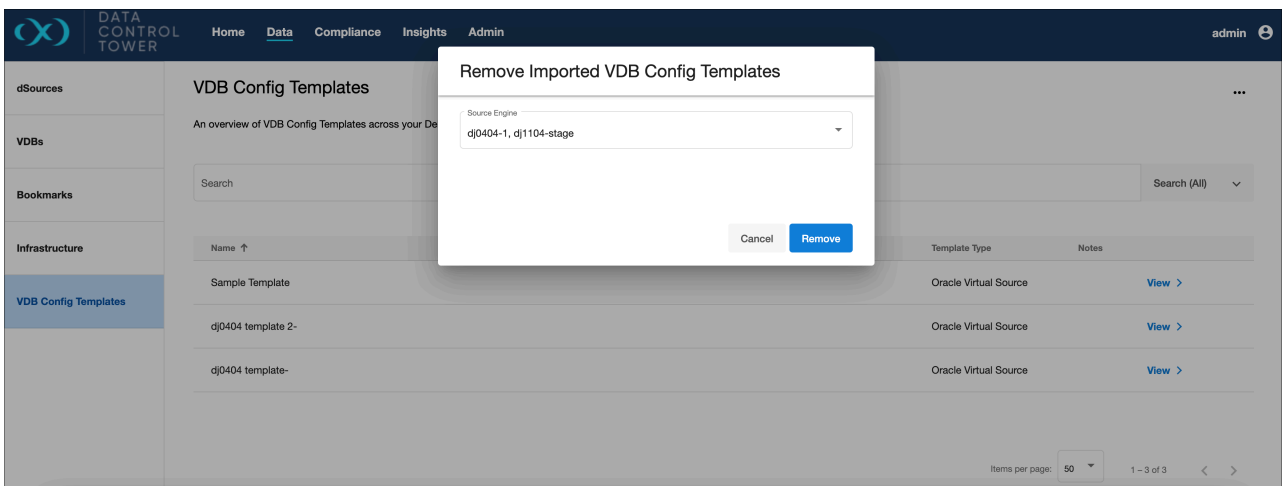
7.2.4   Governing environments (Continuous Data)

7.2.4.1   Global environments list

DCT provides the ability to view, search, sort, and filter all Continuous Data environments within a connected Delphix ecosystem. This page can be found under the **Data** section and is used to find and act upon all environment connections.
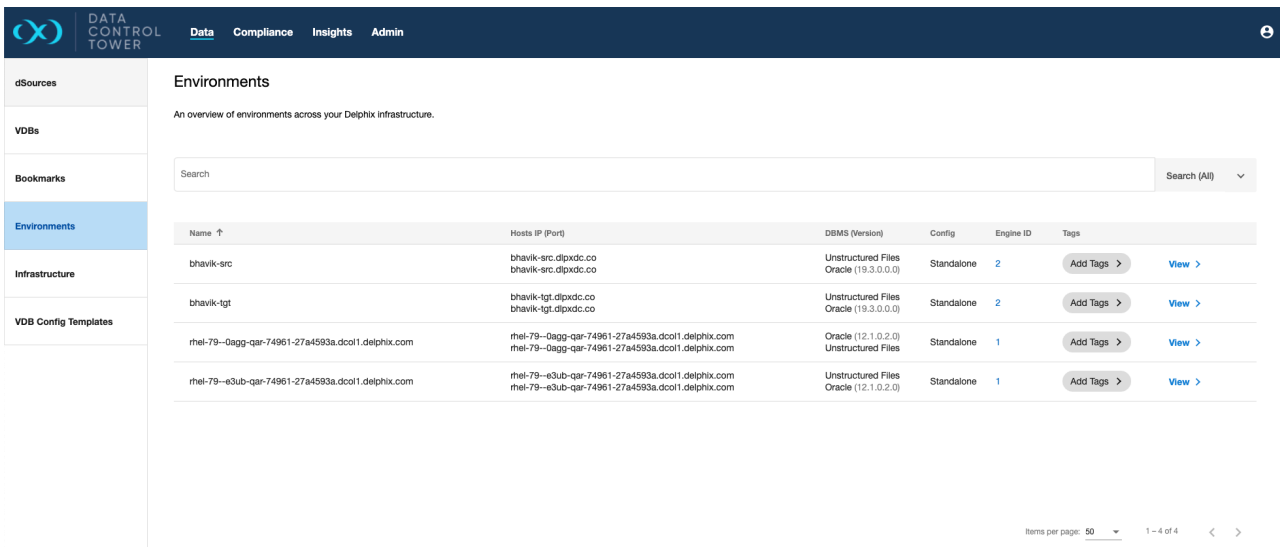
7.2.4.2 Manage environments

Selecting a standalone environment in the **Data** page shows an ellipsis in the top right corner. When the button is selected, the option to Enable/Disable, Refresh, or Delete the environment appears.
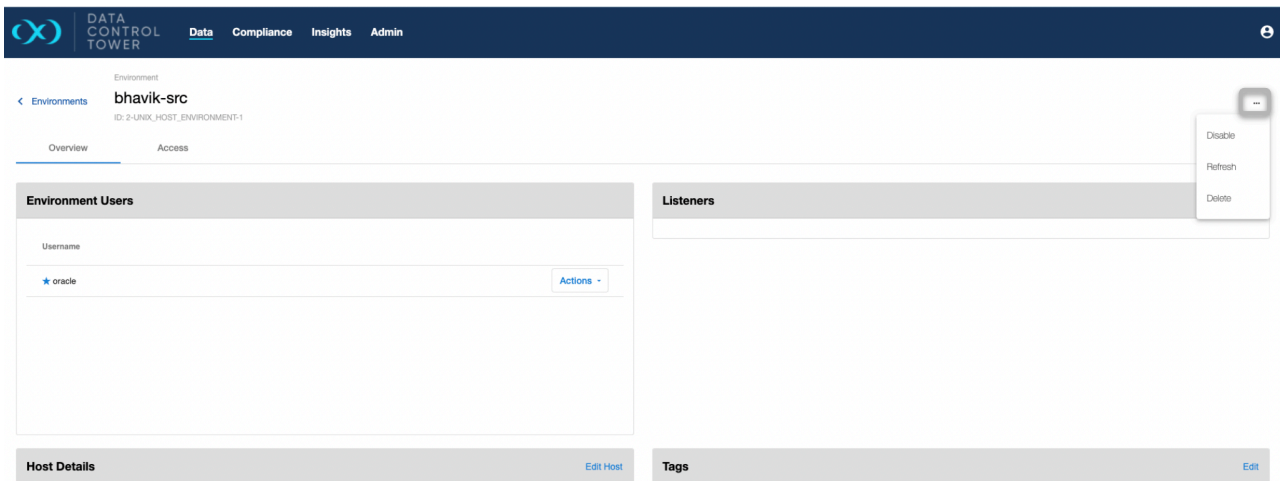
7.2.4.3 Edit host details

Selecting a standalone environment in the **Data** page shows an **Edit Host** option; it is not yet available for cluster environments. When the button is selected, the host details window will open, showing the input fields that can be edited. Select 'Save' to confirm the changes and close the window.
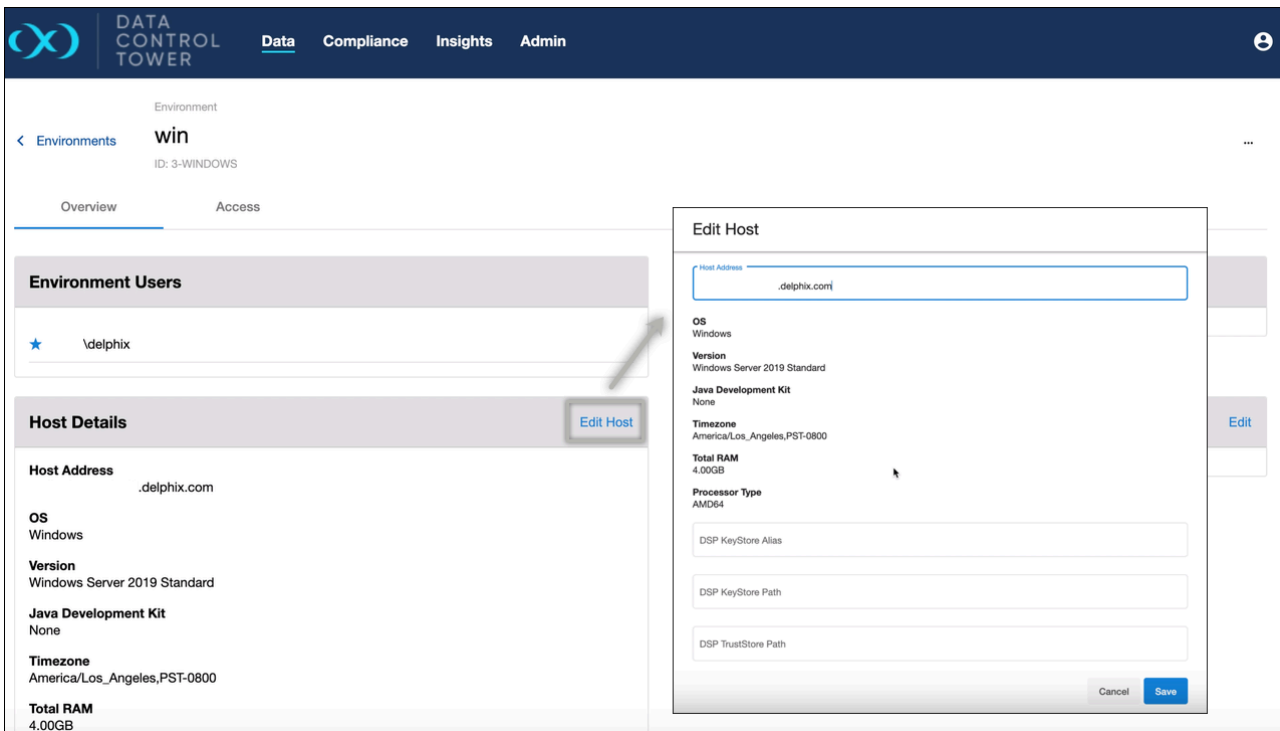
7.2.5 Governing bookmarks

7.2.5.1 Global Bookmarks List

DCT provides a near real-time list of all bookmarks across all VDBs and VDB-groups and their associated VDB(s). From the below screen, Delphix administrators can easily view and manage their bookmark estate.
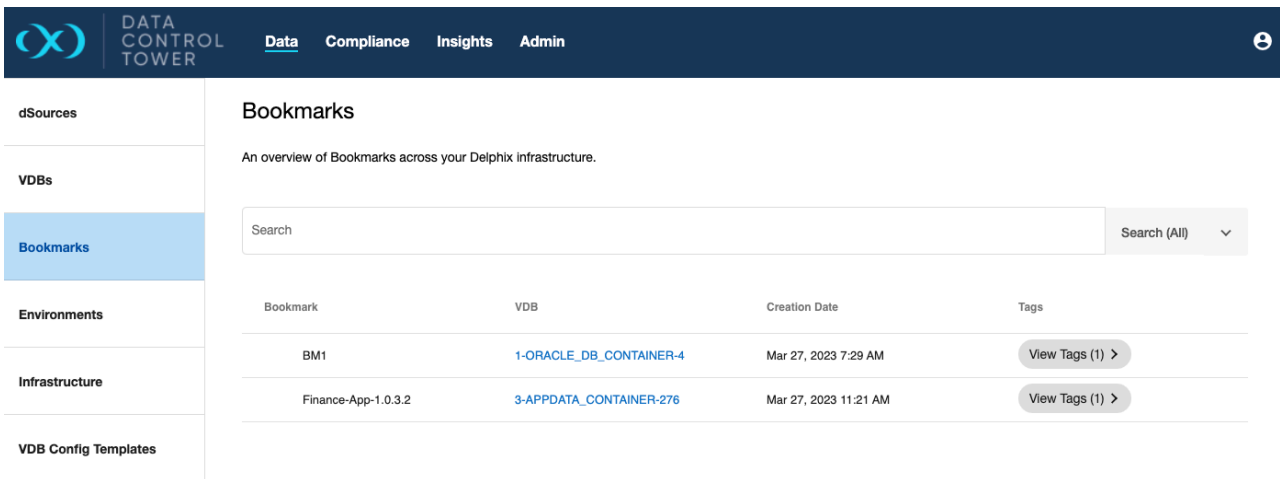
Support has been added for the creation of bookmarks for dSource snapshots, similar to VDB snapshots. These properties have been added:

- `inherit_parent_tags` is a new property added in the request payload. This indicates whether this bookmark should inherit tags from the parent dataset.

- `dsource_ids` is a new property in the create bookmark response. This indicates the list of dSource IDs associated with this bookmark.

The Bookmarks list now has additional search capabilities on the dSource Details page. The VDB Details page also has search functionality and a list of bookmarks compatible with said VDB. Columns show the Source and Expiration Date of a bookmark.

In DCT 18.0.0, bookmarks are no longer deleted by the system when all its datasets are removed from DCT. These bookmarks will only be removed when they expire or are manually deleted. DCT will automatically delete all bookmarks linked to a VDB when the VDB does not exist anymore, the same is applied to VDB groups.

7.2.6 Governing replications

7.2.6.1 Introduction

DCT 9.0.0 introduces the feature to differentiate between the replicated objects and original objects in case of a parent and replicated engines, both are registered with DCT.

7.2.6.2 Prerequisites

One parent and one replicated engine is required with few replicated dSources and VDBs.

7.2.6.3 User interface

- Convenient separation between the replicated objects(VDB/dSource/environments) and original objects.

- Users can filter the replicated object using the advanced filter.

- All the actions on the replicated objects are disabled.

- The status of the replicated objects are N/A.
- On the VDB provisioning wizard, replicated objects(VDB/dSources) are marked as replicated.

7.2.6.4 API

Expand the section below to show search APIs that are updated to return three additional fields: `is_replica`, `namespace_id`, and `namespace_name`. These fields are described below.

Search APIs

1. **Sources**
   GET: `/sources`
   GET: `/sources/{sourceId}`
   POST: `/sources/search`

2. **VDBs**
   GET: `/vdbs`
   GET: `/vdbs/{vdbId}`
   POST: `/vdbs/search`

3. **dSources**
   GET: `/dsources`
   GET: `/dsources/{dsourceId}`
   POST: `/dsources/search`

4. **CDBs**
   GET: `/cdbs`
   GET: `/cdbs/{cdbId}`
   POST: `/cdbs/search`

5. **VCDBs**

GET: `/vcdbs`

GET: `/vcdbs/{vcdbId}`

POST: `/vcdbs/search`

6. **Environments**

GET: `/environments`

GET: `/environments/{environmentId}`

POST: `/environments/search`

7. **Dataset-groups**

GET: `/groups`

GET: `/groups/{groupId}`

POST: `/groups/search`

8. **Timeflows**

GET: `/timeflows`

GET: `/timeflows/{timeflowId}`

POST: `/timeflows/search`

9. **Policies**

GET: `/virtualization-policies`

GET: `/virtualization-policies/{policyId}`

POST: `/virtualization-policies/search`

10. **Snapshots**

GET: `/snapshots`

GET: `/snapshots/{snapshotId}`

POST: `/snapshots/search`

1. `is_replica` : Boolean telling that this object id replicated or not.

2. `namespace_id` : This field will only come for replicated objects and is essentially the `namespace id` of the replicated object.

3. `namespace_name` : This field will only come for replicated objects and is essentially the `namespace name` of the replicated object.

7.2.6.5 Replication profile management

7.2.6.5.1 Overview

Data Control Tower (DCT) version 16.0.0 and above offers comprehensive CRUD (Create, Read, Update, Delete) functionalities for replication specifications directly within the Delphix Engines registered with DCT. This enhances the management and efficiency of replication processes.

7.2.6.5.2 Prerequisites

To leverage the new replication functionalities, you need:

- A running instance of DCT.
- At least one registered Delphix Engine.
- One or more dSources or VDBs on the engine.

7.2.6.5.3 Features

7.2.6.5.3.1 GUI access

- **Replication profiles Summary page**: View all known replication profiles, including details like name, source, target host, and mode.
- **Profile Creation Wizard**: Create new profiles using the guided wizard, initiated via the *+Replication Profile* button, available in the upper right corner of the page.

7.2.6.5.3.2 API support

- **Endpoints**: Create, update, delete, and execute replication profiles directly through API calls.
- **Replication Spec Parameters**: Greater visibility and control with the inclusion of previously absent parameters in the APIs.

7.2.6.5.3.3 Profile creation in SDD mode

- **Selective Data Distribution (SDD) mode (DCT 18.0.0 and up)**: Allows the creation of profiles where only masked VDBs can be selected.
- **VDB selection rules**:
    - Only one masked VDB per parent dSource.
    - No multiple sibling masked VDBs from the same unmasked dSource.
    - Parent dSource or VDB (and its lineage) not automatically included, except for some data for disk space optimization.
    - Environments containing database instances used in a replicated VDB are included.

- Only necessary database instances and repositories for replicated VDBs are included.
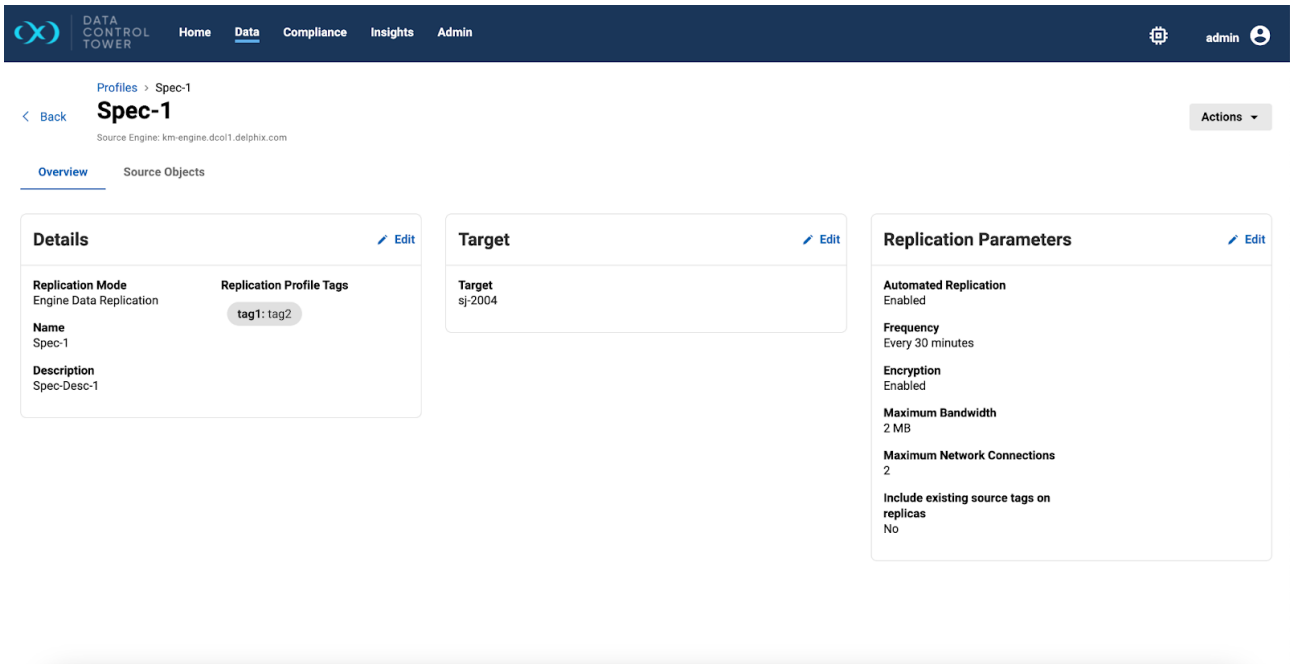
7.2.6.5.4 Getting Started

Access the Replication Profiles page by navigating to **Data** > **Replication** in the DCT GUI.
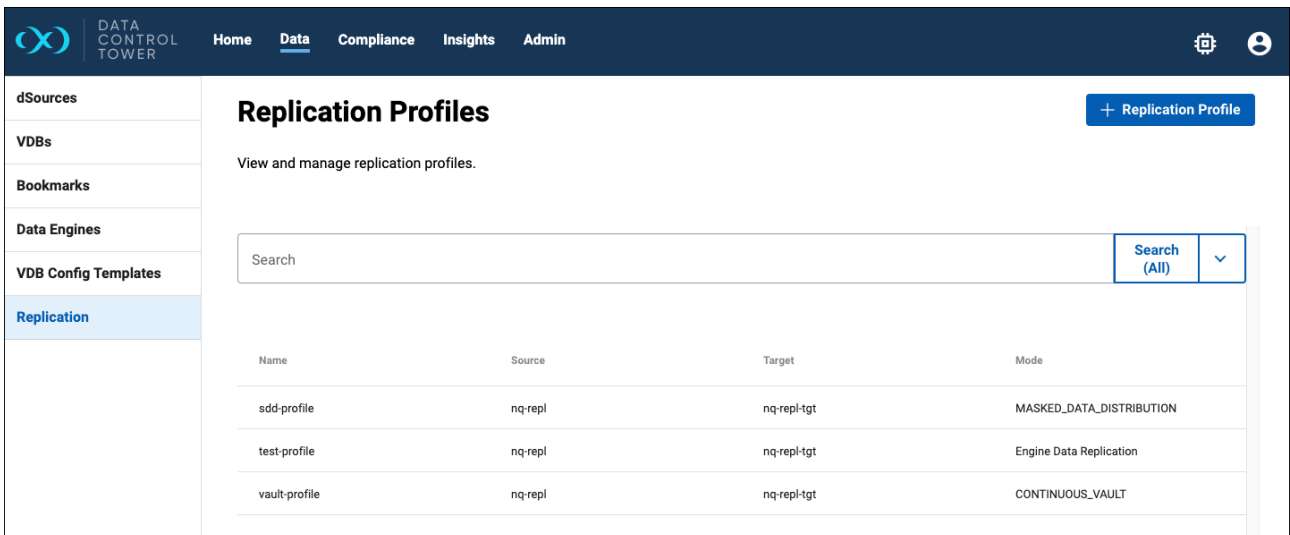
7.2.6.5.4.1 Creating and Setting Up Replication Profiles

Use the GUI wizard or API calls to set up new replication profiles by specifying parameters such as source, target, and replication mode.

Wizard sections

- **Mode**: Engine Data replication, Selective Data Distribution (SDD), and Continuous Vault are available options as of the latest version, DCT 20.0.0.

- **Details**: Includes Name, Description, and Tags (used in the DCT profile).

- **Target Engines**: Only target engines registered with DCT can be selected.

Create Replication Profile ✕

Target Engines

Select a target engine destination for replicas created by this repication profile.

	Status		Name/UUID	Type	Host	Tags
☐	⚡	Online	nq-repl ec2c9c0a-bf9d-a40	Virtualization 21.0.0.0	nq-repl.dlpxdc.co	Add Tags ›
☐	⚡	Online	nq-repl-tgt ec2df672-50e7-205	Virtualization 21.0.0.0	nq-repl-tgt.dlpxdc...	Add Tags ›

Search (All) ⌄

- **Replication Objects**: Chosen relative to the source engine selected from the dropdown menu.
  - **Display Options**: View only selected or unselected objects.
  - **Filtering**: Filter the replication objects tree nodes using the search input.

Create Replication Profile ✕

Replication Objects

Select the objects to replicate.

Source Engine: km-engine   Display Option: Show All   Search

- ☐ km-engine
  - ☐ 📁 Oracle RAC
    - ☐ 🔊 CDOMLOSRCA1D
    - ☐ 🗎 =(1+1)
    - ☐ 🗎 VDBO_7SY
    - ☐ 🗎 VDBO_J8G
    - ☐ 🔊 CDOMLOSRCA1DPDB1
    - ☐ 🔊 DBOMSR3A85E9
    - ☐ 🗎 VCCD_TVG
  - ☐ 📁 Testing DCT

Cancel   Back   Next   Save Profile

- **Parameters**: Includes automated replication and traffic options similar to the Delphix Engine.
  - **Tags**: Existing source object tags can be incorporated onto replicas.

Create Replication Profile   ✕

Mode

Details

Target Engines

Replication Objects

Parameters

Summary

Parameters

Configure replication traffic parameters and optionally specify tags to be added to the replicas created from this replication profile.

Automated Replication

☐ Enable Automated Replication

Every [1]   Time Period [Day(s) ▾]   at [12 ▲▼] : [00 ▲▼] [AM ▾]

[Show Cron String Format]

Traffic Options

☐ Enable Encryption

☐ Limit Bandwidth

Maximum Bandwidth (MB)
[0]

Maximum Network Connections
[1]

Replica Tags

Automatically apply tags to replicas created by this replication profile.

☐ Include existing source tags on replicas

[Cancel] [Back] [Next] [Save Profile]

- **Summary**: Review all configured details before saving the profile.

Create Replication Profile   ✕

Mode

Details

Target Engines

Replication Objects

Parameters

Summary

Summary

Review the configuration for this replication profile.

Details	
Replication Mode	**Replication Profile Tags**
Engine Data Replication	No tags added
Name	
None	
Description	
None	

Replication Parameters
Automated Replication
Disabled
Encryption
Disabled
Maximum Network Connections
1
Include existing source tags on replicas
No

Source Objects

⌄ 📁 Oracle RAC

      🔊 CDOMLOSRCA1D

      📄 =(1+1)

      📄 VDBO_7SY

[Cancel] [Back] [Next] [Save Profile]

7.2.6.5.5 Additional GUI Details

7.2.6.5.5.1 Replication profile List page

- **View Button**: Allows users to view and edit replication profile details directly from the list page. This is where existing replication profiles live.
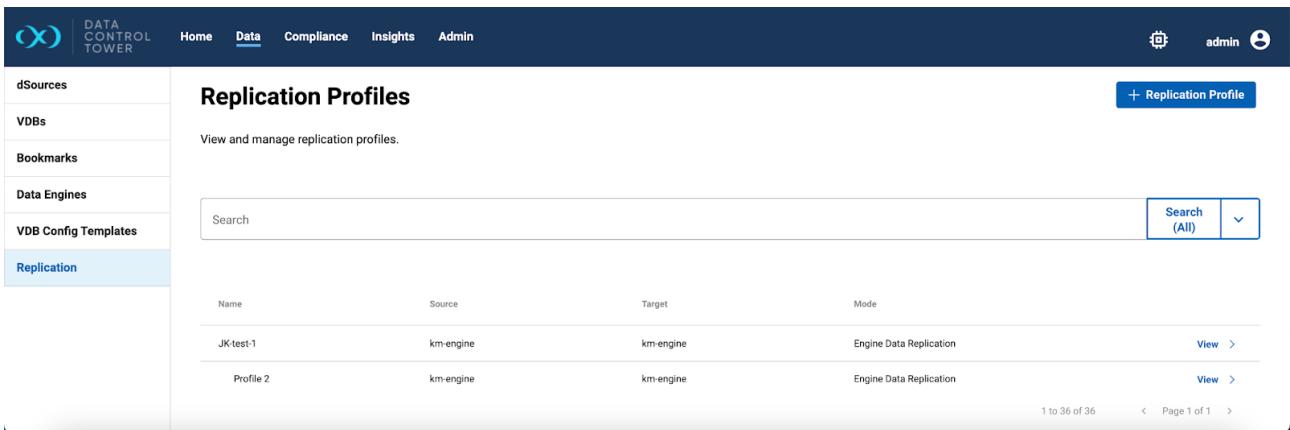
7.2.6.5.5.2   Replication profile Details page

- **Actions menu**: Lists actions that can be performed on the replication profile.
  - **Delete**: Deletes the replication profile, redirecting users back to the list page.
  - **Replicate Now**: Initiates the replication process immediately, displaying operations in the monitoring bar.
  - **Start Failover (DCT 19.0.0 and up)**: Initiate a failover directly from the UI via **Data** > **Replication** > **Replication Profile Details View** > **Actions** > **Start Failover**. This helps minimize downtime during failover events.
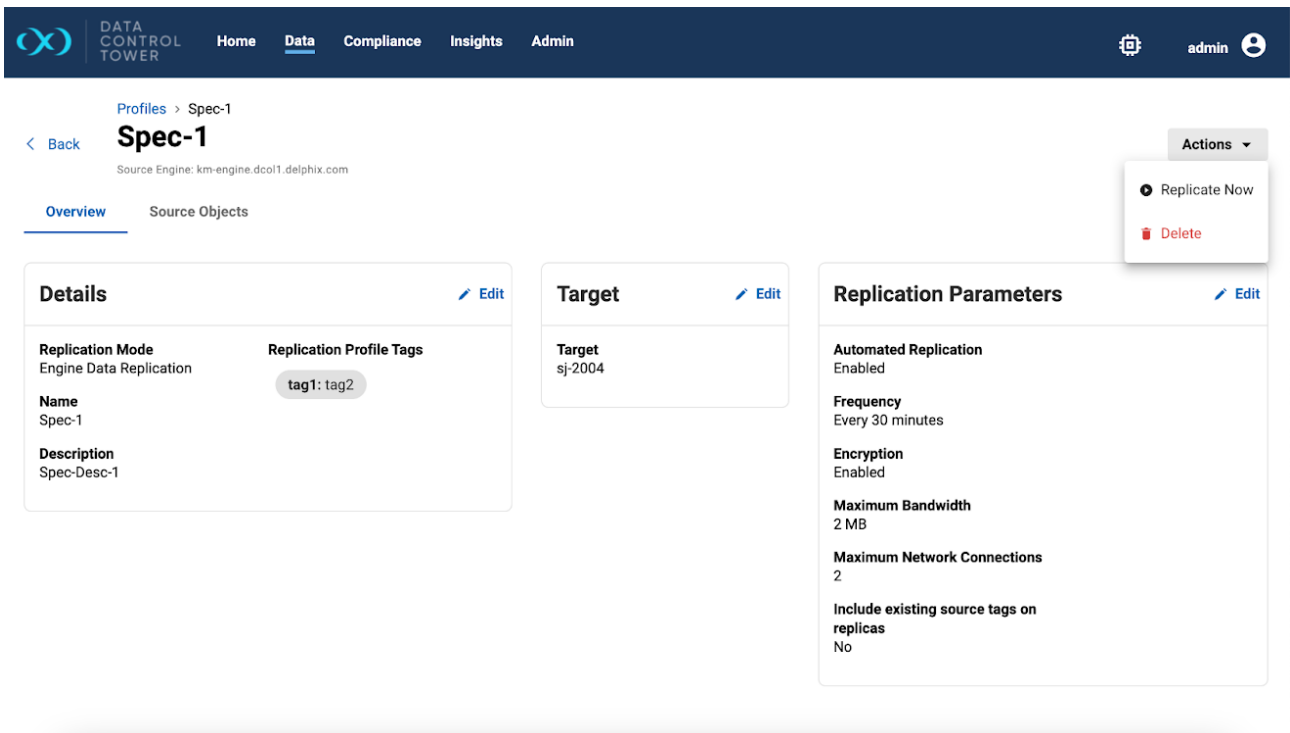
- **Overview tab**: Organized into four informative tiles:
  - **Details tile**: Displays replication mode, name, and description.
  - **Target tile**: Shows the target host name.

- **Replication Parameters tile**: Contains settings like Automated Replication, Frequency, Encryption, Bandwidth, Connections, and Tag Inclusion.
- **Received Replicas list (DCT 19.0.0 and up)**: Access the Received Replicas list via **Data** > **Data Engines** > **Engine Details View** > **Replication tab**, which provides a comprehensive list of all received replicas.
- **Execution History tab**: Shows the history of executions, including:
  - Last execution status and timestamp.
  - Start Time, End Time, Duration, Data Lag.
  - Edit options in each tile, accessible via the top-right corner.

7.2.6.5.5.3 API endpoints

- **Create Replication Profile**

```
POST /v3/replication-profiles
```

- **Update Replication Profile**

```
PATCH /v3/replication-profiles/{replicationProfileId}
```

- **Delete Replication Profile**

```
DELETE /v3/replication-profiles/{replicationProfileId}
```

- **Execute Replication Profile**

```
POST /v3/replication-profiles/{replicationProfileId}/execute
```

7.2.6.5.5.4 Key Parameters

- `replication_mode` : Profile's replication mode.
- `description` : Brief description.
- `schedule` : Replication schedule.
- `targetEngineId` , `targetHost` , `targetPort` : Target engine's ID, hostname, and port.
- `vdb_ids` , `dsource_ids` : Lists of VDBs and dSources to replicate.
- `tags` : Key-value pairs for categorizing profiles.

7.2.6.5.5.5 Example API Calls

- **Create Replication Profile**

```
curl -X 'POST' \
  'https://localhost/v3/replication-profiles' \
  -H 'accept: application/json' \
  -H 'Authorization: apk
1.4rJrWuVbB54W53ufEn6f5k8zDM9RUtOH4ciJtLGeVZNO0N6HnFt30WHMyj9WynKD' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "replication-profile-name",
  "replication_mode": "ENGINE_DATA_REPLICATION",
  "engine_id": 1,
  "target_engine_id": 2,
  "target_port": 8415,
  "description": "description",
  "schedule": "0 0 0 * * ?",
  "tags": [
    {
      "key": "key1",
      "value": "value1"
    },
    {
      "key": "key2",
      "value": "value2"
    }
  ],
  "enable_tag_replication": true,
  "bandwidth_limit": 0,
  "number_of_connections": 1,
  "encrypted": false,
  "automatic_replication": false,
  "use_system_socks_setting": false,
  "group_ids": [
    "1-GROUP-1"
  ]
}'
```

- **Update Replication Profile**

```
curl -X 'PATCH' \
  'https://localhost/v3/replication-profiles/1-REPLICATION_SPEC-68' \
  -H 'accept: application/json' \
  -H 'Authorization: apk
1.4rJrWuVbB54W53ufEn6f5k8zDM9RUtOH4ciJtLGeVZNO0N6HnFt30WHMyj9WynKD' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "string",
  "description": "string",
  "target_engine_id": "string",
  "target_host": "string",
  "target_port": 65535,
  "schedule": "string",
```

```
    "vdb_ids": [
      "string"
    ],
    "dsource_ids": [
      "string"
    ],
    "cdb_ids": [
      "string"
    ],
    "vcdb_ids": [
      "string"
    ],
    "group_ids": [
      "string"
    ],
    "replicate_entire_engine": false,
    "bandwidth_limit": 0,
    "number_of_connections": 16,
    "encrypted": true,
    "automatic_replication": true,
    "use_system_socks_setting": true
}'
```

- **Delete Replication Profile**

```
curl -X 'DELETE' \
  'https://localhost/v3/replication-profiles/1-REPLICATION_SPEC-68' \
  -H 'accept: application/json' \
  -H 'Authorization: apk
1.4rJrWuVbB54W53ufEn6f5k8zDM9RUtOH4ciJtLGeVZNO0N6HnFt30WHMyj9WynKD'
```

7.2.6.5.5.6 Other considerations

- **Password Vaults**: Not supported for replication operations.
- **Profile IDs**: Find IDs through the `GET /v3/replication-profiles` endpoint.

7.2.6.5.5.7 Limitations

- **GUI operations**: Currently, only profile creation is supported through the GUI. Future updates will expand GUI functionalities to cover more operations.

7.2.6.6 Replication relationship

7.2.6.6.1 Overview

DCT 11.0 release adds a functionality to get the primary and replica objects for a given object in DCT. Whenever DCT identifies that a replica object has been added to the engine, it creates a job to fetch the primary object information from its source engine. In addition, when replica objects are updated with primary object information, the primary objects are also updated with their children (replicas).

This feature requires that both source and target engines are registered in DCT.

7.2.6.6.2 Prerequisites

A running instance of DCT, two instances of Delphix Engines, and some of the dSource and VDBs on one of those engines. A replication profile needs to be created from an engine that contains dSource and VDBs to replicate to another engine.

7.2.6.6.3 Replication details

A replication details box will appear on the VDB or dSource tab showing the **Source Engine** and **Source VDB/ dSource** for replication.
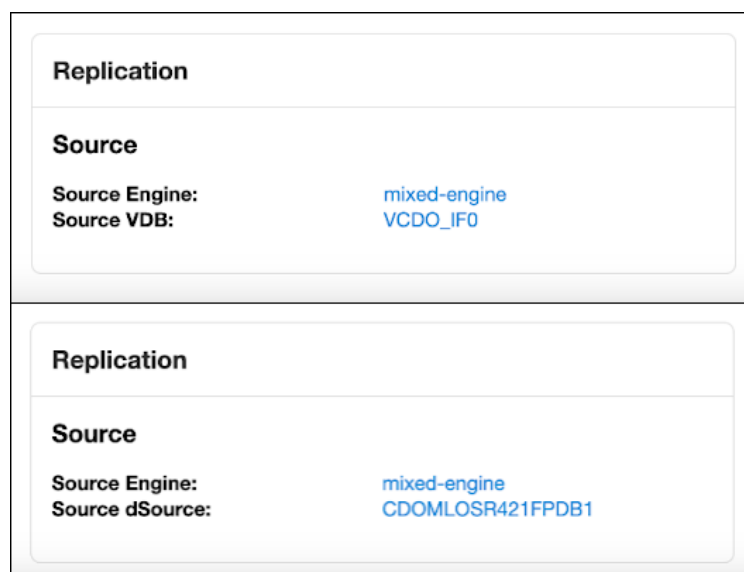
7.2.6.6.4 API changes

New fields have been added in the following APIs:

- GET - /v3/dsources
- GET - /v3/dsources/{dsourceId}
- POST - /v3/dsources/search
- GET - /v3/vdbs

- GET - /v3/vdbs/{dsourceId}
- POST - /v3/vdbs/search

The new fields are as follows:

- primary_object_id
- primary_engine_id
- primary_engine_name
- replicas
    - replica_id
    - replica_engine_id
    - replica_engine_name
    - replica_namespace_id

7.2.6.6.4.1 VDB replica and primary object sample response:

```
curl --location 'http://localhost:8080/v3/vdbs' \
--header 'Authorization: apk {{authToken}}'

{
    "items": [
        {
            "id": "1-ORACLE_DB_CONTAINER-11",
            "database_type": "Oracle",
            "name": "VCDO_IF0",
            "namespace_id": "1-NAMESPACE-2",
            "namespace_name": "ip-10-110-221-77-1",
            "is_replica": true,
            "is_locked": false,
            "database_version": "19.3.0.0.0",
            "size": 0,
            "storage_size": 2778624,
            "engine_id": "1",
            "masked": false,
            "content_type": "PDB",
            "parent_timeflow_timestamp": "2023-10-16T10:11:02Z",
            "parent_timeflow_timezone": "America/New_York,EDT-0400",
            "environment_id": "1-UNIX_HOST_ENVIRONMENT-3",
            "ip_address": "10.43.89.210",
            "fqdn": "ora-src.dcol1.delphix.com",
            "parent_id": "1-ORACLE_DB_CONTAINER-8",
            "parent_dsource_id": "1-ORACLE_DB_CONTAINER-8",
            "group_name": "Untitled",
            "engine_name": "e1",
            "cdb_id": "1-ORACLE_SINGLE_CONFIG-24",
            "creation_date": "2023-10-16T10:38:39.05Z",
            "hooks": {
                "pre_refresh": [],
```

```
                "post_refresh": [],
                "pre_self_refresh": [],
                "post_self_refresh": [],
                "pre_rollback": [],
                "post_rollback": [],
                "configure_clone": [],
                "pre_snapshot": [],
                "post_snapshot": [],
                "pre_start": [],
                "post_start": [],
                "pre_stop": [],
                "post_stop": []
            },
            "config_params": {
                "_cdb_disable_pdb_limit": "TRUE",
                "audit_file_dest": "'/u01/app/oracle/admin/CDOMLOSR421F/adump'",
                "audit_trail": "'DB'",
                "compatible": "'19.0.0'",
                "diagnostic_dest": "'/u01/app/oracle'",
                "dispatchers": "'(PROTOCOL=TCP) (SERVICE=CDOMLOSR421FXDB)'",
                "enable_pluggable_database": "TRUE",
                "log_archive_format": "'%t_%s_%r.dbf'",
                "max_pdbs": "4098",
                "memory_max_target": "1342177280",
                "memory_target": "1342177280",
                "nls_language": "'AMERICAN'",
                "nls_territory": "'AMERICA'",
                "open_cursors": "300",
                "processes": "300",
                "remote_login_passwordfile": "'EXCLUSIVE'"
            },
            "mount_point": "/mnt/provision",
            "current_timeflow_id": "1-ORACLE_TIMEFLOW-11",
            "vdb_restart": false,
            "is_appdata": false,
            "primary_object_id": "1-ORACLE_DB_CONTAINER-6",
            "primary_engine_id": "1",
            "primary_engine_name": "e1"
        },
        {
            "id": "1-ORACLE_DB_CONTAINER-6",
            "database_type": "Oracle",
            "name": "VCDO_IF0",
            "is_replica": false,
            "is_locked": false,
            "database_version": "19.3.0.0.0",
            "size": 794755072,
            "storage_size": 32859648,
            "engine_id": "1",
            "status": "RUNNING",
            "masked": false,
            "content_type": "PDB",
```

```
            "parent_timeflow_timestamp": "2023-10-16T10:11:02Z",
            "parent_timeflow_timezone": "America/New_York,EDT-0400",
            "environment_id": "1-UNIX_HOST_ENVIRONMENT-1",
            "ip_address": "10.43.89.210",
            "fqdn": "ora-src.dcol1.delphix.com",
            "parent_id": "1-ORACLE_DB_CONTAINER-2",
            "parent_dsource_id": "1-ORACLE_DB_CONTAINER-2",
            "group_name": "Untitled",
            "engine_name": "e1",
            "cdb_id": "1-ORACLE_SINGLE_CONFIG-2",
            "creation_date": "2023-10-16T10:38:39.05Z",
            "hooks": {
                "pre_refresh": [],
                "post_refresh": [],
                "pre_self_refresh": [],
                "post_self_refresh": [],
                "pre_rollback": [],
                "post_rollback": [],
                "configure_clone": [],
                "pre_snapshot": [],
                "post_snapshot": [],
                "pre_start": [],
                "post_start": [],
                "pre_stop": [],
                "post_stop": []
            },
            "config_params": {
                "_cdb_disable_pdb_limit": "TRUE",
                "audit_file_dest": "'/u01/app/oracle/admin/CDOMLOSR421F/adump'",
                "audit_trail": "'DB'",
                "compatible": "'19.0.0'",
                "diagnostic_dest": "'/u01/app/oracle'",
                "dispatchers": "'(PROTOCOL=TCP) (SERVICE=CDOMLOSR421FXDB)'",
                "enable_pluggable_database": "TRUE",
                "log_archive_format": "'%t_%s_%r.dbf'",
                "max_pdbs": "4098",
                "memory_max_target": "1342177280",
                "memory_target": "1342177280",
                "nls_language": "'AMERICAN'",
                "nls_territory": "'AMERICA'",
                "open_cursors": "300",
                "processes": "300",
                "remote_login_passwordfile": "'EXCLUSIVE'"
            },
            "mount_point": "/mnt/provision",
            "current_timeflow_id": "1-ORACLE_TIMEFLOW-6",
            "vdb_restart": false,
            "is_appdata": false,
            "replicas": [
                {
                    "replica_id": "1-ORACLE_DB_CONTAINER-11",
                    "replica_engine_id": "1",
```

```
                            "replica_engine_name": "e1",
                            "replica_namespace_id": "1-NAMESPACE-2"
                        }
                    ]
                }
            ],
            "response_metadata": {
                "total": 2
            }
        }
```

7.2.6.6.4.2 Sample response for dSource change:

```
curl --location 'http://localhost:8080/v3/dsources/search' \
--header 'Content-Type: application/json' \
--header 'Authorization: apk {{authToken}}' \
--data '{
    "filter_expression" : "engine_id EQ '\''1'\''"
}'


{
    "items": [
        {
            "id": "1-ORACLE_DB_CONTAINER-10",
            "database_type": "Oracle",
            "name": "CDOMSHSR6706PDB2-NO-CHILD",
            "namespace_id": "1-NAMESPACE-2",
            "namespace_name": "ip-10-110-221-77-1",
            "is_replica": true,
            "database_version": "19.3.0.0.0",
            "content_type": "PDB",
            "data_uuid": "46f1613b223b1dd5364bdc3ecfd2755d",
            "storage_size": 180355584,
            "creation_date": "2023-10-16T10:09:00.567Z",
            "group_name": "Untitled",
            "engine_id": "1",
            "source_id": "1-ORACLE_PDB_CONFIG-20",
            "engine_name": "e1",
            "cdb_id": "1-ORACLE_SINGLE_CONFIG-23",
            "current_timeflow_id": "1-ORACLE_TIMEFLOW-7",
            "is_appdata": false,
            "primary_object_id": "1-ORACLE_DB_CONTAINER-4",
            "primary_engine_id": "1",
            "primary_engine_name": "e1"
        },
        {
            "id": "1-ORACLE_DB_CONTAINER-12",
            "database_type": "Oracle",
```

```
        "name": "CDOMLOSR421FPDB2",
        "is_replica": false,
        "database_version": "19.3.0.0.0",
        "content_type": "PDB",
        "data_uuid": "837bbc258a479ff86dbce5656beeba2a",
        "storage_size": 236433920,
        "creation_date": "2023-10-17T10:19:19.496Z",
        "group_name": "Untitled",
        "enabled": true,
        "engine_id": "1",
        "source_id": "1-ORACLE_PDB_CONFIG-7",
        "status": "RUNNING",
        "engine_name": "e1",
        "cdb_id": "1-ORACLE_SINGLE_CONFIG-2",
        "current_timeflow_id": "1-ORACLE_TIMEFLOW-12",
        "is_appdata": false
    },
    {
        "id": "1-ORACLE_DB_CONTAINER-2",
        "database_type": "Oracle",
        "name": "CDOMLOSR421FPDB1",
        "is_replica": false,
        "database_version": "19.3.0.0.0",
        "content_type": "PDB",
        "data_uuid": "834a794e51ec096059922dc06625fb66",
        "storage_size": 245758976,
        "creation_date": "2023-10-16T10:08:35.425Z",
        "group_name": "Untitled",
        "enabled": true,
        "engine_id": "1",
        "source_id": "1-ORACLE_PDB_CONFIG-6",
        "status": "RUNNING",
        "engine_name": "e1",
        "cdb_id": "1-ORACLE_SINGLE_CONFIG-2",
        "current_timeflow_id": "1-ORACLE_TIMEFLOW-2",
        "is_appdata": false,
        "replicas": [
            {
                "replica_id": "1-ORACLE_DB_CONTAINER-8",
                "replica_engine_id": "1",
                "replica_engine_name": "e1",
                "replica_namespace_id": "1-NAMESPACE-2"
            }
        ]
    },
    {
        "id": "1-ORACLE_DB_CONTAINER-21",
        "database_type": "Oracle",
        "name": "DSOURCE-WITH-CUSTOM-POLICY",
        "is_replica": false,
        "database_version": "19.3.0.0.0",
        "content_type": "PDB",
```

```
        "data_uuid": "dd4216e99ee17e8a7afa97cce012aa42",
        "storage_size": 181809664,
        "creation_date": "2023-10-18T13:09:30.299Z",
        "group_name": "Untitled",
        "enabled": true,
        "engine_id": "1",
        "source_id": "1-ORACLE_PDB_CONFIG-11",
        "status": "RUNNING",
        "engine_name": "e1",
        "cdb_id": "1-ORACLE_SINGLE_CONFIG-3",
        "current_timeflow_id": "1-ORACLE_TIMEFLOW-21",
        "is_appdata": false
    },
    {
        "id": "1-ORACLE_DB_CONTAINER-4",
        "database_type": "Oracle",
        "name": "CDOMSHSR6706PDB2-NO-CHILD",
        "is_replica": false,
        "database_version": "19.3.0.0.0",
        "content_type": "PDB",
        "data_uuid": "46f1613b223b1dd5364bdc3ecfd2755d",
        "storage_size": 190537728,
        "creation_date": "2023-10-16T10:09:00.567Z",
        "group_name": "Untitled",
        "enabled": true,
        "engine_id": "1",
        "source_id": "1-ORACLE_PDB_CONFIG-10",
        "status": "RUNNING",
        "engine_name": "e1",
        "cdb_id": "1-ORACLE_SINGLE_CONFIG-3",
        "current_timeflow_id": "1-ORACLE_TIMEFLOW-4",
        "is_appdata": false,
        "replicas": [
            {
                "replica_id": "1-ORACLE_DB_CONTAINER-10",
                "replica_engine_id": "1",
                "replica_engine_name": "e1",
                "replica_namespace_id": "1-NAMESPACE-2"
            }
        ]
    },
    {
        "id": "1-ORACLE_DB_CONTAINER-8",
        "database_type": "Oracle",
        "name": "CDOMLOSR421FPDB1",
        "namespace_id": "1-NAMESPACE-2",
        "namespace_name": "ip-10-110-221-77-1",
        "is_replica": true,
        "database_version": "19.3.0.0.0",
        "content_type": "PDB",
        "data_uuid": "834a794e51ec096059922dc06625fb66",
        "storage_size": 226567680,
```

```
            "creation_date": "2023-10-16T10:08:35.425Z",
            "group_name": "Untitled",
            "engine_id": "1",
            "source_id": "1-ORACLE_PDB_CONFIG-22",
            "engine_name": "e1",
            "cdb_id": "1-ORACLE_SINGLE_CONFIG-24",
            "current_timeflow_id": "1-ORACLE_TIMEFLOW-9",
            "is_appdata": false,
            "primary_object_id": "1-ORACLE_DB_CONTAINER-2",
            "primary_engine_id": "1",
            "primary_engine_name": "e1"
        }
    ],
    "response_metadata": {
        "total": 6
    }
}
```

7.2.6.7 Replica insights

7.2.6.7.1 Overview

DCT 20.0.0 introduces new replication insights, such as the replication policy list of an engine and a detailed view of a replica, in addition to failback support.

> A source and target engine connected to DCT with one or more replication profiles.

- **Replication policy list of an engine**
  This feature allows users to view the list of replication policies associated with a particular engine, and is available by navigating to **Data > Data Engines > Engine Details View > Replication tab**.
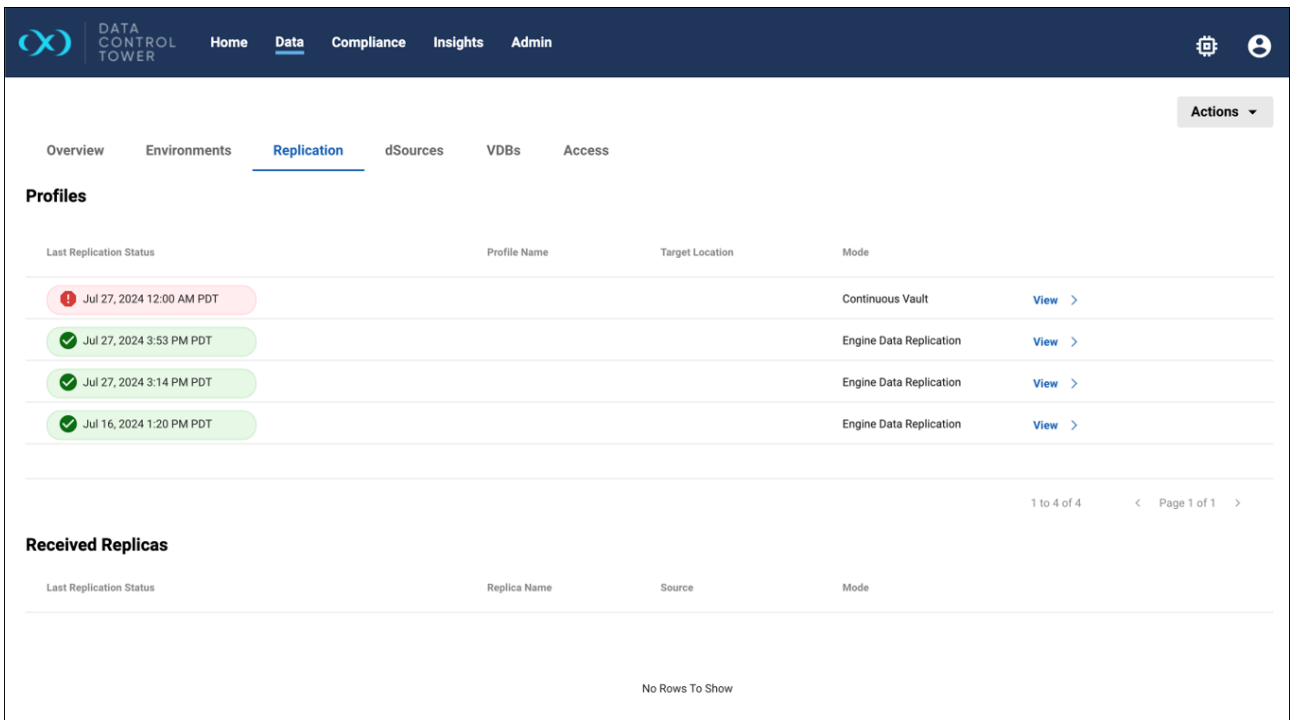
- **Replica detail view**
  This feature allows users to see detailed information about a received replica. Use the **View** button available for each replica in the received replica list under **Data > Data Engines > Engine Details View > Replication tab**.
  - In the **Overview** tab, details of the replica and its corresponding replication policy's details are displayed. If the replication policy is not available on DCT, a banner is shown to indicate.
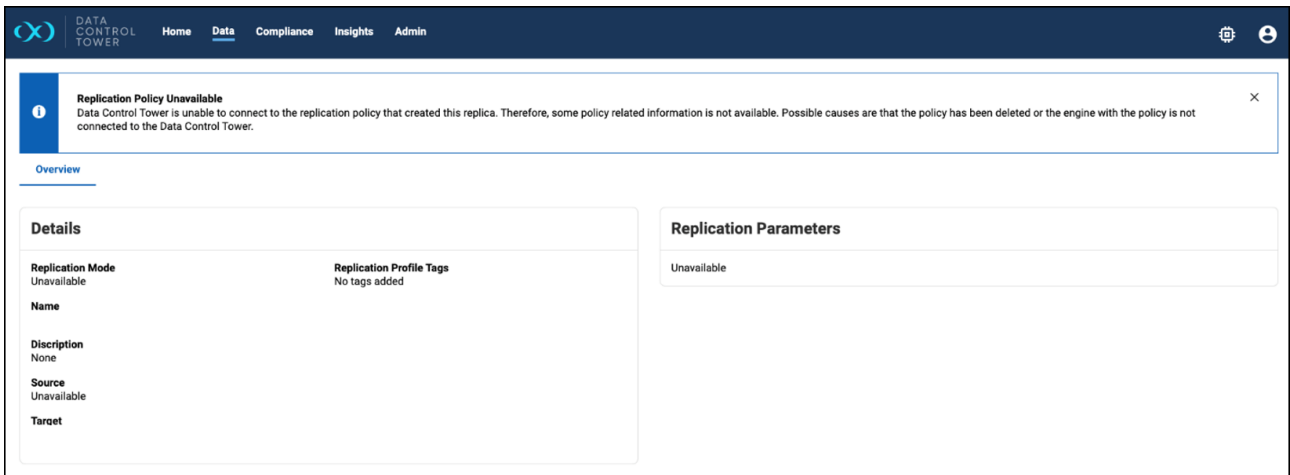
- **Failback support**
  This is a new checkbox option for creating a failback replica during failover that can be launched from the Failover window at **Data > Replication > Replication Policy or Replica Details View > Actions > Start Failover**.

7.3 Insight reports

7.3.1 Activity Audit Log summary

The Activity Audit Log Summary provides a high-level audit log summary capturing the utilization of DCT by displaying user activity and the historical count of actions executed within the platform. This concise report enables stakeholders to quickly identify trends, monitor user engagement, and assess the overall effectiveness of data governance processes.

7.3.2 Block Storage report

This report provides users with a comprehensive view of storage usage across different engines. With this report, users can easily identify the engines that are utilizing the most storage and take necessary action to optimize storage usage.

In addition, there are three distinct charts to help better understand storage usage across engines. The search can be utilized in order to filter out engines. All three charts only include data for engines that match the search criteria. They are exemplified below.

- **Top N Storage Capacity chart**: Shows a breakdown of the current storage usage across the top 10 or 15 largest engines.
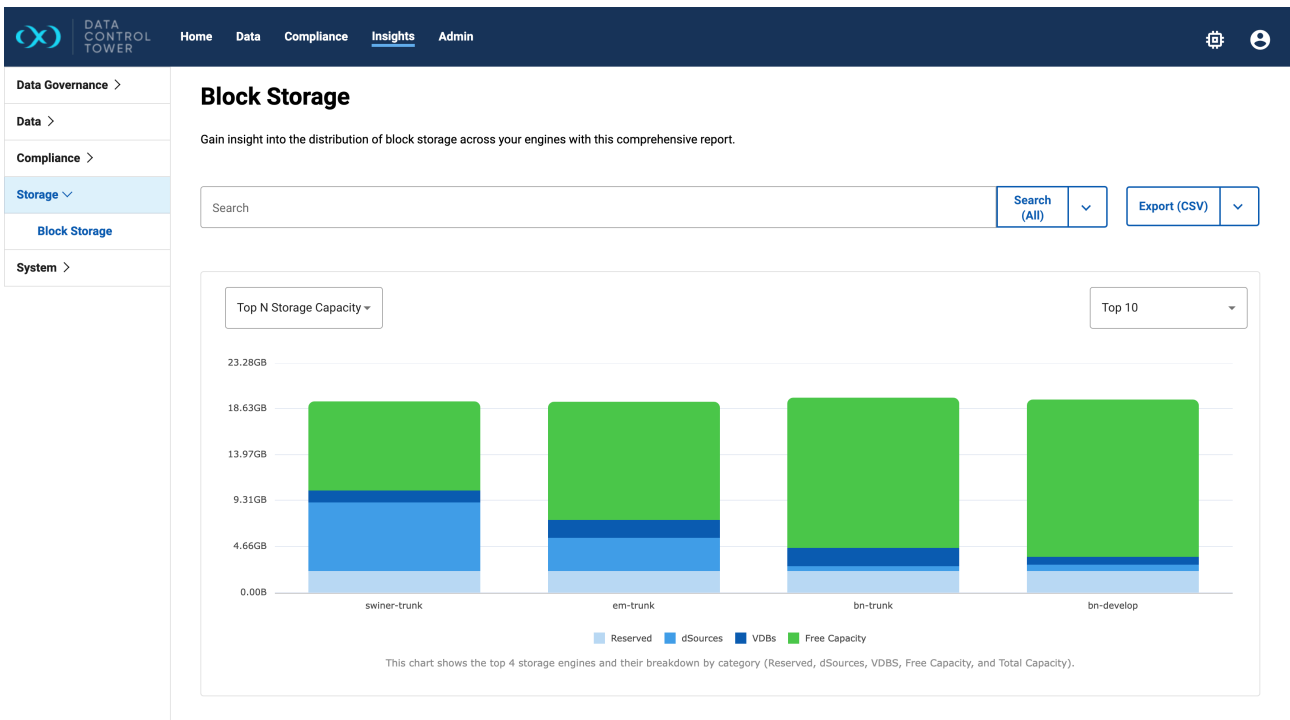
- **Capacity chart**: Shows the historical storage usage and free space across all engines. This chart shows data for the past 3 or 6 month and supports an optional projection of used capacity for that same timeframe.

> Data for the current day (UTC time zone) includes additional system data that is not taken into account for the historical data points.

Block Storage

Gain insight into the distribution of block storage across your engines with this comprehensive report.
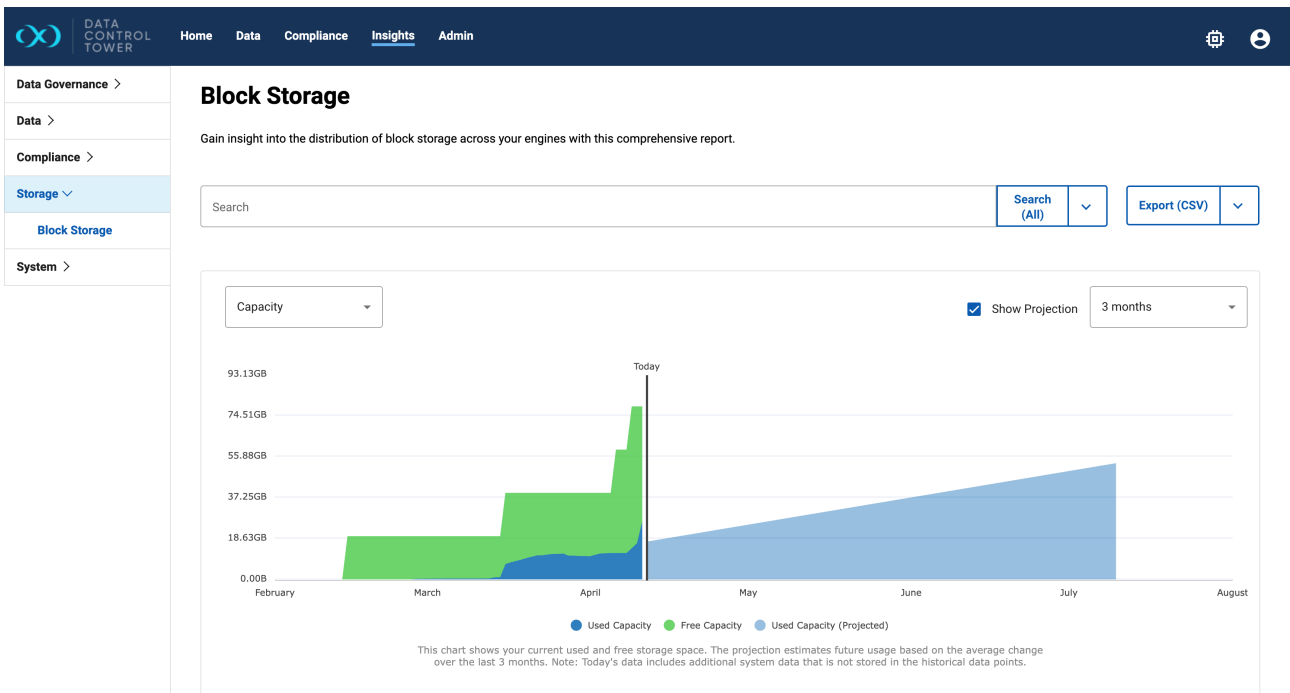
- **Top 5 Engines Storage Usage chart**: Shows the historical usage capacity for the 5 largest engines. This chart also shows data for the past 3 or 6 month with an optional projection of used capacity.

> ℹ️ Data for the current day (UTC time zone) includes additional system data that is not taken into account for the historical data points.

Block Storage

Gain insight into the distribution of block storage across your engines with this comprehensive report.
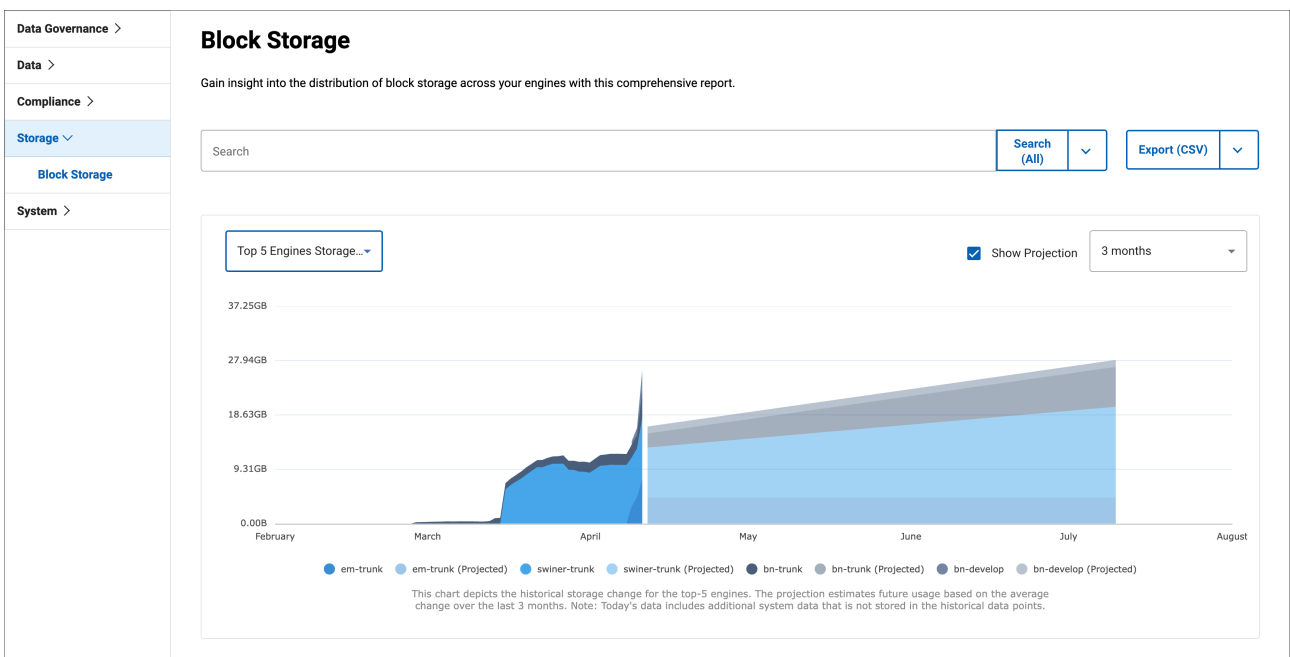
7.3.3 Central Governance insights

DCT provides global reporting of real-time statuses. This section will break down all of the reports in the **Insights** section of DCT.

> ⓘ All insight dashboards can be exported to CSV or JSON format.

7.3.4 Compliance Job Executions report

The Compliance Job Executions Report dashboard offers a comprehensive overview of the compliance jobs executed within DCT, by presenting key metrics that enable stakeholders to assess the efficiency and effectiveness of their data governance efforts.

7.3.5 Continuous Compliance Engine Performance report

The Engine Performance dashboard provides users with a comprehensive view of performance information for Delphix Continuous Compliance Engines. The performance metrics include system CPU utilization, system memory usage, latency and throughput details for Disk protocols, and network throughput.
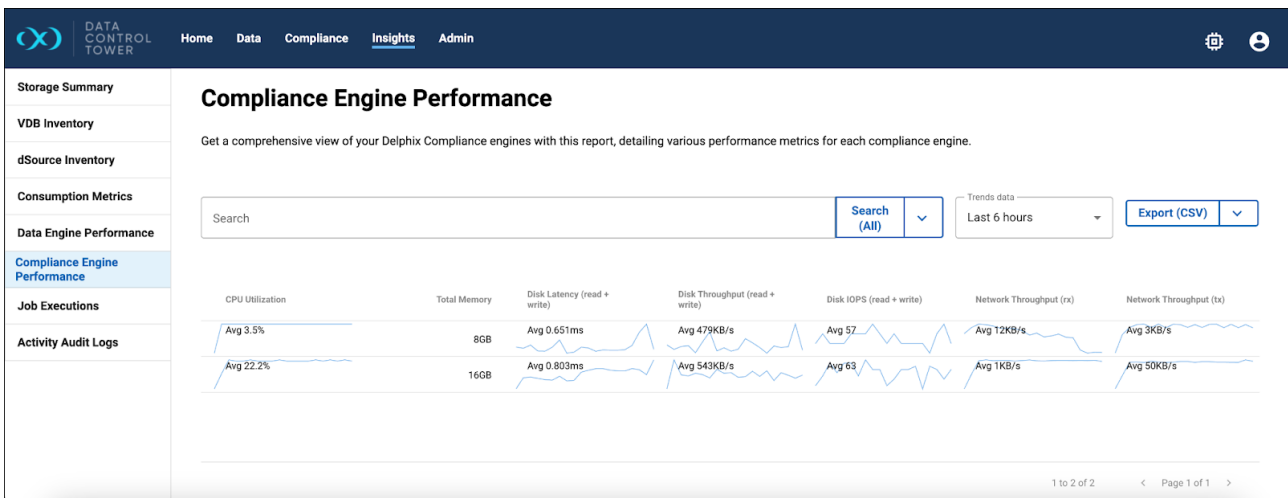
7.3.6 Continuous Data Engine Performance report

The Engine Performance dashboard provides users with a comprehensive view of performance information for Delphix Continuous Data Engines. The performance metrics include system CPU utilization, system memory usage, latency, IOPS, and network throughput details for Disk, NFS, and iSCSI protocols.
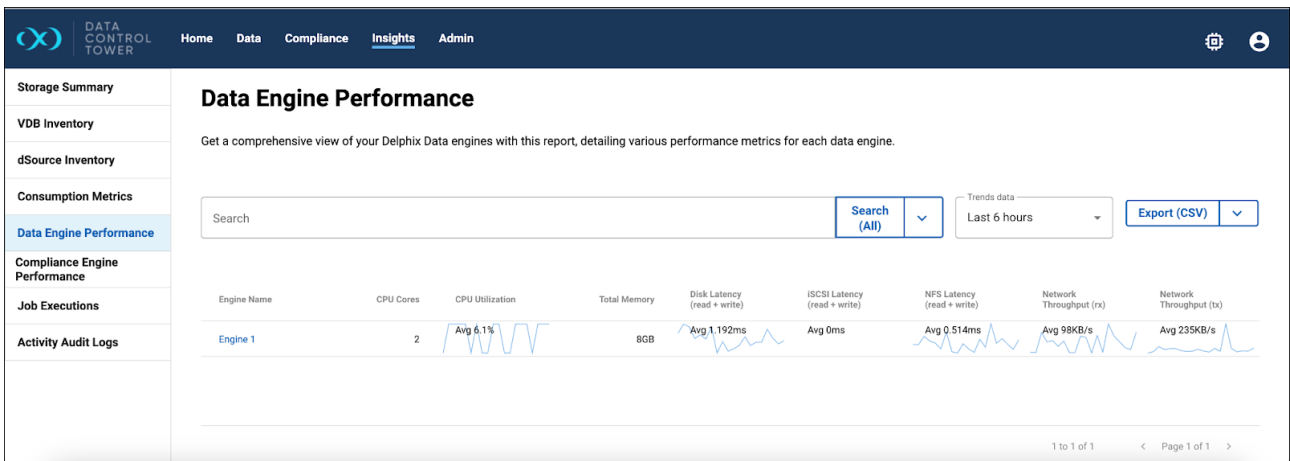
By default, a fix set of data columns are shown. Users have the flexibility to add or remove columns based on their preferences.
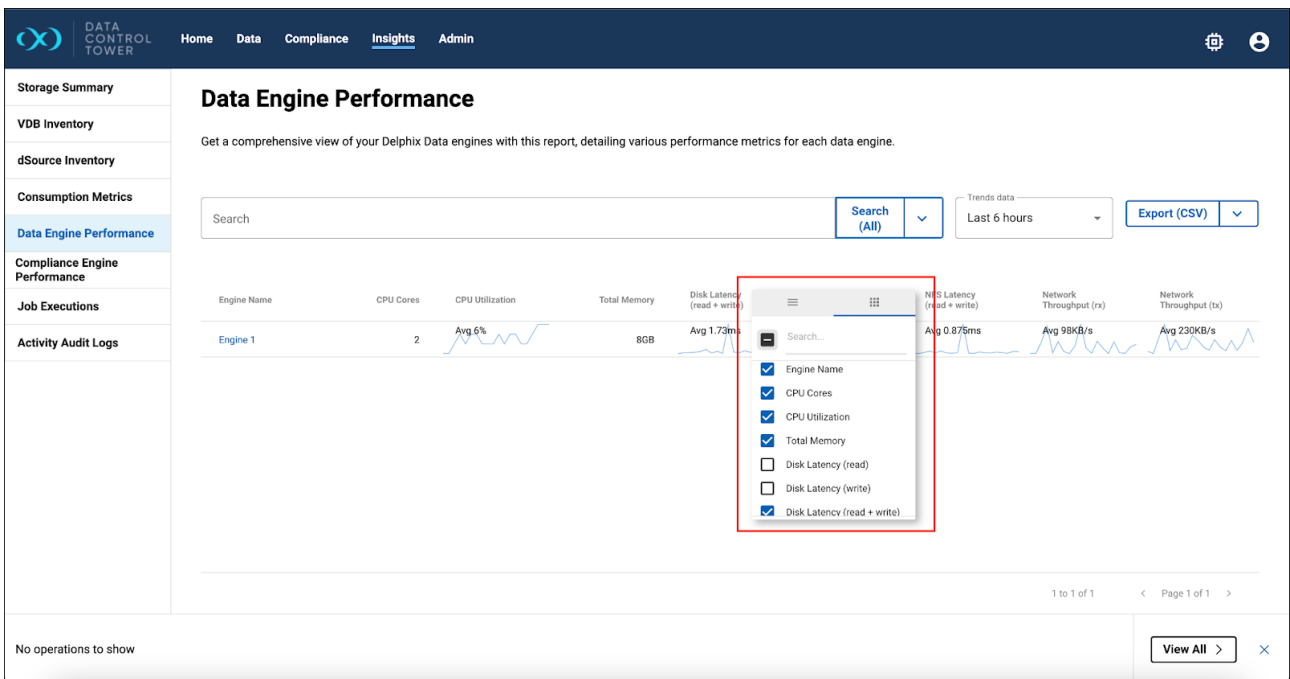
7.3.7 Data Risk report

The Data Risk report is available in DCT version 16.0.0 and above. It provides visibility into data risk across your organization by aggregating metrics and insights from all Continuous Compliance Engines, including their environments and jobs, to give governance stakeholders a global overview of potential risks in sensitive data. More information on this report can be found in the Data Risk report[117] page.

---

117 https://dct.delphix.com/docs/latest/data-risk-report
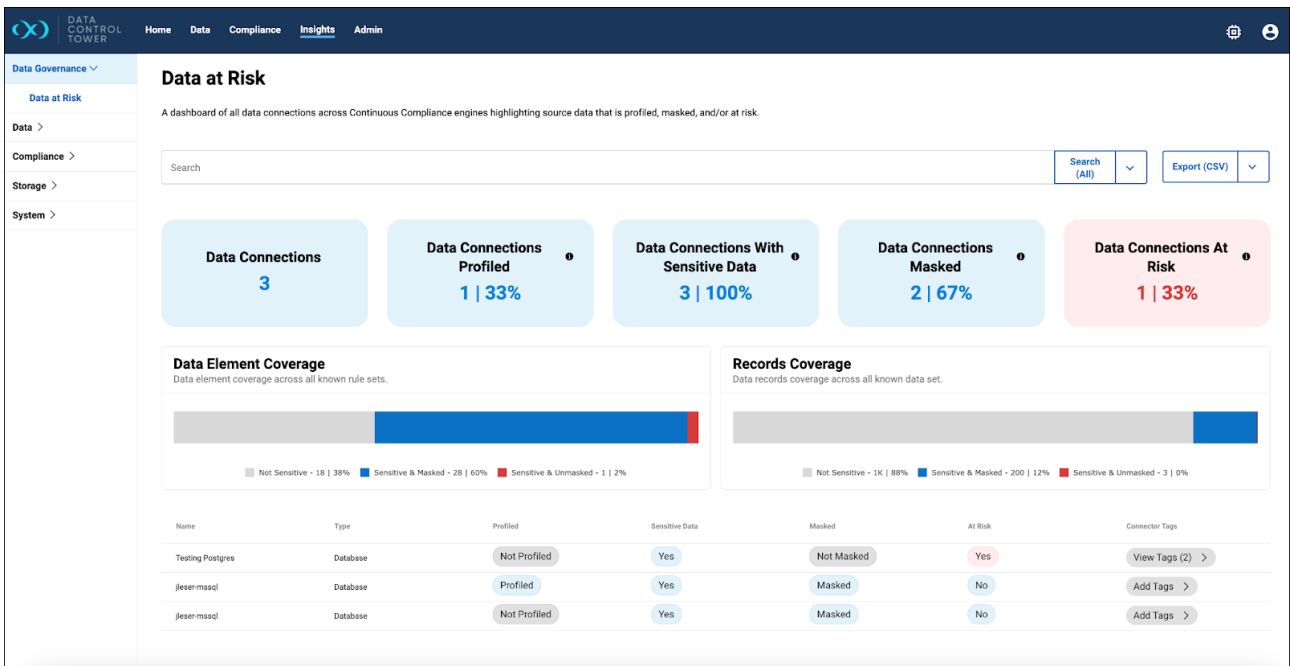
7.3.8 dSource Inventory report

The dSource Inventory report provides users with a comprehensive list of all the dSources created in the DCT platform and their identification metadata.

7.3.9 Global State report

The Compliance Engine (Global) State report provides the users with an overview of which engines are in sync and can be used to horizontally scale a compliance job. The users should use the **Engine Configuration Revision Hash** (Global Object Revision Hash) to determine if engines are synced – they have identical global object like (algorithms, rule sets, domains, etc.). Search filtering and sorting can be used to find all engines that are in-sync, and also determine if an engine has fallen out of sync. If the user wants to take action by then syncing an un-synced engine, they can do so from the Hyperscale Orchestrators' details page via the **Manually Deploy Job Dependencies** action menu.

7.3.10 Source Ingestion Metrics

The Data Source Ingestion Metrics dashboard is designed to help users find their virtualization source ingestion metrics, which are often required for contract renewal purposes. The dashboard contains the data sources and informs the user of the total size of that ingestion source.

7.3.11 Storage Savings report

The Storage savings report illustrates how Delphix is effectively reducing storage costs through the implementation of data virtualization. The data in the table shows space metrics for each dSource and its child/descendant VDBs. The tiles above show the aggregates across all the pages of the table. The tiles will update to reflect the results in the table as the result of any search filtering.

Savings are reported as the storage you would need without Delphix versus the storage Delphix is using for those same objects. The difference between the two is then used to show the savings and savings percentages.

Delphix's dataset timeflows/timelines enable users to navigate through different data states at various time points, mimicking having multiple dataset copies, thus offering significant storage savings. This functionality, known as "VDB Refresh," is typically external, requiring separate data copies. Users interested in the storage Delphix saves for only the current data state can toggle the current active timeline columns.

These current active timeline metrics are available on engine versions 22.0.0.0 or later. Columns for these metrics are hidden by default if any dSources come from older engines but will appear when all dSources are from compatible newer engines. Although the default view focuses on the current active timeflow, metrics for all timeflows remain accessible via the column menu for users needing broader data insights.

7.3.12 VDB Inventory report

The VDB Inventory report provides users with a comprehensive list of all the Virtual Databases (VDBs) created in the DCT platform and their identification metadata.

7.3.13 Oracle Container Database Inventory report

This report provides a comprehensive list of all Container Databases (CDBs) and virtual Container Databases (vCDBs). This report features an expandable list that reveals pluggable data associated with each CDB or vCDB, providing in-depth visibility into your Oracle database environment at a glance.

7.3.14 Data Risk report

Data Control Tower (DCT) version 16.0.0 and above introduces the Data Risk report, offering an in-depth view of your organization's data security posture by presenting global visibility metrics aggregated across all data connections of every connected Continuous Compliance Engine. This is crucial for identifying and mitigating potential data risks in sensitive data.

> ℹ️ While the DCT version must be 16.0.0 or above, the Continuous Compliance Engine enhancements from version 20.0.0.0 and above are required to obtain database record counts by connectors. Thus, connected Continuous Compliance Engines with versions prior to 20.0.0.0 will not display record counts, and a warning message will replace the Records Coverage chart.

Records Coverage

Data records coverage across all known data set.

Incomplete Data

Cumulative record counts will be inaccurate as one or more rulesets are missing record counts. This will occur when a compliance engine with version earlier than 20.0.0.0 is registered or when one or more rulesets have not finished their initial refresh.

> ℹ️ Currently, the report only includes database connectors. File and mainframe dataset connectors are not covered and will not appear in the report's UI table.

7.3.14.1 Feature list

- **Global visibility metrics**
  At the heart of the report are the global visibility metrics, presented through a combination of billboards and charts at the top of the page:
    - **Billboards**: Show data connection counts across five categories.
        i.   Total number of data connections
        ii. Data connections with at least one successful profiling job
        iii. Data connections identified with sensitive data
        iv. Data connections with at least one successful masking job
        v.   Data connections at risk (sensitive but not masked)
- **Detailed data insights**
    - **Charts**: Offer insights into the count of data elements (e.g., database columns) and records in three categories, listed below. Legends and tooltips provide exact counts and percentages for each category, enhancing the understanding of your data's sensitivity and protection status.
        i.   Not sensitive
        ii. Sensitive and masked
        iii. Sensitive and unmasked

- **Report Scheduling**
    - A new report type, `DATA_RISK_SUMMARY`, has been added to the Reporting Schedule. This allows users to schedule regular updates on data risk metrics via email, at predefined intervals, to allow continuous monitoring and timely risk assessment.
- **GUI additions**
    - Below the charts, a table presents detailed data risk metrics for each data connection. Users can customize this table by toggling additional fields on or off through the column menu, for more relevant data insights.

    - The report also supports exporting of information in both CSV and JSON formats. This includes both global metrics and per-connection metrics for in-depth data analysis and reporting.

7.3.14.1.1 Terminology and definitions

To help provide clarity and understand the definitions for key terms used within this report, refer to the definitions below.

Records Coverage
Data records coverage across all known data set.

Not Sensitive
1,442 | 88%

Not Sensitive - 1K | 88%   Sensitive & Masked - 200 | 12%   Sensitive & Unmasked - 3 | 0%
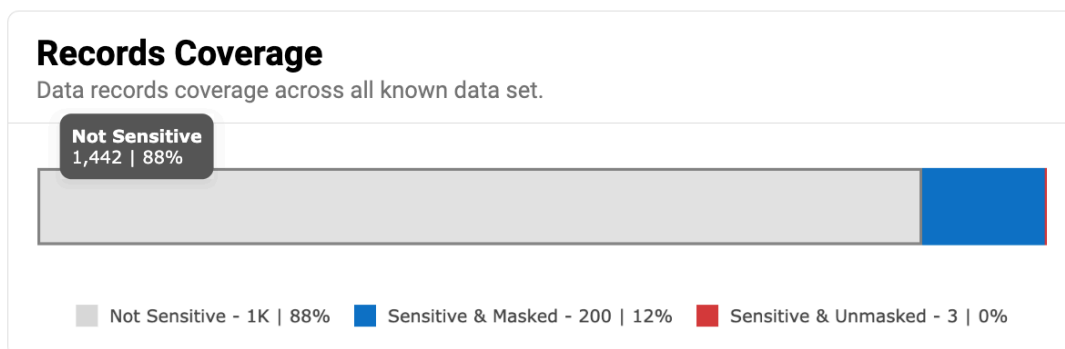
- **Is profiled**: A connector has undergone a profiling job.
- **Is sensitive data**: Sensitive data identified within a connector's data elements.
- **Is masked**: A successful masking job has been run on the connector's data.
- **Is at risk**: Sensitive data within a connector that has not been masked.
    - This is applied to data classified as sensitive (with at least one algorithm assignment), that has not had a successful masking job run. This definition is based on the masking job's perspective.

- Please note that data might be refreshed, potentially mislabeling it as *not at risk* despite changes in the actual data. This discrepancy arises as DCT may not detect that the underlying data has been updated, leading to potential inaccuracies.
- **Data elements total**: The total count of data elements associated with a connector's data.
- **Records total**: The total number of records detected for a connector's data.
- **Sensitive and unmasked**: Data elements or records that are sensitive but have not been masked.

7.4 Tabular data in DCT

Data Control Tower (DCT) leverages a common reporting framework that brings a polished user experience to the way you work with data tables. This suite of enhancements includes features like:

- **Customizable columns**
    - Resize columns
    - Show/hide columns dynamically
    - Rearrange columns for a tailored analysis process
- **Column-based filtering**
    - Filter data based on selected fields
    - Filter menu is automatically populated based on type of data in the column
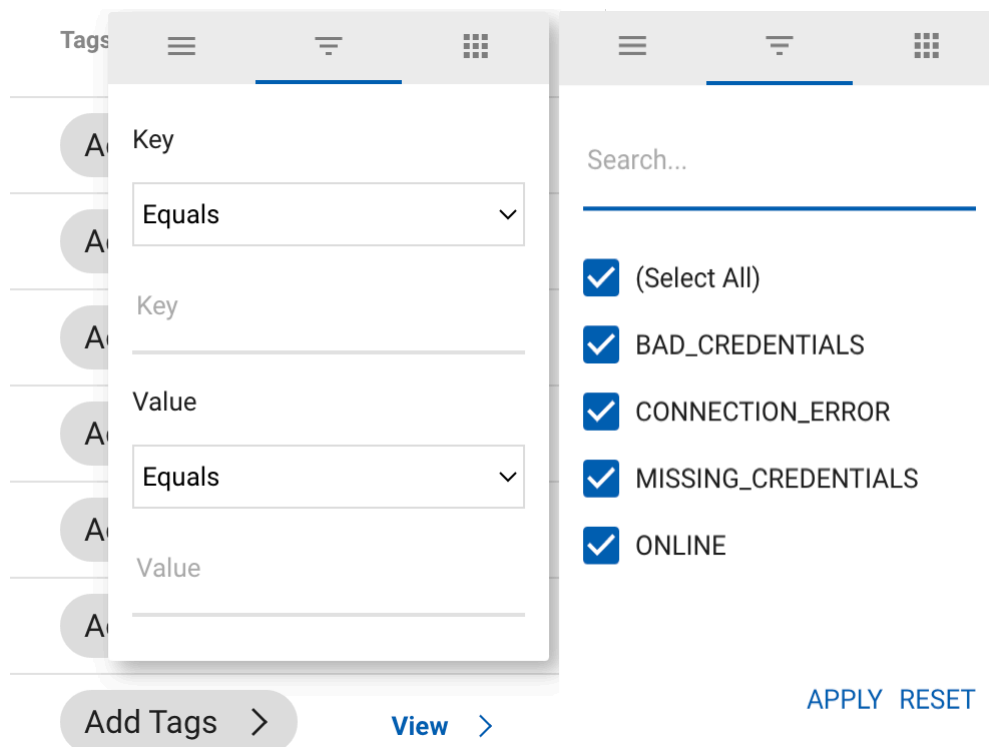
- **Search bar functionality**
    - Search for complex data types and expressions

Search	Search (All)	⌄

> ℹ️ Any search input or expression from the search bar will be "AND"ed with the filtering from the column menus – meaning search results obtained from the search bar will be combined using a logical "AND" with any filters that have been applied through column menus.
> Thus, the final set of results shown will only include entries that satisfy both the search criteria and the column filters, resulting in a narrower (or more specific) subset of the data.

7.5 Faults, alerts, and actions in DCT

7.5.1 Introduction

DCT 21.0.0 introduces the ability for DCT to ingest and list/search Continuous Data Engine faults, alerts, and actions (audit logs). These objects are gathered from registered engines via telemetry updates and can be examined by either the DCT API or the UI.

Here are the explanations of what a Fault, Alert, and Action mean in Data Control Tower (DCT):

- **Fault**: A fault in DCT represents a persistent issue that affects Delphix's behavior and requires manual resolution by the user. For example, a connectivity problem would be considered a fault. While Delphix attempts to resolve faults automatically when the underlying issue is fixed, there are situations where user intervention is necessary to clear the fault. Additionally, faults can be ignored, which prevents Delphix from posting the same fault for the same object in the future.
- **Alert**: An alert is a notification generated by the engine to inform users of specific events or conditions that have occurred. Unlike faults, alerts do not necessarily indicate a persistent problem that needs immediate attention. Instead, they serve as informational messages to make users aware of certain situations or changes within the system that might need attention but do not necessarily impact the engine's overall behavior.
- **Action (Audit Log)**: Actions refer to recorded events or operations within the engine, typically logged for auditing purposes. These actions are tracked in audit logs and provide a historical record of changes, activities, or operations performed in the engine. Actions are generally used for monitoring compliance and reviewing past events.

7.5.2 Prerequisites

To use these new APIs, you need a running DCT instance with at least one registered Continuous Data Engine.

7.5.3 API Changes

This feature introduces two new APIs for Continuous Data alerts and actions, and five new APIs for Continuous Data faults.

7.5.3.1 Alerts

- `GET /virtualization-alerts/history` – Fetch a list of all Continuous Data alerts.
- `POST /virtualization-alerts/history/search` – Search Continuous Data alerts.

7.5.3.1.1 Example response:

```
{
  "items": [
    {
      "id": "1-ALERT-1",
      "engine_id": "1",
      "alert_timestamp": "2024-08-06T07:09:45.74Z",
      "event": "alert.system.shutdown.management.initialconfig",
      "event_severity": "INFORMATIONAL",
      "event_title": "Initial configuration",
      "event_response": "Initial configuration",
      "event_description": "The management service is going down after initial
configuration.",
      "target_name": "system"
    },

  "response_metadata": {
    "total": 1
  }
```

7.5.3.2 Actions

- `GET /virtualization-actions/history` – Fetch a list of all Continuous Data actions.
- `POST /virtualization-actions/history/search` – Search Continuous Data actions.

7.5.3.2.1 Example response:

```
{
  "items": [
.    {
```

```
        "id": "1-ACTION-10210",
        "engine_id": "2",
        "action_type": "USER_LOGIN",
        "title": "USER_LOGIN",
        "details": "Log in as user \"admin\" from IP \"127.0.0.1\".",
        "start_time": "2024-08-16T19:56:59.463Z",
        "end_time": "2024-08-16T19:56:59.463Z",
        "user": "USER-2",
        "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36",
        "origin_ip": "172.16.124.16",
        "state": "COMPLETED",
        "work_source": "WEBSERVICE",
        "work_source_name": "admin",
        "work_source_principal": "admin"
    }
  ],
  "response_metadata": {
    "total": 1
  }
}
```

7.5.3.3 Faults

- `GET /virtualization-faults/history` – Fetch a list of all Continuous Data faults.
- `POST /virtualization-faults/history/search` – Search Continuous Data faults.
- `POST /virtualization-faults/resolveOrIgnore` – Mark a list of faults as resolved or ignored.
- `POST /virtualization-faults/{engineId}/resolveAll` – Resolve all active faults on an engine the user has permissions over as resolved.
- `POST /virtualization-faults/{faultId}/resolve` – Resolve or ignore an individual fault.

7.5.3.3.1 Example response:

```
{
  "items": [
    {
      "id": "1-FAULT-1",
      "engine_id": "1",
      "bundle_id": "fault.email.smtprequired",
      "target_name": "system",
      "title": "Password reset requires SMTP to be enabled",
      "description": "Password reset emails cannot be sent if SMTP is disabled or not
configured.",
      "fault_action": "Configure and enable SMTP, or disable the password reset
function.",
```

```
        "severity": "WARNING",
        "status": "RESOLVED",
        "date_diagnosed": "2024-08-06T07:10:45.836Z",
        "date_resolved": "2024-08-28T04:39:40.938Z"
    },
  ],

  "Response Metadata": {
      "total": 1
  }
}
```

7.5.4 GUI changes

Faults, Events (Alerts), and Audit (Actions) tabs are made available to the UI via the details page of a Continuous Data engine. By default, the faults table only shows **Active** faults. You can choose to show faults with other statuses by configuring the column filter for the **Status** column.

Implementation

These APIs were implemented to mimic the behavior of the corresponding Delphix engine APIs. In the case of actions and alerts, this is a straightforward list, search, and filter functionality. For faults, it is worth understanding the difference between the three APIs specific to resolving or ignoring a given fault:

1. `ResolveOrIgnore` – This endpoint accepts a list of fault IDs and a boolean indicating whether to ignore them. If false, all faults in the list will be marked as resolved in DCT and on the Continuous Data Engine. If true, those faults will instead be marked as ignored.

2. `ResolveAll` – This endpoint accepts an engine ID and resolves every active fault on that engine (both in DCT and on the engine itself). Permission checks are performed to confirm the active user can operate on the engine before the resolve operation is permitted.

3. `Resolve` – This endpoint accepts a single fault ID and a boolean indicating whether to ignore that fault. If false, the fault will be marked resolved in DCT and on the engine. If true, it will instead be marked as ignored.

7.5.5 Feature limitations

- DCT will only persist the most recent ten thousand (10,000) alerts/faults per registered engine and the most recent one hundred thousand (100,000) actions.
  - More specifically, DCT will remove the oldest alerts and faults from the database once they exceed 10,000 on a given Delphix engine and when the actions exceed 100,000. This was an intentional design choice for scalability reasons but might be subject change in the future.
  - If a registered engine has many pre-existing faults, alerts, or actions, it may take some time for them to fully hydrate in the DCT database (e.g. if multiples of 100,000 actions are present).
    - With many actions (hundreds of thousands or millions), full hydration of the DCT database will likely take several hours, though once completed, subsequent updates will happen every 30 seconds.
- Logins to a Continuous Data Engine by the DCT agent (e.g. to fetch a telemetry update) are not present in the **Actions** drop down list.
- If a registered engine has many pre-existing faults, alerts, or actions, it may take some time for them to fully hydrate in the DCT database (e.g. if multiples of 100,000 actions are present).
  - With many actions (hundreds of thousands or millions), full hydration of the DCT database will likely take several hours, though once completed, subsequent updates will happen every 30 seconds.
- While some targets of faults/alerts/actions are linkable in the GUI (e.g. clicking through to view the details page for those objects), this requires an explicit object type mapping in the backend code that has not yet been implemented. This means that only the following object types are linkable at present:
  - dSources
  - VDBs
  - CDBs
  - vCDBs

7.5.6 Diagnostic data

Any relevant logging specific to the new APIs will be in the `virtualization-app` container.

8 Continuous Data workflows

8.1 DevOps TDM

DCT delivers all of the Continuous Data and developer operations necessary to power DevOps and Test Data Management use-cases. This includes a suite of APIs to drive automation.

Using the above APIs, DCT can seamlessly integrate Delphix data into DevOps pipelines by providing a single point of integration for a broad Delphix deployment.
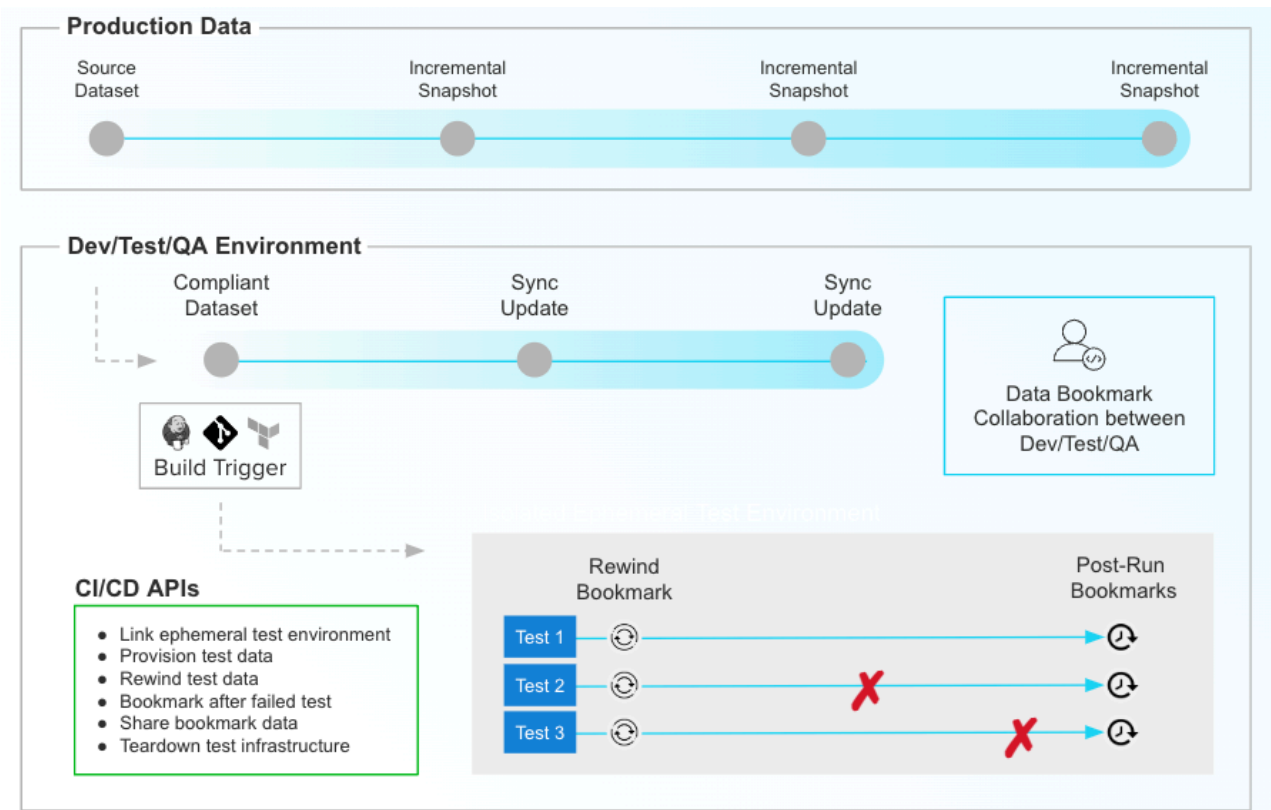
8.2 Developer experience

In addition to automation use cases, DCT provides the APIs and UI to power developer access to Delphix data and common Delphix operations. This section will detail all of the major capabilities that make up this revamped Delphix developer experience.

8.3 Self-Service vs. DCT developer experience

Data Control Tower now provides a central experience for developers. Whether a developer prefers to leverage Delphix via API, integration, or UI, DCT delivers the ability to quickly access data from any connected Delphix engine, and the common capabilities to drive application development and testing.

Previously, Delphix offered a local addon application called Self-Service (or Jet Stream) that was attached to applicable data engines. Self-Service provided an interface to access pre-provisioned datasets encapsulated in "Self-Service containers", which would be made available by admin configuration.

Data Control Tower has taken the most common operations and use-cases, and has made this experience accessible to developers via API, integration, and UI. This article will describe the key use-case and operational overlap, as well as the differences between the local engine Self-Service experience and DCT's developer experience.

8.3.1 Key similarities

1. **Developer access to Delphix Data**
   The DCT developer experience is geared toward driving access to data, with all of the same time-based operations to enable application development and testing. Operations (accessible via the API, integration, or UI) include refresh, rewind, start/stop, enable/disable, bookmark, bookmark share, and timeflow visibility/access.

2. **Developer timeflow history**
   A common UI benefit in Self-Service is the ability to visualize past timeflows (see Timeline history (see page 270) for more detail), which acts like a testing record. Every time a developer runs a test and rewinds/refreshes, that past test results are stored in Delphix as a timeflow. DCT has both API and UI instrumentation to make the visualization and curation of timeflows incredibly simple.

3. **Data-as-Code**
   Developers can use DCT bookmarks to reference a point in time on a VDB (or group of VDBs) with a developer-set retention period and human-readable name. This is valuable for development teams as they evolve application code. Whenever a code change necessitates a new database schema, a developer can bookmark a VDB that is formatted to work with that particular code branch. This empowers development teams to always have access to a viable test data set for any and code branches of an application.

8.3.2 Key differences

1. **DCT delivers a central interface powered by its converged architecture**
   This means that developers have a single location to log into in order to access and manipulate their virtual data sets.

2. **User experience**
   The DCT developer experience UI has completely been reworked to make developer access to Delphix data easy and intuitive. This experience shows itself in three UI tabs, **Active Timeline**, **Timeline History**, and **Bookmarks**, that are located in each VDB's detail menu. This experience is meant to be used by all Delphix users (admins and developers, especially) and will be tailored to the individual based on the DCT Access Control system.

3. **No template/container model**
   Previously, engine administrators needed to create templates encapsulating one or more related VDBs and provision new VDBs into a developer-accessible container. This model required manual administration that created bottlenecks for data access, which was especially prohibitive for automation use-cases. The benefit of this model was two-fold: **first**, containers represented a miniature sandbox for developers (using a Self-Service user role) and **second**, bulk operations could be performed on all container-grouped VDBs while maintaining referential synchronicity, a valuable attribute for integration testing.

4. **DCT Access Control replaces the developer sandbox enabled by Self-Service containers**
   Developers simply log into DCT and can view and act upon data that they are entitled to access with

operations tightly bounded by their defined role. DCT's Access Control system has the ability to automate both user membership of access groups and entitlement access via attribute-defined scoped roles. In addition, roles can be customized in DCT such that granular permissions can be extended and restricted down to both access group and user levels.

5. **DCT VDB Groups replace the Self-Service container grouping mechanism**
Currently only available via API, VDB groups enable the association of one or more VDBs for bulk operations while maintaining referential synchronicity.

6. **Time operations consolidation**
The developer experience UI consolidates the many time-based operations across Continuous Data and Self-Service (e.g. refresh, rewind, rollback, restore, reset, etc.) into a single operation; **refresh**. From the DCT UI, clicking refresh will take users to a contextualized screen that simplifies time operations by focusing on what timeline (and what time) the user would like to align to (parent, self, or relative).

7. **No "branching"**
Branching in Self-Service introduced the notion of task-specific timelines, each with its own associated sets of timeflows. This was a concept that was heavily tied to the "template/container" model and is obviated by the DCT Access Control system that can enable gated provisioning access to a developer. If a new timeline is needed for a separate task, you can provision a new VDB.

> DCT has a Delphix-supported integration with ServiceNow, which is commonly used as a developer resource-request tool. Users can build custom developer-centric workflows with any operation currently instrumented through the DCT API layer.

8.4 Self-Service/Jet Stream migration to DCT

Self-Service is a feature that helps manage data within environments autonomously, while Jet Stream aimed to provide more granular control over the data in those environments. These features are now utilized through Data Control Tower, adding a vast array of additional enhancements to provide full control and offer in-depth insight for your data.

8.4.1 Bookmark and permission import

A method is available to import bookmarks from a Self-Service/Jet Stream engine into DCT. The imported bookmarks from the engine are tagged with the `bookmark_source` "ENGINE", since they are initially managed by the engine. Consequently, any modifications made to these bookmarks on the engine will also be reflected in DCT.

To transition these engine-managed bookmarks into DCT-managed bookmarks, an API is provided that handles the process. It converts the engine-managed bookmarks into DCT-managed and removes both the bookmarks and their parent on the engine. When the API triggers the conversion of engine-managed

bookmarks to DCT-managed, a background job is initiated in DCT. This job retrieves all bookmarks and their respective containers/templates from Self-Service/Jet Stream.

If the job runs for a container on Self-Service, it may have assigned owners. These owners must be brought into DCT. The list of owners is available in the container and the details are retrievable from the engine. Subsequently, the job imports all valid owners (those with an email address) into DCT.

If there is not an account in DCT with the same email, a new account is created; otherwise, the creation process is skipped. In either scenario, the container-linked VDB IDs and its template data-source IDs are added to these accounts with predefined roles. Upon job completion, the accounts in DCT, whether created or updated, will have the ability to perform the same operations on bookmarks or their datasets as in self-service.

The two new roles introduced are as follows:

- `self-service-data-operator-system-role` (for adding IDs of container data sources)
- `self-service-data-viewer-system-role` (for adding IDs of template data sources)

> ℹ️ The process to import bookmarks runs automatically every four (4) hours, but can also be initiated manually.

8.5 Creating and managing bookmarks

8.5.1 Create a Bookmark

Bookmarks are a critical developer tool that enables the creation of a namable time reference to a snapshot of a VDB or VDB group. Bookmarks for single VDBs can be created from the DCT UI by selecting a VDB and expanding into its detailed view. From the **Active Timeline** view, users can select the ellipsis in the top right corner and "Create Bookmark".

Selecting the "Create Bookmark" button will open a window that enables bookmark naming, setting the custom retention period for that bookmark, and assigning any relevant tags. Creating bookmarks this way will initiate a **new** snapshot operation that will then be associated with that bookmark.

Bookmarks relating to a specific VDB can be found under the bookmarks tab in a VDBs details page. This provides a curated list of actionable snapshots that represent anything from a relevant test result to a transformed set of schema that can be associated with a specific branch of code.
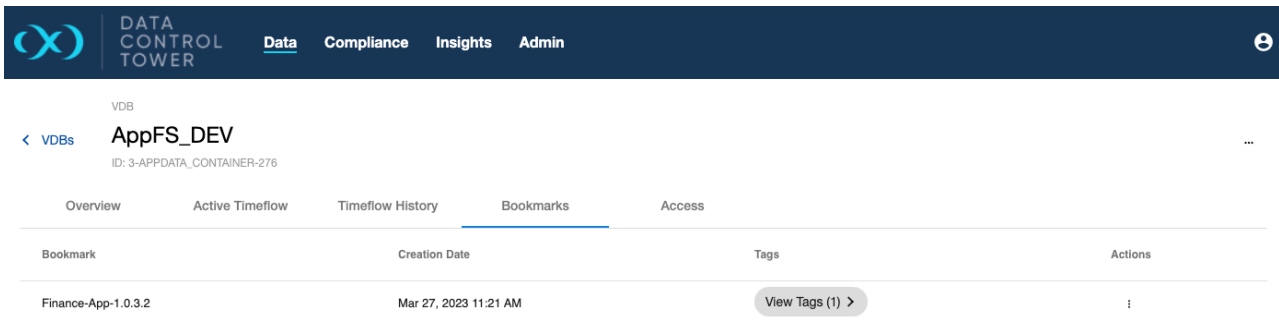
8.5.1.1 Create a bookmark from an existing snapshot

Starting in version 7.0, the UI has an option to create bookmarks from existing snapshots.

On the VDB detail page, under the **Active Timeline** tab for each snapshot, a **Create Bookmark** action is available. This opens a dialogue that shows a list of inputs for the user to select from, to create a bookmark.

Once the user clicks **Create Bookmark** in the dialogue, the bookmark will be created for that particular snapshot (if all the mandatory fields are completed), else errors will be shown.
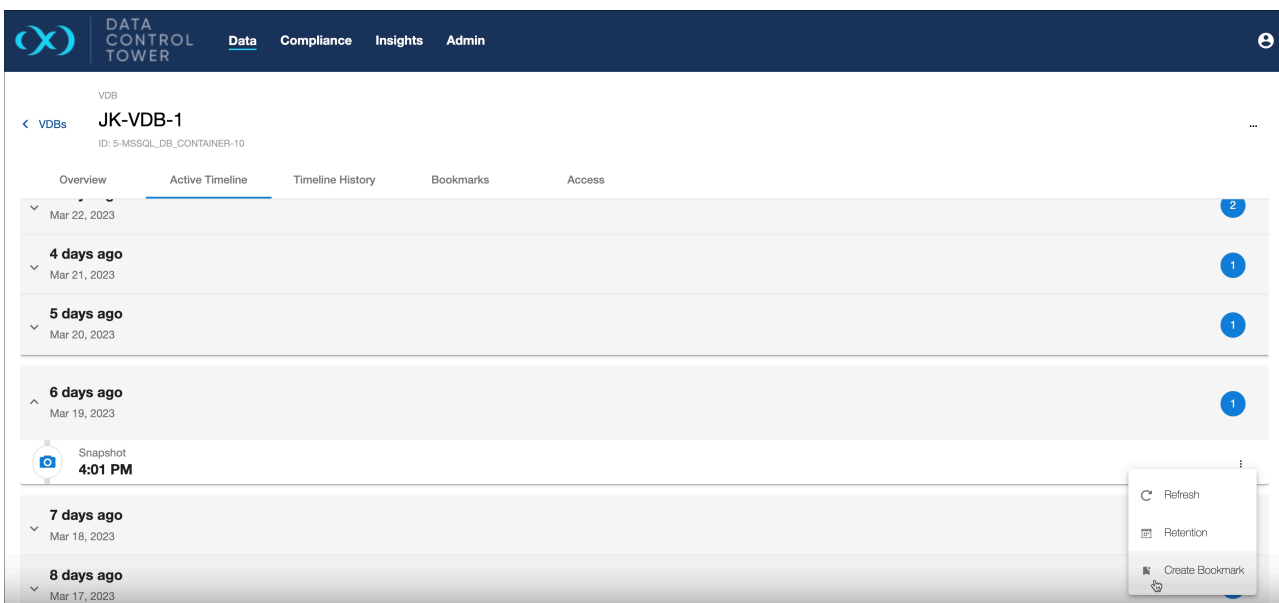
Create a bookmark at the **current** data and time.
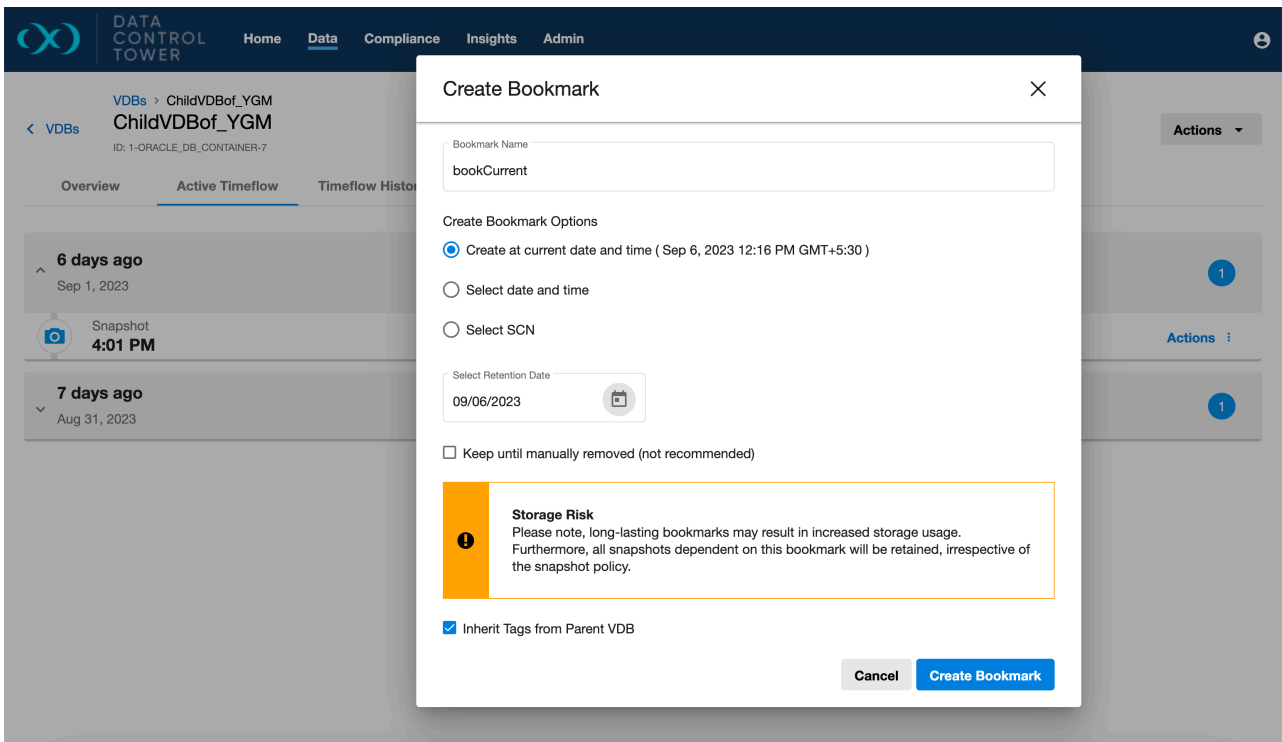
Create a bookmark based on the **selected** date and time.
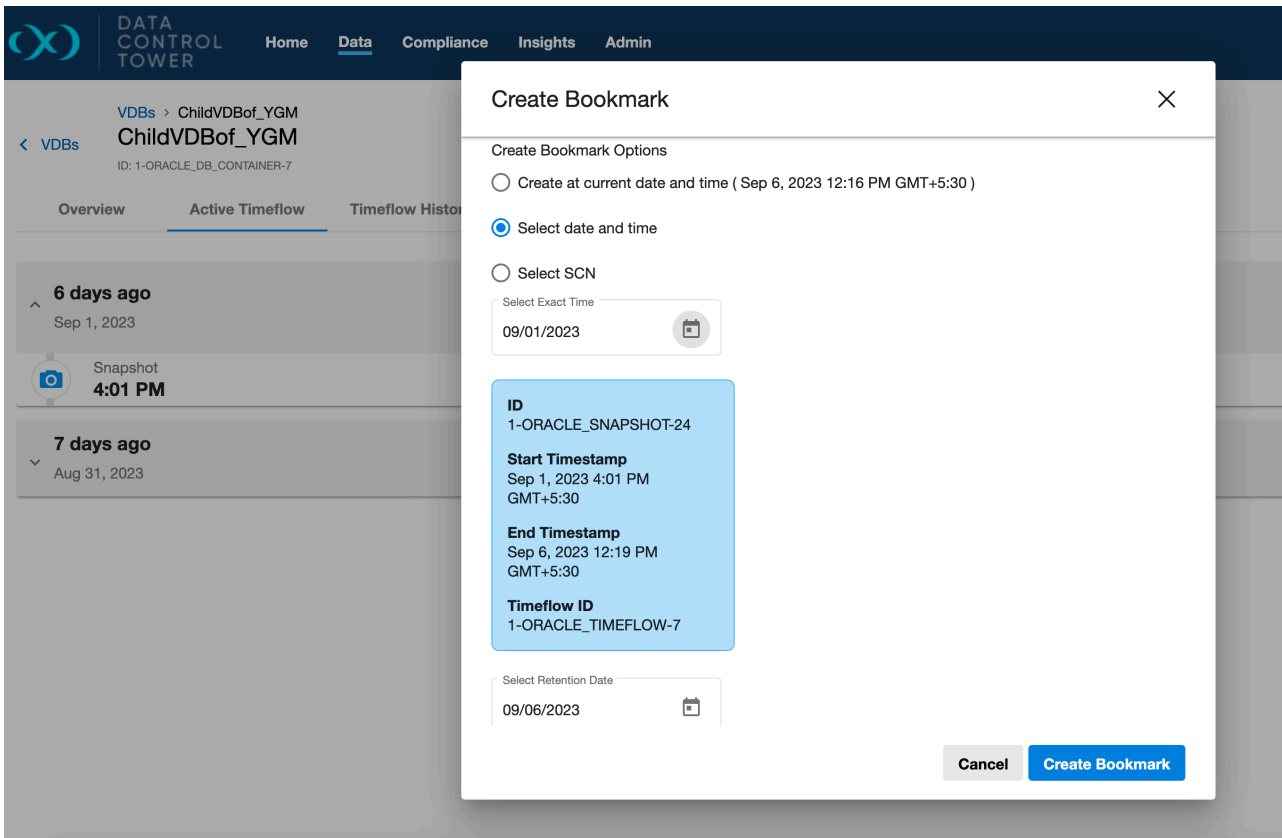
Create a bookmark based on **SCN Number**.

8.5.2 Bookmark API Documentation

Some advanced bookmark operations are only available via API at present, formal documentation can be found via DCT's swagger docs or the Developer resources section. This portion of the bookmarks documentation will discuss examples of advanced use-cases.

8.5.3 Create a Bookmark at the current time for multiple VDBs

DCT (version 6.0 and above) can create bookmarks from existing snapshots. This is particularly useful for users looking to migrate Self-Service bookmarks to DCT or any developer looking to retroactively create a bookmark reference.

In cases such as integration testing, bundling multiple VDBs together to represent a complete set of data that a complex application would run on is helpful. This API example shows how a single bookmark reference can be created off of multiple VDBs, to provide a provision point for new testing sets or the creation of a VDB Group that can be used to maintain referential synchronicity from that bookmark point.

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/bookmarks' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
```

```
    "name": "MyBookmark1",
    "vdb_ids": [
        "1-ORACLE_DB_CONTAINER-2",
        "2-ORACLE_DB_CONTAINER-2"
    ]
}'
```

> These API calls will return a DCT job to track the creation process. This job ID can then be used to poll the status via the jobs API. Example response:

```
{
    "bookmark": {
        "id": "9e8c7223f1af4694a19ac2c2f7696eda",
        "name": "MyBookmark1",
        "creation_date": "2023-03-27T20:56:13.916857Z",
        "vdb_ids": [
            "1-ORACLE_DB_CONTAINER-2",
            "2-ORACLE_DB_CONTAINER-2"
        ],
        "retention": 30,
        "expiration": "2023-04-26"
    },
    "job": {
        "id": "8fe825f5635d45299915c3cb88a17623",
        "status": "PENDING",
        "type": "BOOKMARK_CREATE",
        "target_id": "9e8c7223f1af4694a19ac2c2f7696eda",
        "start_time": "2023-03-27T20:56:14.363549Z"
    }
}
```

8.5.4 Creating a bookmark from a chosen timepoint

Now bookmarks can be created at a chosen snapshot timepoint using new properties in the request payload. Some properties are mutually exclusive, so be sure to use a valid request payload.

Here are some examples of a valid request payload:

```
{
    "name": "my-bookmark-123",
    "vdb_ids": [
        "vdb-123"
    ],
    "location": "112233",
    "expiration": "2021-07-04",
    "retain_forever": false,
```

```
    "tags": [
      {
        "key": "key-1",
        "value": "value-1"
      },
      {
        "key": "key-2",
        "value": "value-2"
      }
    ],
    "make_current_account_owner": true
}
```

```
{
  "name": "my-bookmark-123",
  "timeflow_ids": [
    "timeflow-1"
  ],
  "location": "112233",
  "expiration": "2021-07-04",
  "retain_forever": false,
  "tags": [
    {
      "key": "key-1",
      "value": "value-1"
    },
    {
      "key": "key-2",
      "value": "value-2"
    }
  ],
  "make_current_account_owner": true
}
```

```
{
  "name": "my-bookmark-123",
  "timeflow_ids": [
    "timeflow-1"
  ],
  "timestamp": "2021-05-01T08:51:34.148000+00:00",
  "expiration": "2021-07-04",
  "retain_forever": false,
  "tags": [
    {
      "key": "key-1",
      "value": "value-1"
    },
    {
      "key": "key-2",
      "value": "value-2"
```

```
        }
    ],
    "make_current_account_owner": true
}
```

Properties

- `timestamp`
- `location`
- `timestamp_in_database_timezone`
- `timeflow_ids`

Rules

- `timestamp` , `location` and `timestamp_in_database_timezone` are mutually exclusive.
- If any of the properties from `timestamp` , `location` and `timestamp_in_database_timezone` are provided then:
  - At least one property for `vdb_ids` or `timeflow_ids` must be provided.
  - The `snapshot_ids` property is not allowed to be set.

8.6 VDB operations

> The developer experience will continue to see investment and additional capabilities over the next few releases.

The VDB operations UI serves as an actionable command center for admins and developers. With this UI, users can migrate from using the local engine UI to leveraging DCT to do their daily VDB-related work. This encompasses both continuous data as well as any developers leveraging Delphix Self Service.

The core benefit of this UI experience is the breadth of access coupled with DCT's access control system. Using both together, a user can access and act upon any data on any connected engine within the boundaries of the entitlement and permissions set by the admin-driven access control system.

To access the VDB operations UI, users only need to log into DCT and select the detailed view of any particular VDB.

> Users will only be able to see VDBs if they have been granted access via the Access Control system.
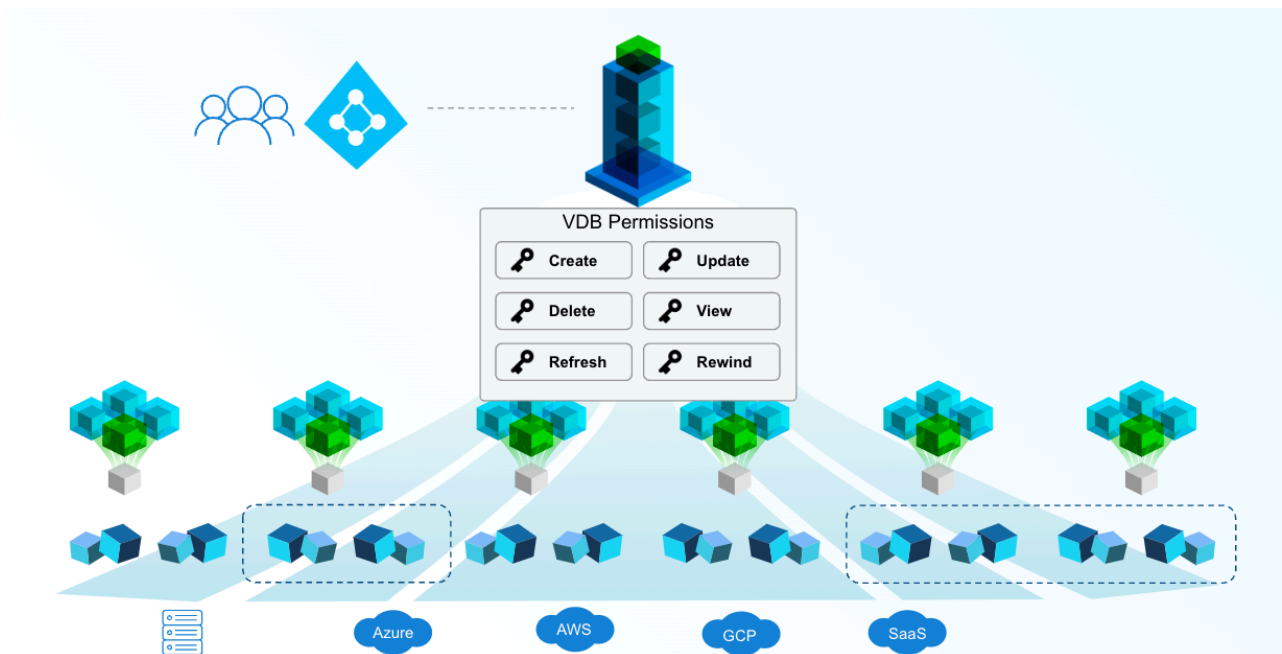
From there, users can perform common operations such as refresh, rewind, and bookmark using the Active Timeline tab. Developers have additional functionality with the Timeline History tab that exposes non-active timelines (also known as timeflows).

8.6.1 VDB provisioning wizard

Users can provision VDBs from the DCT UI using the Provision VDB button located under the action button on a VDB details page.

8.6.1.1 Supported database platforms

- **Oracle Multi Instance (Single Tenant)**
- **Oracle Multi Instance (Multi Tenant) with linked vCDB**
  - TDE is supported for Oracle versions greater than or equal to 12.2.
  - Only vCDBs with database versions greater than or equal to 12.1.0.2 can be linked.
  - Auto VDB restart is supported for Oracle versions greater than or equal to 12.1.0.2.
- **Oracle Single Instance (Multi Tenant) with linked vCDB**
  - TDE is supported for Oracle versions greater than or equal to 12.2.
  - Only vCDBs with database versions greater than or equal to 12.1.0.2 can be linked.
  - Auto VDB restart is supported for Oracle versions greater than or equal to 12.1.0.2.
- **MSSQL Multi Instance**
  - Drive letters are not supported and are set to default values.
- **SAP ASE**
- **AppData**
  - AppData is used for connecting with other databases like Postgres, SAP HANA, etc. by uploading the plugin. Refer to the Provisioning a PostgreSQL VDB[118] page for more details.
- **vFiles**
  - Refer to the Unstructured files and app data[119] page for more details.

Extending the developer experience capabilities in DCT, users can now provision MSSQL single instance databases or Oracle single instance multi-tenant databases with linked CDB data platforms from the user interface, using an intuitive wizard workflow. Located on the VDB page is a **Provision VDB** button that opens the provisioning wizard. (Note: non admin users will only be able to see provisionable sources (dSources and/or VDBs), environments, and engines to which they are authorized to see and act upon).

8.6.1.2 Step examples

The provisioning wizard will walk through the following steps:

- **Source**: search and select either a dSource or VDB to provision from.

---

118 https://ecosystem.delphix.com/docs/latest/provisioning-virtual-databases
119 https://cd.delphix.com/docs/latest/unstructured-files-and-app-data

- **Provision Point**: three options for a provision point, similar to a refresh point.
    - A selected snapshot
    - A specific timestamp (closest snapshot to the timestamp)
    - A location ID/number (closest snapshot to the location number)
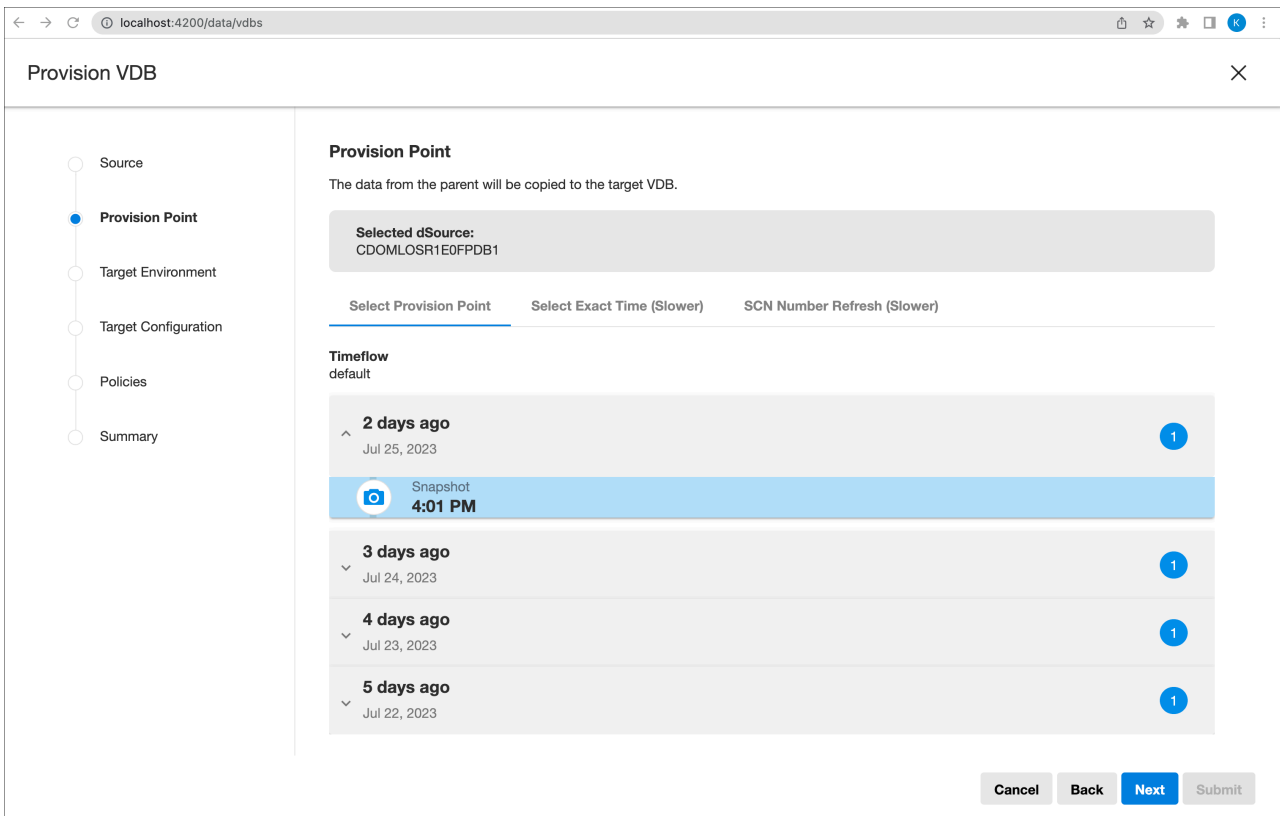
- **Target Environment**: shows compatible environments with compatible repositories, and can optionally provide privileged credentials.
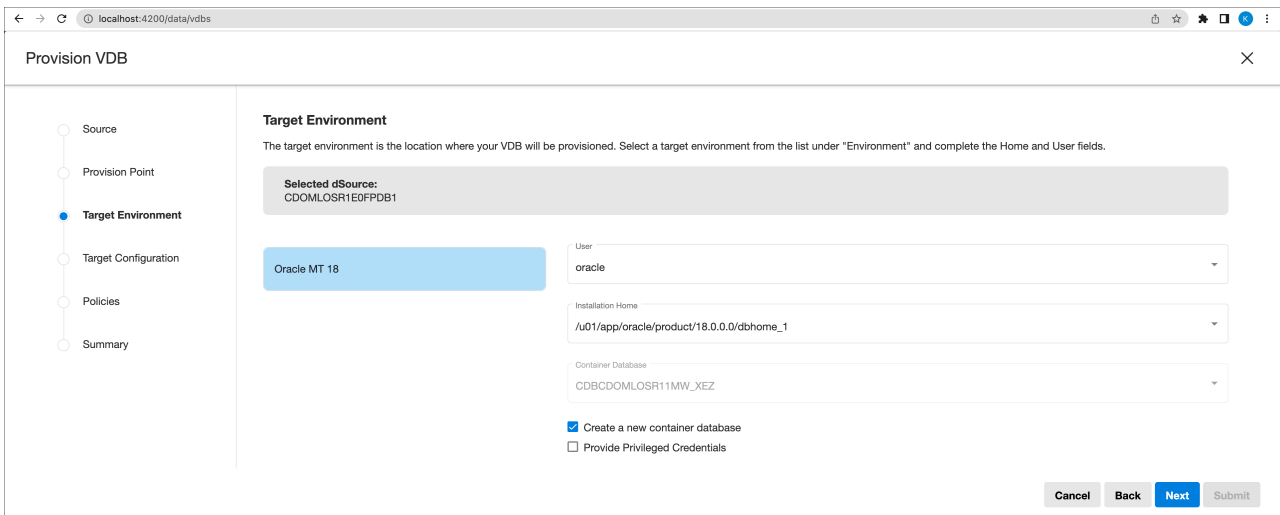
- **Target Configuration**: should be prefilled with default configurations. One thing to note here are the tags which are additive when "Include Tags from Parent" is checked, and you wouldn't immediately see the tags from the parent in the editor. Tags are added when the **Include Tags from Parent** box is checked, you would not immediately see them from the parent in the editor.

  - Users can also select the engine group and register listeners by expanding the advanced section below. In DCT, it is recommended to use tags instead of engine groups.

- The Target Configuration screen will present different options depending on the chosen **Source**.

- **vCDB Configure Parameters**: Applicable to Oracle Single Instance (Multi Tenant) with linked vCDB.

- **Policies**: choose a snapshot policy.
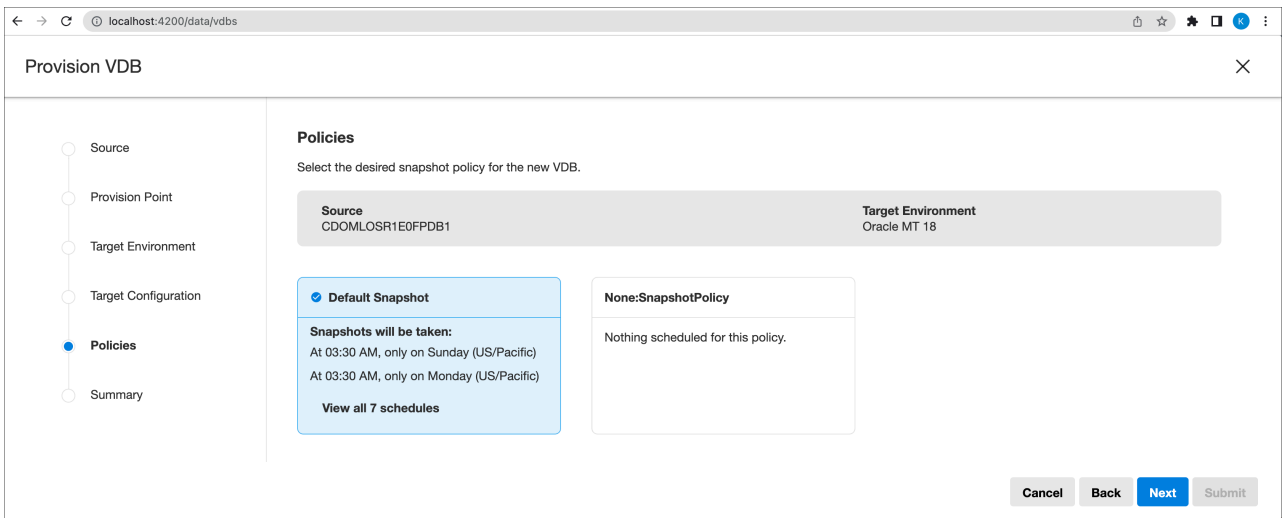
- **Masking**:
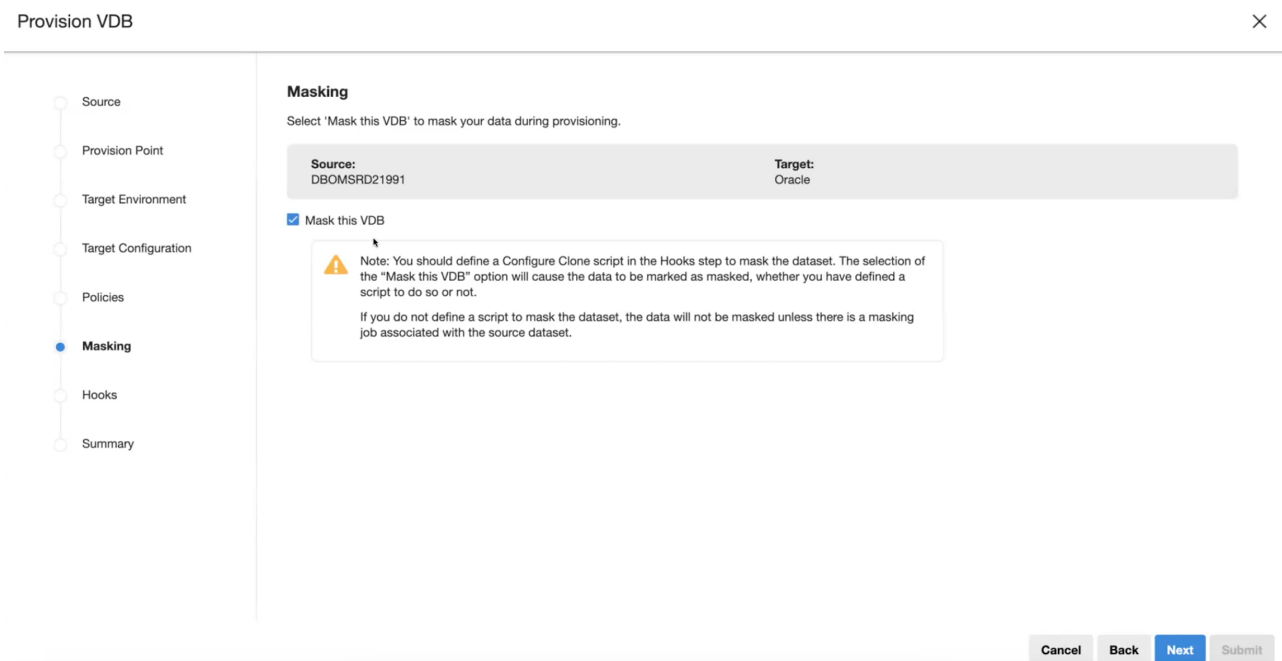
- **Hooks**: Delphix users now have the ability to add custom hooks to a VDB at the time of provision (as part of the provision wizard UI) and update them under a VDB detail page, so that they will execute during refresh operations. A Hooks section in the Summary page shows a list of all the hooks configured (if any). Hooks can be modified in the details page after creation.
    - In DCT 18.0.0 and above, users can reorder hooks for different operations at various hook points from within the VDB provisioning wizard UI. In addition, users can now add, edit, delete, and reorder hooks on the provisioned VDB via the VDB details page under the Hooks tab. Either cancel or confirm the reordering of the hooks using the corresponding buttons in the UI.
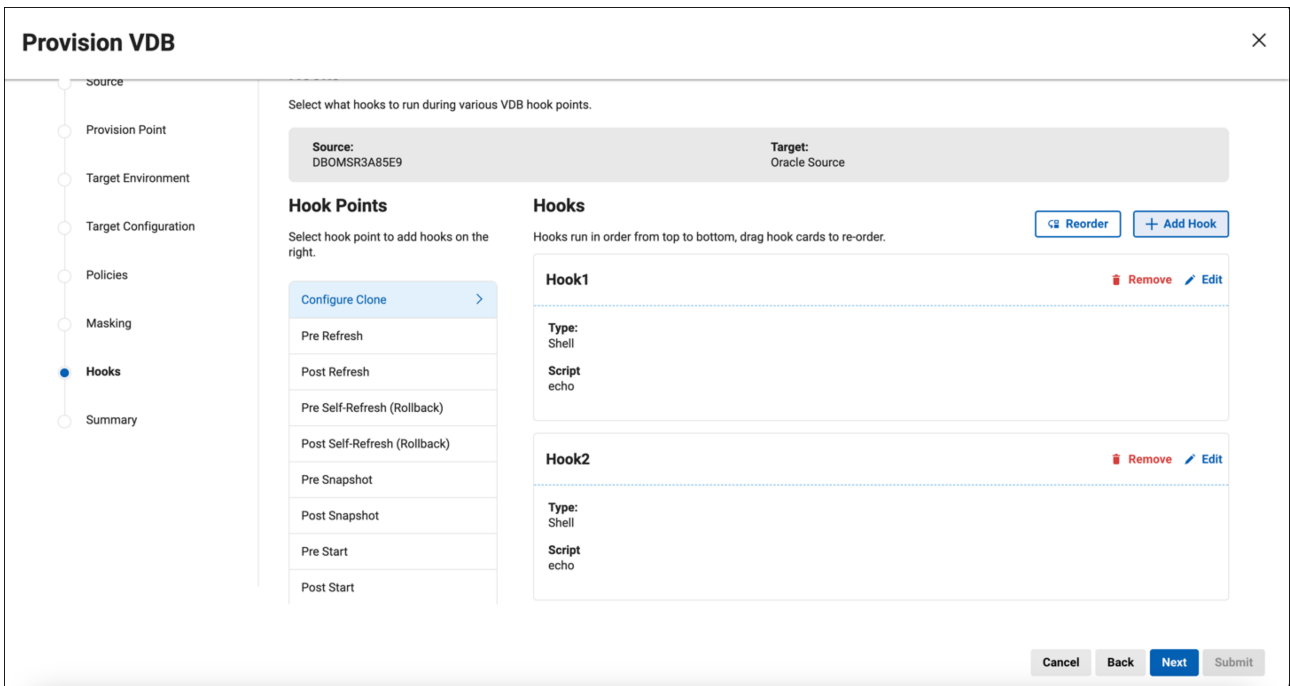
- **Summary**: review the selections that have been made. In 10.0 and above, a hooks header has been added to the Summary page that shows a list of all the hooks configured for Hook Operations while provisioning a VDB (if this was configured).
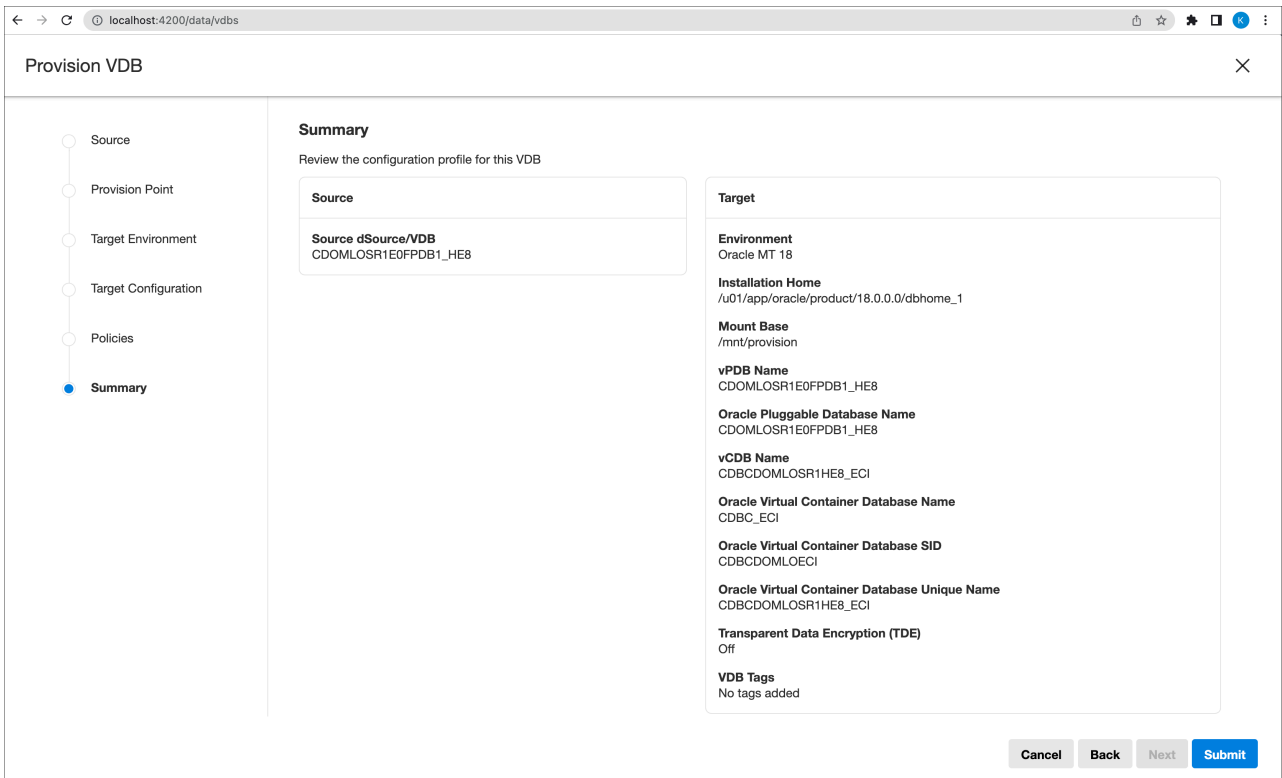
8.6.1.3 Limitations

TDE and Auto VDB restart are only supported for Oracle version 12.2 or higher.

8.6.2 VDB refresh wizard

8.6.2.1 Overview

The VDB refresh wizard in the Data Control Tower (DCT) UI offers important engine refresh operations including:

- **Self-refresh by snapshot, timestamp, or location:** Refreshes a VDB back to a point in its own history.
- **Refresh to parent by snapshot, timestamp, or location:** Data is pulled from the VDB provision parent (the dSource or VDB from which the VDB was provisioned).
- **Refresh to relative by snapshot, timestamp, or location:** Allows selection of data from either the origin dSource of the VDB or any VDB sharing the same origin dSource.
- **Refresh to bookmark snapshot:** Refreshes a VDB to a compatible bookmarked snapshot.

To perform a refresh from a dSource or VDB, the account must have the REFRESH permission on both the VDB being refreshed and the dSource (or VDB) from which the data is being refreshed.

8.6.2.1.1 New changes in DCT version 19.0.0

8.6.2.1.1.1 Timeline selection in VDB refresh wizard

The VDB refresh wizard now shows the timeline to select the snapshot to refresh from. This timeline presents various refresh sources including Parent, Self, and Relative.

- **Bookmarks availability:** Bookmarks are now available in all mentioned refresh sources in the refresh wizard.
- **Action button for bookmarks:** The action button for the bookmarks will not be shown in the VDB refresh wizard on the timeline.

8.6.2.2 User interface

8.6.2.2.1 Initiating Self-Refresh by Snapshot

The Self-Refresh by Snapshot operation can be initiated as an action from the VDB's own active Timeline or Timeline History view using the action menu for snapshots.

8.6.2.2.2 Performing all refresh types

All three refresh types (Self, Parent, Relative) can be performed by opening the refresh wizard on the VDB Details page. The steps are as follows:

1. **Select Refresh Source:**
   - **Self-Refresh:** Select a timeflow from the VDB's own history, then a snapshot within the selected timeflow.
   - **Parent Refresh:** Select a timeflow from the VDB's provision parent's history, then a snapshot within the selected timeflow.
   - **Relative Refresh:** Select a relative dSource or VDB, then a timeflow, then a snapshot.
   - **Refresh to Bookmark**: Select a compatible bookmark from the VDB.

> ℹ️ You can also refresh from a particular bookmark under the VDB details page, via the **Bookmarks** tab. Choose a bookmark from the list as the one to refresh from, then click the Actions menu and select Refresh to Bookmark. Source and Expiration Date columns are now populated in the Bookmarks list.

2. **Review and Submit:**
   - Review the Summary page once configurations are complete, then click Submit. Refreshing happens asynchronously and takes various amounts of time. The DCT UI does not currently show asynchronous job progress or errors; please refer to the engine UI for this function.

8.6.3 VDB Groups

8.6.3.1 Overview

VDB Groups introduce a new functionality that allows users to organize and manage VDBs in a comprehensive listing page, accessible under the **Data** tab in DCT version 20.0.0 and above.

8.6.3.1.1 New in DCT 21.0.0

- **VDB Group configuration management (add/remove VDBs and manage the order of execution)**
  DCT can add/remove VDBs to/from the group and arrange the order from which bulk operations (like refresh) will be applied. This is particularly crucial for data sources like Oracle EBS, where application files and the underlying Oracle database need to be refreshed in a specific order.
- **VDB Group bookmark creation support**
  DCT can create a bookmark across one or more VDBs contained in the group, to provide a human–readable reference point for that specific point in time.
- **VDB Group refresh support**
  DCT can refresh one or more VDBs contained in the group based on a specific bookmark reference or to the latest snapshot on each VDBs' respective parent (VDB or dSource).

New in DCT 22.0.0

- **VDB Group start/stop, disable/enable and lock/unlock support**
  With these features, users can perform key operations on VDB groups, such as starting, stopping, enabling, disabling, locking, and unlocking VDBs. The functionality allows for batch processing, where multiple VDBs within a group can be managed simultaneously, ensuring smooth and efficient operations.

8.6.3.2 Creating a VDB Group

The creation of a VDB Group is facilitated through a three-step wizard:

1. **Configuration**
   - Add identifying details for the VDB group, including **Name** and **Tags**.

2. **Provision Point**

- Select one or multiple VDBs to include in the group based on user needs.

3. **Operation Order for VDBs**

- Add the order for the selected VDBs to ensure the operations are run sequentially for each of them.

4. **Summary**

• Review the configurations and click **Save VDB Group** to finalize the creation process.

8.6.3.3 Viewing and managing VDB Groups

Once created, VDB Groups are displayed in a list with several columns of information: **Name, Status, Platform, Last Refresh, Tags**

Users can select the **View** button from the group list to open the details page of the selected VDB Group.
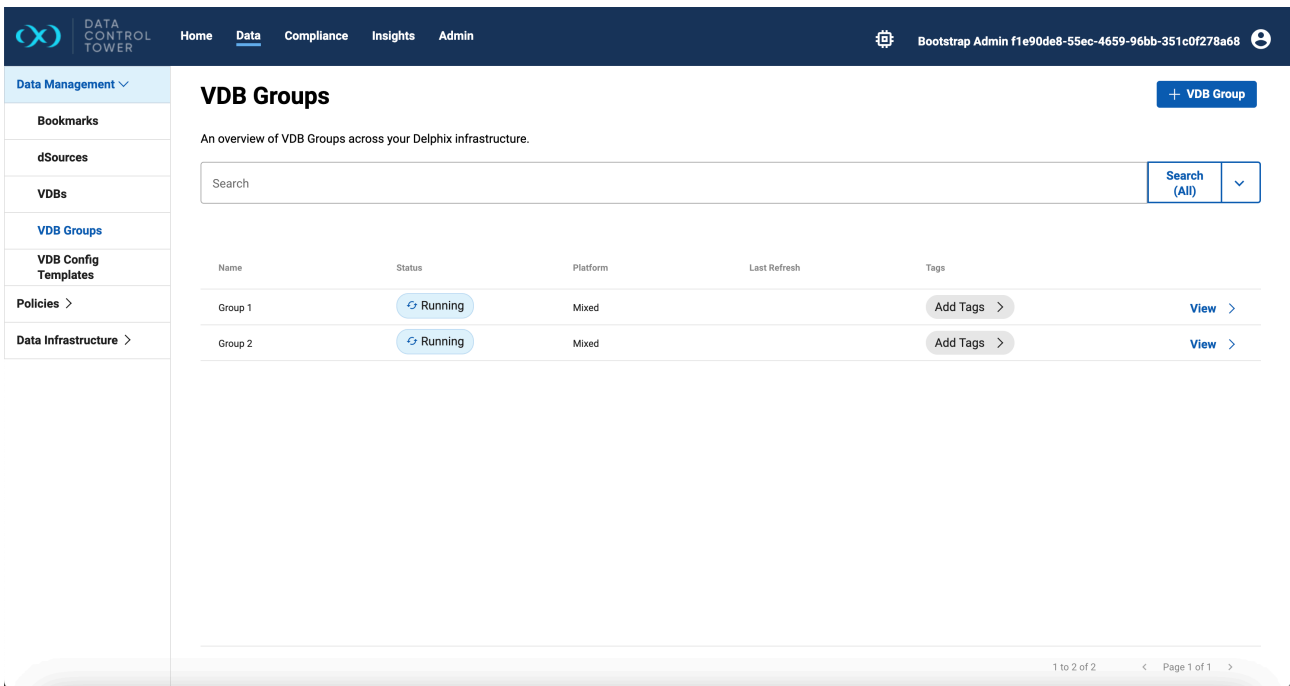
8.6.3.3.1 VDB Group detail page

The detail page for each VDB Group includes two tabs:

1. **Overview tab**

   - **Status:** Displays the status of each VDB within the group.

   - **Number of VDB sets:** Indicates the number of VDB sets.

   - **Details:** Includes information such as Last Refresh and Tags.
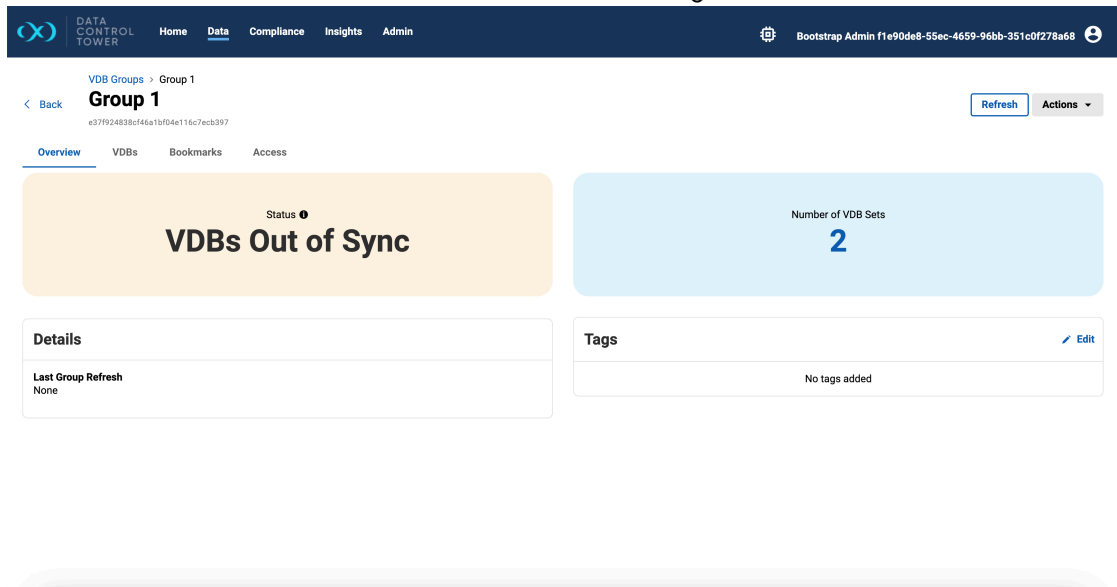
2. **VDBs Tab**

   • Displays the VDBs that are part of the VDB Group, with the following columns of information: **Status, Name, Creation Date, Last Refresh, Platform, Engine, Compliance, Tags**

3. **Bookmarks Tab**

   • Displays the compatible bookmarks that are part of the VDB Group, with following columns of information: **Bookmark Name, Source VDB Group, Creation Date, Expiration Date, Data Timestamp and Tags**

4. **Access Tab**

- Displays the access, with the following columns of information: **User, Access Groups, Permissions**

The **Actions** menu in the top corner of this page has the following options:

- **Delete** - Delete the VDB Group.
- **Create Bookmark** - Creates a bookmark on the VDB Group.
- **Start** - Starts all VDBs within the VDB Group.
- **Stop** - Stops all VDBs within the VDB Group.
- **Enable** - Enables all VDBs within the VDB Group.
- **Disable** - Disables all VDBs within the VDB Group.
- **Lock** - Locks all VDBs within the VDB Group.
- **Unlock** - Unlocks all VDBs within the VDB Group.

8.6.4 Timelines in DCT

8.6.4.1 Introduction

A Timeline in Data Control Tower (DCT) helps to visualize and manage the progression and state of Virtual Databases (VDBs) over time; in essence, a chronological sequence of events in the life cycle of a VDB. The UI implementations of this feature help users trace, access, and manage the various stages of these data environments.

Each event within a timeline could be the creation of a new VDB, refreshing data from its source, or any operational activity that alters the VDB's state. These events are recorded with timestamps to provide a historical record of all activities and changes.

Noteworthy changes in the latest version of DCT include terminology updates, removal of redundant tabs, improved sorting functionalities, and the introduction of new search capabilities.

8.6.4.2 Prerequisites

To utilize Timelines, users must have a running DCT Engine.

8.6.4.3 Terms and UI adjustments

- The term **Timeflow** has been updated to **Timeline** throughout the application, aligning terminology more closely with its functionality.
- The **Active Timeflow** tab has been removed from the VDB Details page. This change eliminates duplication, as the Timeline History tab already present on the same page serves a similar purpose.
- Previously, dSource groupings based on time were featured in the Timeline History view. This grouping has been removed to simplify the timeline presentation.

8.6.4.4 Sorting capabilities

A new **Sort By** dropdown menu enhances how timelines can be organized, by allowing users to sort timelines using with following options:

- **Timeline Activation** [default]: Timelines are ordered by their activation timestamp, ensuring the active timeline is displayed prominently at the top.
- **Timeline Creation**: Sorts timelines based on their creation timestamp.
- **Parent Timeline Location**: Organizes timelines by the timestamp of the parent timeline from which the data was refreshed.
- **Origin Timeline Location**: Sorts timelines by the timestamp indicating where the original data (or dSource) originates.

8.6.4.5 Timeline metadata and search functionality

- The active timeline is now always expanded to show the current VDB's state and recent activities, making it easier to understand the current context of the VDB.
- Timeline metadata now reflects the selected sort mode, displaying relevant timestamps in a clear and understandable format, such as "Timeline Activation: \<time\>", improving the ability to track timeline details.
- A search feature has been added to the Timeline History, allowing for searches based on the top-line metadata of the timeline. This feature improves the efficiency of locating specific timelines or activities within the application.

8.6.4.6 Feature limitations

Please take note of these limitations regarding timeline metadata:

- **Connectivity issues**: If the engine connectivity is lost, DCT cannot determine whether a timeline is active or its activation time, resulting in "not available" messages.

- **Renamed timelines**: DCT infers timeline creation time from its default name. If a timeline is renamed, DCT cannot determine its creation time, leading to "not available" messages.
- **Engine timezone changes**: Creation times are interpreted in the engine's timezone. If the engine's timezone changes, creation times may not accurately reflect the timeline's true creation time.

8.6.4.7 Timeline History

Timeline History can be found by selecting **Details** for any VDB located on the **VDB List** page and then selecting the **Timeline History** tab. This view serves as a developer-centric console that shows the complete history of a VDB, including non-active timelines, which contain information like past test results. It allows developers to curate and access the complete chronology of their testing efforts with operations like renaming individual timeflows, make active, refresh to snapshot, and create bookmark to drive organization and access.

8.6.4.7.1 Timeline History UI

The Timeline History UI is comprised of a vertical timeline-based representation of all actionable points of interest (snapshots) for that VDB. The snapshot list is chronologically grouped over blocks of time to easily identify relevant snapshots to act upon. Once the right snapshot is found, users can access a contextualized action menu by selecting the corresponding ellipsis to the relevant snapshot.

A row at the bottom of each timeline in the Timeline History tab of a VDB shows information about the action that created the timeline and also contains a refresh button, which can directly refresh the VDB back to the Bookmark/Snapshot that was used to originally create the timeline.

DCT has introduced some key fields to enhance data traceability and context understanding.

- **Origin Source**: Identifies the top-most parent (dSource) from which the data originated, and clicking on this field redirects users to the origin source's Detail Page for comprehensive insights.
- **Origin Timeline Location**: Specifies the exact moment in time on the origin source timeline where this data was derived.
- **Parent Source**: Denotes the immediate parent source of the data, with a click-through feature leading to the parent source Detail Page.
- **Parent Timeline Location**: Pinpoints the precise moment in time on the direct parent's timeline from which the data originated, further solidifying the data's lineage and accessibility.

8.6.4.7.2 Non-active timelines

Non-active timelines (previously, "timeflows") are a critical aspect to data virtualization, in addition to being an important tool for developers. Whenever a time-based operation takes place, the previous timeline (and associated data) becomes non-active, and a fresh timeline takes its place.
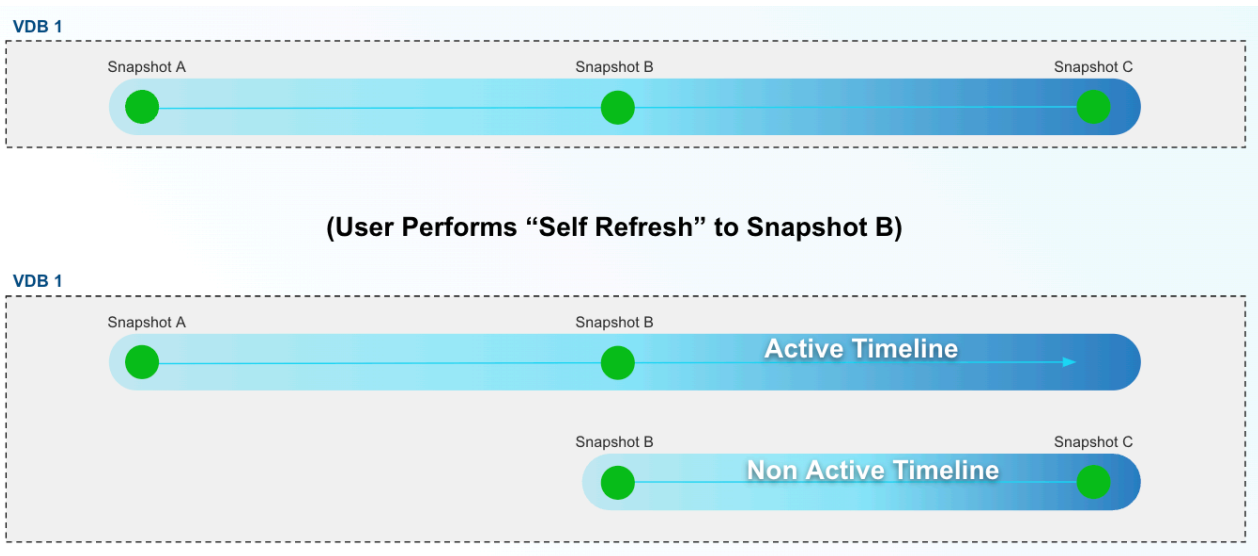
> **ⓘ Non-active timeline creation from self-refresh**
> In this example, a developer has performed a self-refresh (previously, "rewind" or "rollback"). In refreshing Snapshot B, the developer has created a new active timeline that represents a clean slate starting with the data state of Snapshot B. If the developer wants to refer back to the past results, represented on the non-active timeline, they can activate the past timeline by hitting the Make Active button in the DCT UI, or by refreshing to a point in time by referencing the non-active timeline's ID.

For developers, having the ability to catalog (name and tag) and reference past timelines is a critical aspect of application development such as performing ad hoc code validation or manual testing. The Timeline History UI in DCT provides a home for single VDB visibility of all accessible timelines (note: timeline availability is controlled through capacity management and snapshot retention policies).

8.6.4.7.3 Time concepts within the Timeline History tab

DCT's UI highlights different notions of time within Delphix (e.g. dSource, VDB, and VDB lineages) and how they relate to one another. These time-based relationships are exposed in the Timeline History view to drive accurate testing for developers.

The "Based on dSource time" designation helps to drive awareness of the relationship between a VDB and its dSource provision point (in most cases this will equate to the production database's state at that time), which is helpful for use cases such as capturing data from a meaningful event. As a VDB refreshes to newer snapshots on a dSource, those changes of data state are grouped separately on the Timeline History view as they represent completely different data.

8.6.4.7.4 Timeline operations

To reference another timeline, like one from the past, select it and use the **Make Active** button. This would make the timeline active, so that the data therein can be accessed.
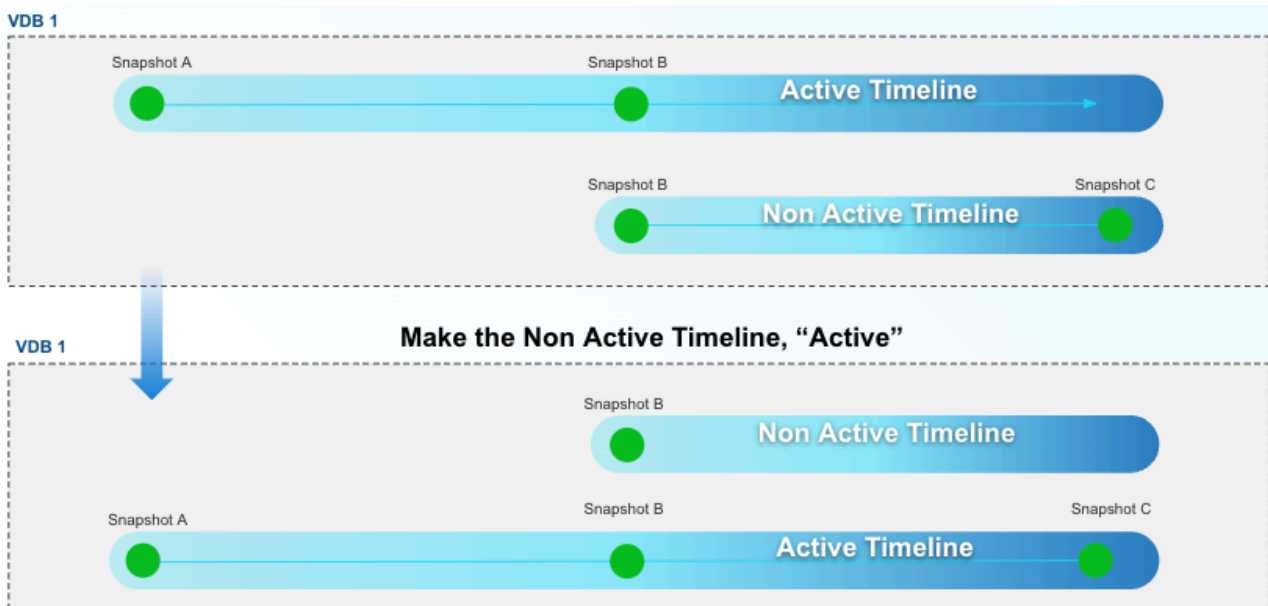
8.6.4.7.4.1 Refresh to snapshot

Included within each timeflow are snapshots that can be accessed via a dropdown menu under each timeflow. If given the right permission, a developer can refresh the VDB to that snapshot. While this may seem similar to Make Active, there are underlying differences; a refresh will reprovision the VDB based on that point in time, whereas, Make Active simply changes the reference to different blocks of the underlying storage.
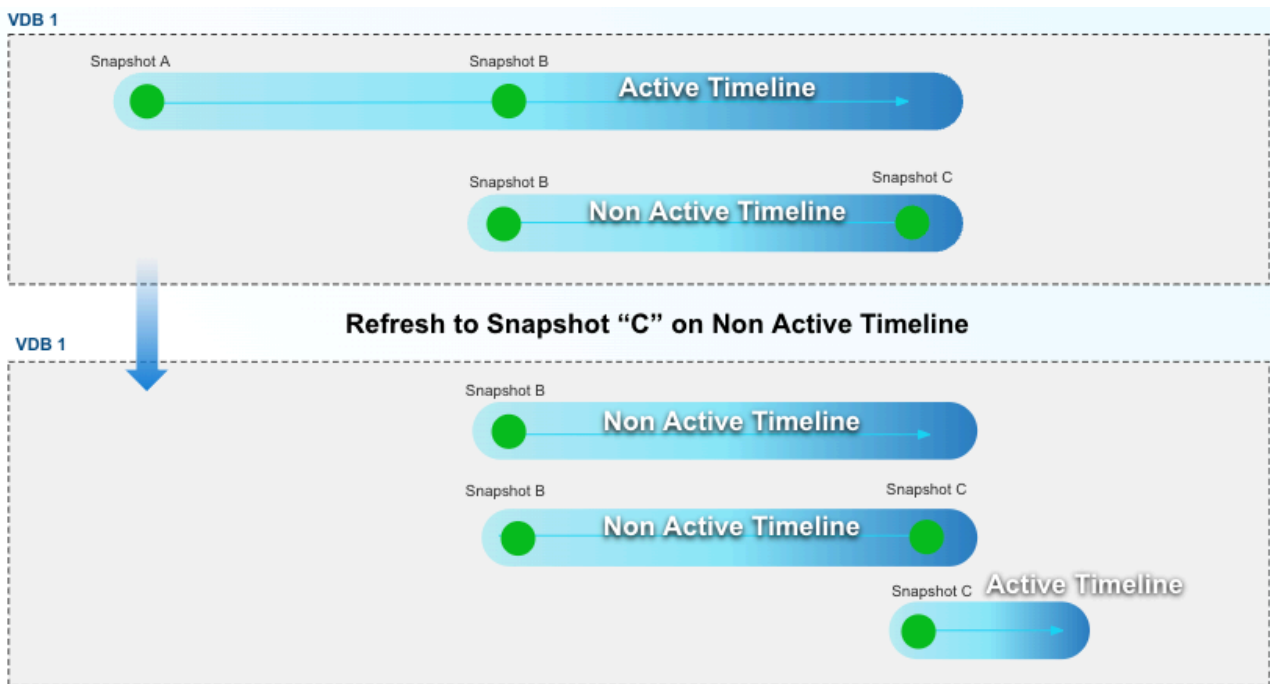
8.6.4.7.4.2 Create bookmark

A snapshot allows developers to convert it into a DCT Bookmark (see page 204), which enables them to assign a name and special retention policy to that time reference. This also enables stronger collaboration between developers, as bookmark references can be used for a sibling refresh operation.

8.6.4.7.4.3 Rename timeline

Curate timelines to correspond with tests by renaming them to easily reference the resulted data. This action can be performed by clicking the ellipsis on the timeline to reach a drop-down menu. This menu includes the option to rename which will open a dialog box to input a new name. The input is prefilled with the current timeline name by default. The changes may take a short time to be reflected in the UI.
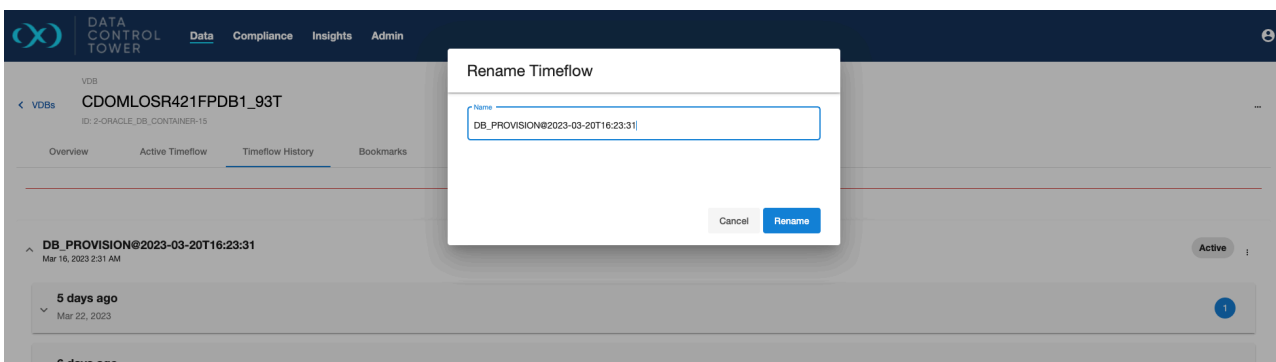
8.6.4.7.4.4 Delete timeline

Delete timelines via the UI by selecting the ellipsis next to the corresponding timeline and selecting delete. When clicked, this opens a confirmation dialog. Once confirmed, a job is kicked off to delete the timeline. This menu item is disabled for the currently-active timeflow.
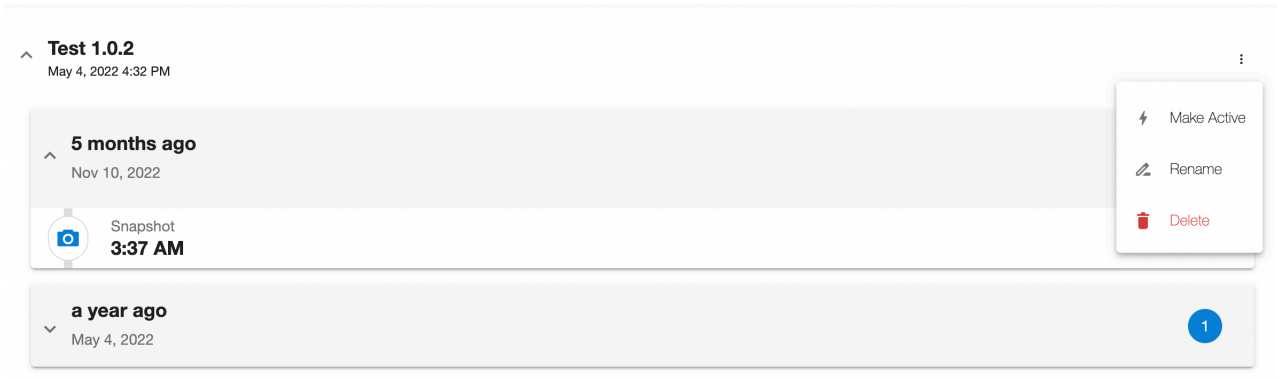
8.6.4.7.5 API documentation

While the DCT UI provides a visual aspect to timeflow chronology and grouping, the DCT API has all of the necessary instrumentation to integrate these concepts into automation. The formal documentation can be found via DCT's Swagger docs or the **Developer resources** documentation section. This portion of the timeline documentation will provide some general examples:

8.6.4.7.5.1 Get a list of a VDB's timelines

This simple call will enable the listing of the entire timeflow roster underneath a VDB enabling a developer to take inventory and action.

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/timeflows/search?limit=50&sort=id' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "filter_expression": "dataset_id eq '\''1-ORACLE_DB_CONTAINER-2'\''"
}'
```

8.6.4.7.5.2 **The UPDATE timeflow API can be used to change a timeflow's name**

Once the appropriate timeflow is found, developers can rename the timeflow to align with a naming scheme or other relevant designations such as a compatible code branch.

```
curl -X 'PATCH' \
  'https://<APPLIANCE_ADDRESS>/v3/timeflows/1-ORACLE_TIMEFLOW-7' \
  -H 'accept: application/json' \
```

```
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Latest on refresh state"
}'
```

8.6.4.7.5.3 List all the snapshots for a timeflow

Developers can then use the snapshot API to find all relevant "actionable" points in time on the identified timeflow. The below example shows how to only expose snapshots on a particular timeflow. From there, developers can feed the relevant snapshot ID into refresh or provision endpoints or create a bookmark reference from that snapshot.

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/snapshots/search?limit=50&sort=id' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "filter_expression": "timeflow_id eq '\''1-ORACLE_TIMEFLOW-7'\''"
}'
```

8.6.5 Locking and unlocking a VDB

This feature allows users with **LOCK** and **UNLOCK** permission to lock and unlock a VDB, similar to the lock/unlock feature available via Self-Service containers on the Continuous Data Engine.

The VDB can be locked by a user with **LOCK** permissions for that particular VDB. Once a VDB is locked, all of the VDB operations (Refresh, Start, Stop, Delete, and Create Bookmark) are disabled for all other users.

The user needs the **UNLOCK** permission on the VDB to perform an unlock operation. Thus, only the user that locked the VDB **OR** a user with the **FORCE_UNLOCK** permission can unlock it. The **FORCE_UNLOCK** permission is only granted to the **admin** system role by default.

> Users cannot lock a VDB with refresh policies assigned to it or have any active engine job running at the time of locking.

The VDB details page has the options to lock and unlock a VDB under the Actions menu. All VDBs are in the unlocked state by default, thus, the only option available at first is to lock the VDB.
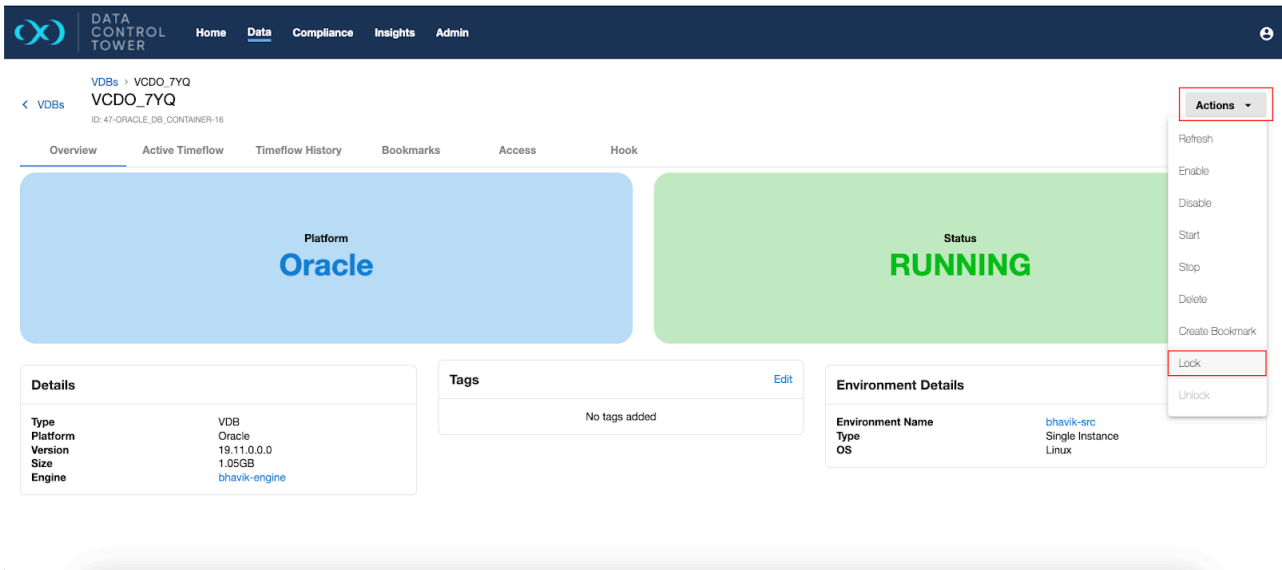
A warning message is shown on the VDB detail page if the VDB is locked by a user.
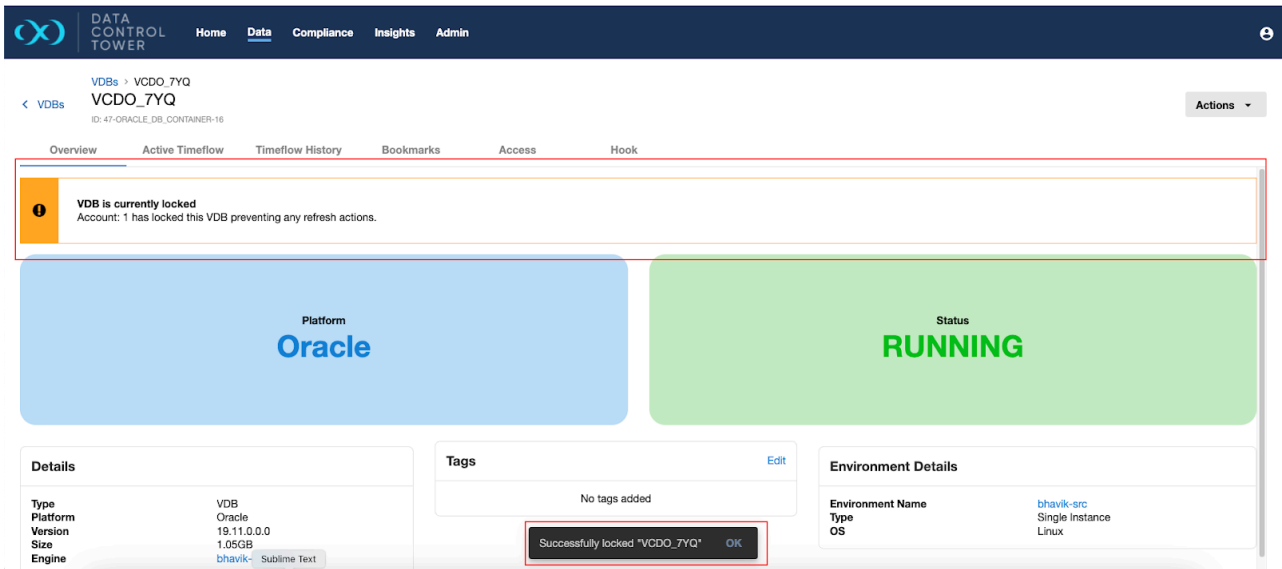
Select the Actions menu on a locked VDB to show the unlock action.

8.6.5.1 Locking and unlocking a VDB via API

DCT also provides a capability for an account to lock the VDB on behalf of another account, but this is an API only feature. For an account to be able to lock the VDB on behalf of another account, it needs to have the `LOCK_FOR_OTHER_ACCOUNT` permission on the particular VDB.

Formal documentation around the API signature and related payloads can be found via DCT API documentation[120] in the **Developer resources** section. Below is an example of an account requesting to lock a VDB on behalf of an account with `account_id` 2.

```
curl --location 'https://<APPLIANCE ADDRESS>/v3/vdbs/<VDB ID or NAME>/lock' \
--header 'Authorization: <APIKEY>' \
--header 'Content-Type: application/json' \
--data '{
    "account_id" : 2
}'
```

8.7 Linking Wizard

8.7.1 Overview

The DCT version 12.0.0 update marks a significant enhancement to data virtualization capabilities in DCT by introducing the Linking Wizard, which guides the user through intuitive steps to create a dSource.

As of this release, DCT 13.0.0 supports the integration of these types of database sources:

- **Oracle** sources

---

120 https://dct.delphix.com/docs/latest/api-references

- Staging Push supported
- **MSSQL** sources
    - Staging Push supported
- **ASE** sources
    - PostgreSQL, Db2, MySQL, HANA, and MongoDB
- **AppData** sources

8.7.1.1 Limitations in MSSQL dSource linking

The ability to link a dSource from backups managed by third-party tools like NetBackup or CommVault is not supported for MSSQL databases as of yet. Users can still create dSources from live MSSQL databases.

This is a known limitation and may be addressed in future updates to expand the versatility and applicability of the dSource feature in diverse data management scenarios.

8.7.1.2 Prerequisites

To utilize the dSource creation feature effectively in DCT 12.0.0, certain prerequisites must be met:

- **Continuous Data Engine registration**: A prerequisite for dSource integration is having a Continuous Data Engine already registered to a DCT instance. This setup is crucial for enabling the core functionalities of data virtualization and synchronization.
- **Availability of eligible sources**: It is necessary to have access to eligible database sources that can be linked as a dSource. These sources should not already be functioning as a dSource, to avoid redundancy and ensure optimal utilization of the new feature.

8.7.2 Using the Linking Wizard

This new wizard can be launched from the dSource list page, giving the user the ability to create a dSource from DCT. The steps are outlined below:

8.7.2.1 1. Select Data Source step

Navigate an alphabetically sorted list of data sources, or utilize the search function and pagination to locate and choose the preferred source.

- Upon selecting a source, essential details such as the **Instance**, **Environment Name**, and **Engine Name** are displayed on the right-hand side of the screen. **Database Authentication** will be required.
- To that point, a crucial part of this step includes a database connectivity check. The **Validate** button verifies your connection, ensuring a secure and authenticated progression to the next stages.

8.7.2.2 2. dSource Type step

In this step, select the nature of the dSource being linked. The options available include:

- **Source Based dSource**: linked directly to a source database.
- **Oracle Staging Push**: linked to a user-managed Oracle staging database.
- **MSSQL Staging Push**: linked to a user-managed MSSQL staging database.

Following the selection of the dSource type, users must then choose the associated engine from the dropdown menu, which lists the available engines that have been registered in the DCT environment. This setup dictates the subsequent configuration steps to ensure the linked dSource is correctly aligned with the user's infrastructure and database management strategy.

8.7.2.3 3. dSource Configuration step

Name your dSource here. By default, the dSource name mirrors the name of the source, but it can be modified. Choose the **Target Group** within the engine to which the new dSource will be added. Add **Notes**, and a **Tag Key** and **Tag Value** pair, if needed. You must also configure any necessary settings required, based on the source.

8.7.2.4 4. Data Management step

The step contains various dSource configurations for optimal data management. The wizard fetches and pre-populates the default values for various data fields, saving you time and reducing manual entry errors.
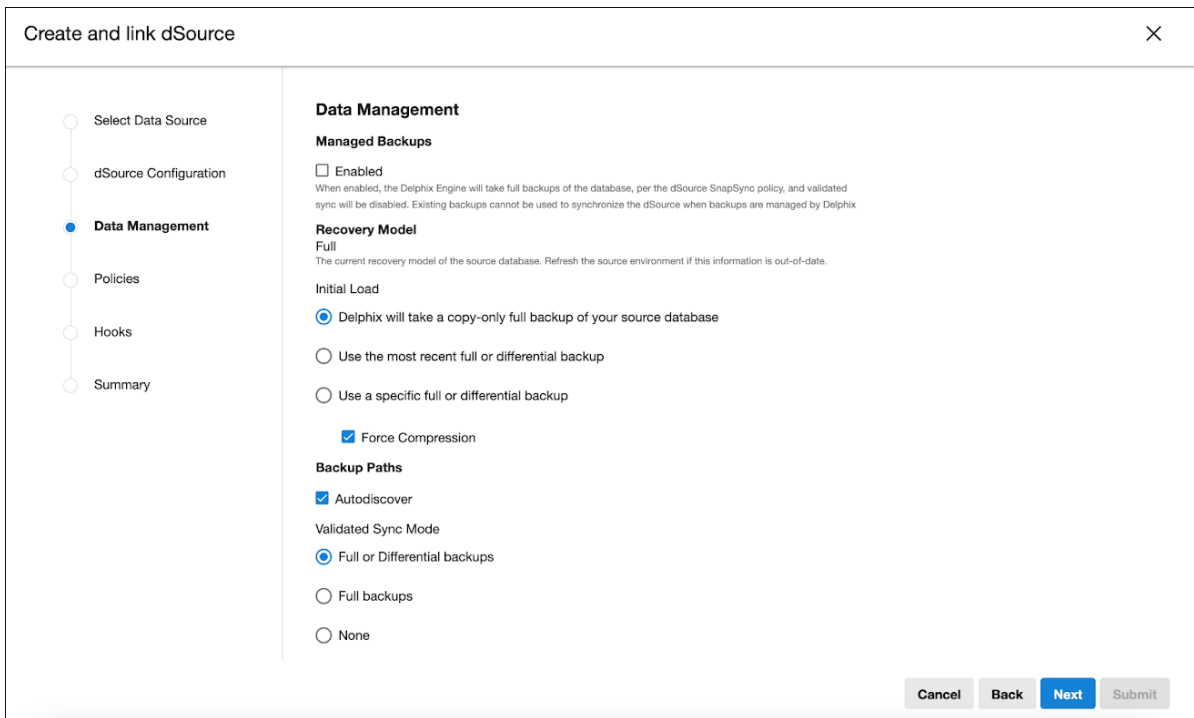
8.7.2.5 5. Policies step

Tailor the SnapSync and Retention policies according to your requirements. Default policies are selected initially, but these settings can be modified through the **Select Policy** dialog.
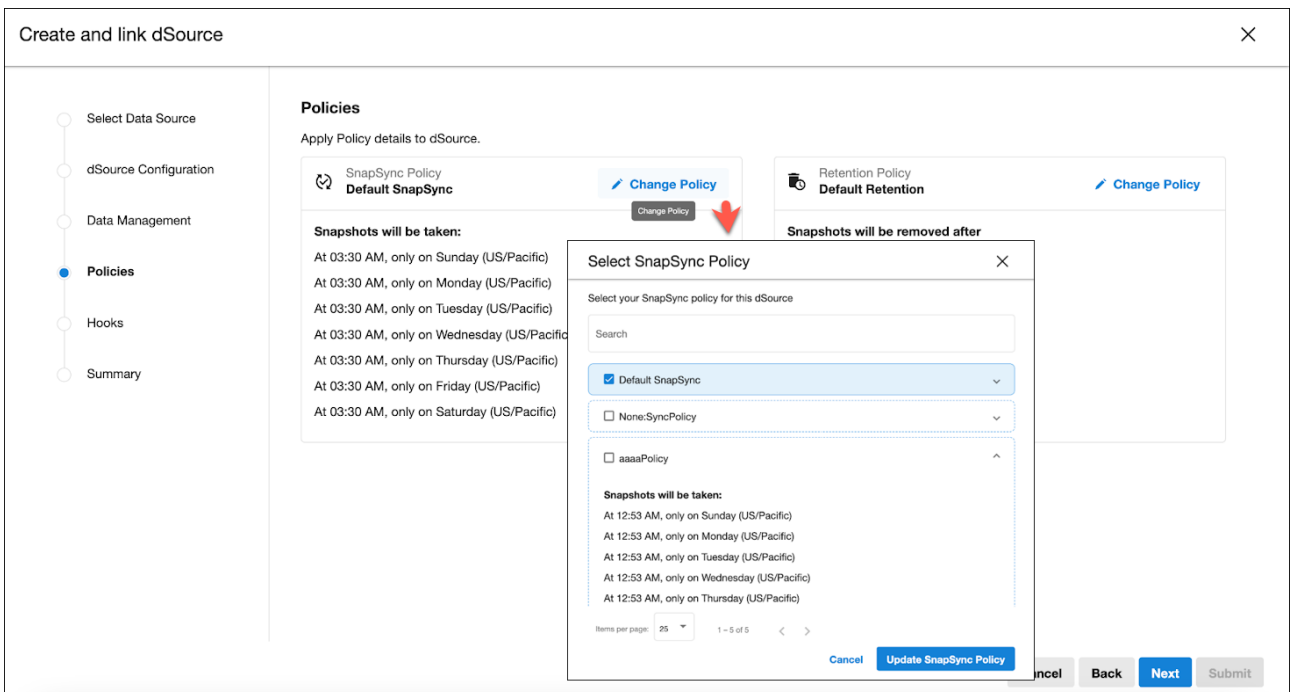
8.7.2.6 6. Hooks step

Enhance the dSource with custom created Hooks. Use the **+ Hook** button to define the scripts that will execute at various stages of the process, offering greater control and customization over the dSource creation workflow.

> ℹ️ For Oracle dSources in the linking wizard, a Reorder button is available to rearrange hooks for different operations at various hook points. Additionally, you can add, edit, delete, and reorder hooks on the linked dSource via the dSource Details page under the Hooks tab.

8.7.2.7 7. Summary step

Before finalizing, a comprehensive summary of all your configurations is presented. This overview allows you to review and confirm that every detail aligns with the intended setup, ensuring accuracy and completeness. **Submit** the configuration to finish linking your database.

8.7.3 Adding an AppData dSource

The steps to add a dSource in DCT, as outlined in the Linking wizard[121] page, should be still be followed. However, this page covers details specific to adding an AppData dSource to DCT. AppData dSources allow for efficient handling and integration of application data.

Before initiating the linking process, ensure that an AppData type source is already added in DCT.

8.7.3.1 Steps effected during the Linking process

8.7.3.1.1 Data Source step

1. **Select AppData source**: Initiate the dSource Linking wizard and select the desired AppData source. This selection is crucial in determining the linking procedure.

---

[121] https://dct.delphix.com/docs/latest/linking-wizard

2. **Dynamic UI form**: For an AppData database, a toolkit associated with it contains a `linkedSourceDefinition`. This definition, essentially a JSON schema, is rendered as a Dynamic UI form alongside the Source list. It outlines the necessary steps for linking sources of this type.

- **Direct AppData source**: If selecting a direct AppData source, there's an option for adding advanced settings, provided the `linkedSourceDefinition` has `usesGrandfatheredAppDataProperties` set to true.

8.7.3.1.2 Data Management step

- This step is conditional and appears if:
  - a. The source is a staged type.
  - b. The source, whether direct or staged, requires snapshot parameter configuration.
- **Snapshot Parameters**: These are determined by the toolkit's `snapshotParametersDefinition` and are presented as a Dynamic UI form.
- **Staging Environment selection**: Users can choose the appropriate staging environment and user credentials during this step.

Create and link dSource   ×

dSource Type

Select Data Source

dSource Configuration

Data Management

Policies

Hooks

Summary

Data Management

Staging Environment

⌄

User

postgres   ⌄

Snapshot Parameters

Parameters for the first snapshot taken on the new dSource.

☐ resync

snapParamArray   + Add

☐ Snapshot Parameter Boolean Value

A boolean value.

Snapshot Parameter Enum Value   ⌄

An enum value.

Snapshot Parameter Integer Value

An integer value.

Snapshot Parameter Number Value

An Number value.
Snapshot Object

Cancel   Back   **Next**   Submit

8.7.3.1.3 Completion

Continue through the linking process and after submission, the process of creating an AppData dSource begins. The new dSource will soon be visible on the dSource page, reflecting the configurations made during the linking process.

8.7.4 Adding an Oracle Staging Push dSource

This page outlines the process for adding an Oracle Staging Push dSource within Delphix Data Control Tower (DCT). This functionality is crucial for managing Oracle databases, particularly in environments where staging operations are pushed to target engines. Integrate Oracle Staging Push dSources into your DCT setup by following the Linking Wizard steps.

8.7.4.1 Prerequisites

Ensure that a target engine, with the source backup already cloned, is registered in the DCT before beginning the linking process.

8.7.4.2 Linking process

8.7.4.2.1 dSource Type step

1. **Select dSource type**: Begin the dSource Linking Wizard and choose the 'Oracle Staging Push' option.
2. **Target engine selection**: From the engine dropdown, select the desired target engine where the source backup is located.

8.7.4.2.2 dSource Configuration step

- **Select dSource configuration**: Depending on the type of Oracle database (CDB, PDB, or single tenant), select the appropriate dSource configuration.
- **Database details**: Based on the previous selection, enter the necessary details that would appear as shown in the screenshot below.

Create and link dSource ✕

dSource Type	
dSource Configuration	
Data Management	
Hooks	
Summary	

dSource Configuration

dSource Name
DSod_7491E1YN

Target Group
Untitled ▼

Database Type

○ Non Multi Tenant

◉ CDB

○ PDB

Staging Environment
delphix.com ▼

User
oracle ▼

Repository
/u01/app/oracle/product/19.8.0.0/dbhome_1 ▼

CDB Name
CDOMLOSR

CDB Unique Name
CDOMLOSR

SID
CDOMLOSR

Mount Base
/mnt/staging

Custom Environment Variables

Key
var1   🗑 Delete

Value
var1value

＋ Add

Cancel   Back   Next   Submit

- **Staging database parameters**: An option to configure Staging Database Parameters is available. If selected, it adds a new step for entering these parameters.

Create and link dSource   ✕

dSource Type

dSource Configuration

Staging Database Parameters

Data Management

Hooks

Summary

Staging Database Parameters

| Default   ▾ | **Save as New Template**

Table   Text

🗑   +

Name	Value
parameter	parametervalue

Cancel   Back   **Next**   Submit

8.7.4.2.3 Data Management step

- **Enable LogSync**: This step involves the option to enable log synchronization for real-time data updates and management.

Create and link dSource   ✕

dSource Type

dSource Configuration

Staging Database Parameters

Data Management

Hooks

Summary

Data Management

LogSync

☐ Enable (Archive Only)

LogSync is an add-on service that can synchronize the Delphix Engine within seconds of any change to a source database. LogSync requests and stores changes to log files. By coordinating changed log blocks with snapshots from SnapSync, users can provision VDBs from a specific point in time, either an arbitrary time, or a time relative to the last transaction.

Cancel   Back   **Next**   Submit

8.7.4.2.4 Summary step

- **Review configuration**: Examine the details of the summary, which includes information about the dSource type, target engine, database details, and any additional parameters set in the previous steps, to ensure all configurations are correct and aligned with the requirements of the Oracle Staging Push dSource.

8.7.4.3 Completion

Upon confirming and submitting the details in the Summary step in the Linking Wizard, the process of creating an Oracle Staging Push dSource is initiated. The new dSource will be integrated into the DCT environment and will be available for use once the setup is complete.

8.7.5 Adding a MSSQL Staging Push dSource

This page outlines the process for adding a MSSQL Staging Push dSource within Delphix Data Control Tower (DCT). This functionality is crucial for managing MSSQL databases, particularly in environments where staging operations are pushed to target engines. Integrate MSSQL Staging Push dSources into your DCT setup by following the Linking Wizard steps.

8.7.5.1 Prerequisites

Ensure that a target engine, with the source backup already cloned, is registered in the DCT before beginning the linking process.

8.7.5.2 Linking process

8.7.5.2.1 dSource Type step

1. **Select dSource type**: Begin the dSource Linking Wizard and choose the 'MSSQL Staging Push' option.
2. **Select Associated Engine**: From the engine dropdown, select the desired target engine where the source backup is located.

8.7.5.2.2 dSource Configuration step

- **Configure dSource details**: Enter the dSource Name, Database Name, select a Target Group from the dropdown, and enter Notes or add Tags, if desired.

8.7.5.2.3 Data Management step

- **Staging Environment selection**: Choose the appropriate Staging Environment to host the staging database from the dropdown list.
- **Repository selection**: Select the corresponding Repository version.

Create and link dSource   ✕

Data Management

dSource Type

dSource Configuration

● Data Management

Policies

Hooks

Summary

Staging Environment

.co

This environment will host the staging database used for validated sync.
Repository

SQL2016

Cancel   Back   **Next**   Submit

8.7.5.2.4 Summary step

- **Review configuration**: Examine the summary, which includes includes information about the dSource type, target engine, database details, and any additional parameters set in the previous steps, to ensure that all configurations are correct and aligned with the requirements of the MSSQL Staging Push dSource.

8.7.5.3 Completion

Upon confirming and submitting the details in the Summary step in the Linking Wizard, the process of creating an Oracle Staging Push dSource is initiated. The new dSource will be integrated into the DCT environment and will be available for use once the setup is complete.

8.7.6 Advanced dSource creation options

8.7.6.1 Overview

With the release of DCT 15.0.0, advanced options when creating dSources in DCT have expanded.

- Users can now select external backups from NetBackup or Commvault when adding a Microsoft SQL Server (MSSQL) dSource.
- For MSSQL and Adaptive Server Enterprise (ASE) sources, the additional option to use a password vault for database authentication is available with the "Select Data Source" step.
- Transparent Data Encryption (TDE) for Oracle Container Database (CDB) type staging push dSources can now be activated.

- The option to use non-SYS login credentials when linking an Oracle Single Tenant (ST) database is now available, which enhances security and flexibility.

8.7.6.2 Prerequisites

To utilize these new features effectively, adhere to the following prerequisites:

- The source database intended for linkage should already be registered within DCT.
- For those opting to use NetBackup or Commvault as their external backup solution when linking an MSSQL source, the necessary backup credentials must be prepared in advance.

8.7.6.3 Linking Wizard changes

8.7.6.3.1 NetBackup/Commvault

In the Data Management phase of the Linking Wizard, a "show advanced" option is now available. This addition allows for the specification of external backup configurations, catering to users' preferences for NetBackup or Commvault solutions.

4 NetBackup checkbox

When selecting NetBackup as the external backup choice, users have the further opportunity to integrate NetBackup configuration templates. This is achieved by selecting the "Configure Netbackup Config Template" checkbox, available in the subsequent step.

5 Commvault checkbox

Upon specifying the desired external backup configuration, it is essential for users to validate these settings to ensure accuracy and compatibility.

The final review of these configurations is available in the Summary step. This stage serves as a last checkpoint before the submission process, where users can verify all entered details are correct. Once satisfied, finish with the wizard to create the dSource, incorporating the selected external backup settings.

8.7.6.3.2 PasswordVault authentication

Upon reaching the Data Source step, users now have the choice to authenticate using a Password Vault. This update supports integration with leading vault solutions, including **HashiCorp**, **CyberArk**, and **Azure**. Selecting this authentication method leverages the security framework of the respective provider, in an effort to provide versatile, secure, and efficient tools for managing database connections within DCT.

8.7.6.3.3 TDE for Oracle CDB

The process of creating a Container Database (CDB) dSource within the Oracle Staging Push portion of the Linking Wizard now includes an option for enabling Transparent Data Encryption (TDE). This enhancement is designed to elevate data security by offering encryption at the storage level, thus safeguarding sensitive information.

Upon enabling TDE, users are presented with a choice among three KeyStore configuration types to best suit security needs:

- **File-based KeyStore**: A traditional, file-based approach to store encryption keys securely on disk.
- **Hardware Security Module (HSM)**: A physical device that manages digital keys for strong authentication and provides a higher level of security.
- **Oracle Key Vault (OKV)**: A comprehensive key management solution that centralizes control over encryption keys and secrets.

8.7.6.3.4 Oracle ST non-SYS credentials

The Oracle Single-Tenant (ST) database linking process within the Linking Wizard now includes the ability for users to provide non-SYS login credentials, allowing for a more secure and restricted access approach. This would be particularly beneficial for organizations that adhere to strict security policies requiring minimal privilege use.

Furthermore, users can now view the nodes of their Oracle ST cluster directly in the Source Instances grid. This feature provides a clearer overview of the database topology.

Create and Link dSource   ✕

dSource Type

Select Data Source

dSource Configuration

Data Management

Policies

Hooks

Summary

Select Data Source

Select an available Data Source to connect and create a dSource

ℹ Validation will succeed if the OS User Credentials can access the selected database, and the entered Database User Credentials will be ignored. If the OS User Credentials cannot access the database, the Database User Credentials will be used.

Username

Password

Validate

Hide advanced   ⌃

Non-SYS Username

Non-SYS Password

Cancel   Back   Next   Submit

9 Continuous Compliance workflows

> ⚠️ Compliance Engines limit any syncing operations while a profiling or masking job is running. When using DCT job **move**, **execute**, or **migrate** operations, please ensure that the target Compliance Engine is in an idle state. Future Compliance enhancements to DCT will remove this limitation.

With the ability to distribute and run jobs, DCT enables advanced Compliance Engine architectures to be orchestrated and monitored using DCT's real-time, persistent relationships with connected Compliance Engines. When syncing a Compliance Engine, DCT will create references for all Compliance jobs on that Engine. These will show up as unique objects tracked by DCT that can now be leveraged with job move APIs.

9.1 Listing and searching compliance jobs

When a Compliance Engine is registered with DCT, compliance jobs (referred to as MaskingJobs within the DCT API) on the Engine are automatically ingested and presented as DCT MaskingJob objects.

Example of **listing all MaskingJobs**:

```
curl -X 'GET' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>'
```

Example of **searching for OnTheFly MaskingJobs**:

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/search' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "filter_expression": "is_on_the_fly_masking eq true"
}'
```

With the new job move APIs, DCT can now be used to power two advanced masking reference architectures[122]: **Software Development Lifecycle (SDLC)** and **Horizontal Scale** architectures. SDLC enables the separation of duties for the development, quality assurance, and production use of masking jobs whereas Horizontal Scale enables the use of a central configuration engine with the movement of jobs to headless compute engines.

---

122 https://masking.delphix.com/docs/latest/managing-multiple-engines-for-masking-1

To enable these architectures, DCT has introduced three new operations: Job Copy, Job Execute, and Job Migrate:

- **Copy**: Supports SDLC by copying a job, but maintaining separate references in DCT.
- **Execute**: Supports Horizontal Scale by copying a job, but maintaining the same reference between two copies. DCT will also keep both of these copies in sync.
- **Migrate**: Supports the movement of a single instance from one engine to another.

9.2 Consolidated operations (intelligent syncing)

DCT has simplified the set of operations required to move a job and its dependencies. Previously, orchestrating movement of jobs required three separate API calls: Job Sync, Global Object Sync, and Credentials Update (on the newly created job). DCT has now consolidated all three of these operations into each of the job move APIs. In addition, if two jobs are held in sync (see Job Execute), DCT will auto update synced jobs whenever one of those jobs has been modified (i.e. updated rule set, new algorithms, etc.).

> In order to transfer connector credentials with a job as part of the job move, you will need to associate those credentials using the connector credentials API. See sample code below on how to update credentials.
>
> Example of updating a MaskingJob with connector credentials:

```
curl -X 'PATCH' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/d53812ce-9186-485d-a388-44bc52087ead'
 \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "connector_username": "user123",
  "connector_password": "password123"
}'
```

9.3 Managing engines (Continuous Compliance)

DCT provides a near real-time list of all connected Continuous Compliance engines and lists them in an aggregate view. From the below screen, Delphix administrators can easily view and manage their engine connections.

From this screen, administrators can manage engine connects via the "Connect Engine" button on the top right corner. By clicking this button, the below window will appear asking for connection details.

> DCT will access the engine as a registered user and, as detailed in the Deployment section, requires both a username and password as well as admin-level access to the engine. For compliance engines, select "Masking" type when registering an engine.

The first two boxes should be checked for DCT to automatically create predefined tags on discovered objects ingested from the engine. This can be done in the **Tags** step of the **Connect Engine** wizard. Checking the box will make Compliance Jobs and Connectors automatically get tagged with the name of their Environment and/or Application on the engine.

Checking the third box will include the engine name as a tag on ingested objects. Finally, checking the last box allows users to set custom tags on ingested objects. Tags will be applied on Jobs, Connectors, and Algorithms.

Connect Engine

×

Engine Details

Authentication

Security

Tags

Summary

Tags

Add Engine Tags

| Enter Tag Key | Enter Tag Value | + Tag |

Add Tags to Discovered Objects

⚠ **Warning Tags can Grant Access**
Caution: Auto-tagging can modify object access. Review permissions carefully after auto-tagging objects.

☑ Include engine environment information as a tag on objects with the key "dlpx-environment"

☑ Include engine application information as a tag on objects with the key "dlpx-application"

☑ Include engine name as a tag on objects with the key "dlpx-engine"

☑ Include custom tags on all objects from this engine

| Enter Tag Key | Enter Tag Value | + Tag |

Cancel   Back   **Next**   Submit

ℹ ~~Finally, since auto-tagging can have an impact on access control, non-admins will require special permissions to configure auto-tagging~~ Auto-tagging can modify object access. Review permissions carefully after auto-tagging objects.

All existing and new objects from then on will automatically get the configured tags, which are no different from ordinary tags that can be added to objects. The auto-tagging configuration can later be updated in the Data Engine Details page by going to the **Action** menu and selecting **Auto-tagging Configuration**. This will open up a window with a checkbox to enable or disable auto-tagging.

Disabling auto-tagging will not remove any of the tags that were previously added. Instead, new ingested objects will not have tags automatically applied. In order to delete these tags, this must first be disabled and the tags will need to be deleted manually on all affected objects.

Since auto-tagging can have an impact on access control, non-admins will require special permissions to configure auto-tagging.

- To configure auto-tagging for pre-defined tags when registering an engine for the first time, the account must have a role with the **Configure pre-defined auto-tagging at registration** permission.
- To update the auto-tagging configuration for pre-defined tags on a pre-registered engine, the account must have the **Configure pre-defined auto-tagging** on the ENGINE object they wish to update.

Configuring custom tags also requires another level of permissions.

- To configure auto-tagging with custom tags when registering an engine for the first time, the account must have a role with the **Configure custom auto-tagging at registration** permission.
- To update the auto-tagging configuration's custom tags on a pre-registered engine, the account must have the **Configure custom auto-tagging** on the ENGINE object they wish to update.

9.3.1 Engine overview

Individual engine details can be seen and acted upon by clicking down on a particular engine detailed view. Once clicked, users will be sent to an "overview" tab that provides relevant metadata related to the engine.
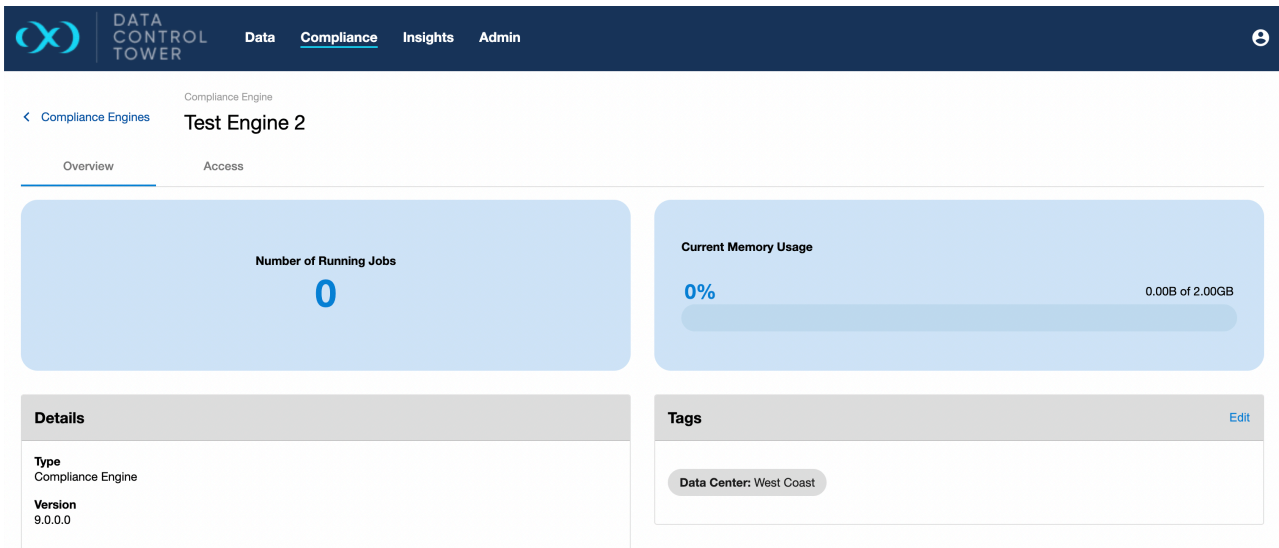
9.3.2 Engine-based operations access

Users are able to audit which users have access to this particular engine, what access group they belong to, and the associated permissions that each user has on this engine. Admins are able to click on the "View" button to access further details under the access control screen related to that specific user.
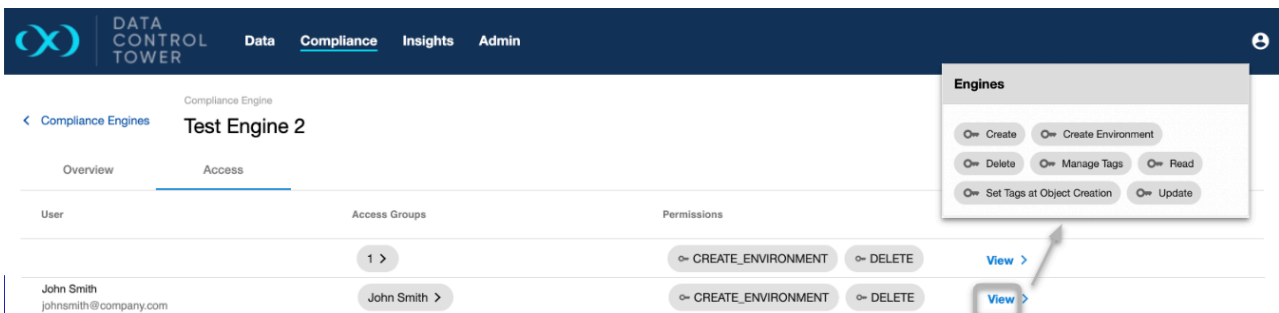
9.4 Compliance jobs

> ℹ️ For a Compliance job to be successfully moved (copied or migrated) from a source to a target engine, all global objects referenced by the job (such as algorithms and domains) must be synchronized between both engines. This provides consistency for the masking operations across different environments.

> ℹ️ When initiating job operations through DCT, ensure the target Compliance Engine is not engaged in any profiling or masking activities. Future DCT updates will eliminate the need to do this.

9.4.1 Job UI

9.4.1.1 Global compliance jobs list

When connected to a Continuous Compliance engine, DCT will sync and create references to every compliance job on the engine. All of those job references can be found in the global compliance list, which is a taggable, filterable, sortable, and searchable list of all compliance jobs across a connected Delphix ecosystem.

9.4.1.2 Compliance job overview

Individual compliance job details can be seen and acted upon by clicking down on a particular compliance job detailed view. Once clicked, users will be sent to an "overview" tab that provides relevant metadata related to the VDB.

9.4.1.3 Access tab

Users are able to audit which users have access to this particular compliance job, what access group they belong to, and the associated permissions that each user has on this job. Admins are able to click on the "View" button to access further details under the access control screen related to that specific user.
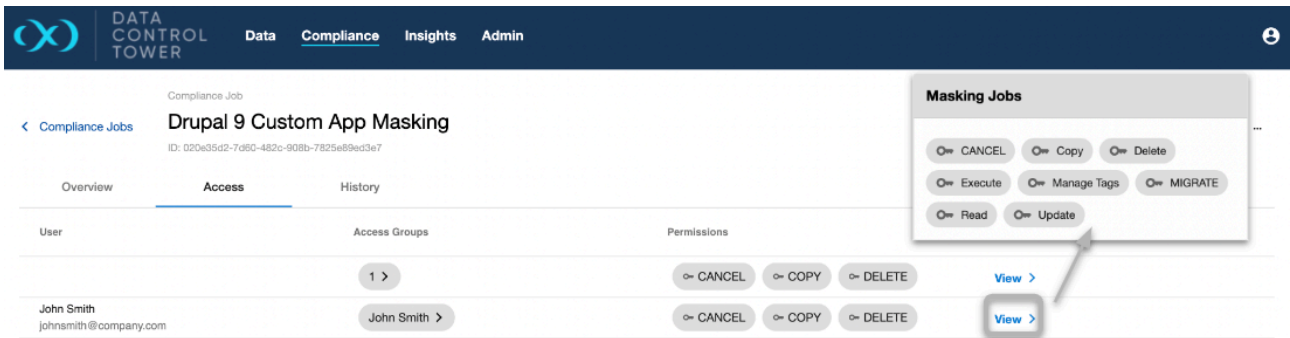
9.4.1.4 Execution history tab

The compliance jobs page also includes job execution history. Execution information will be recorded and displayed on this tab, and includes the status (success, running, failed), run time, submit, and end timestamps, and engine on which it ran. Additionally, clicking the "details" button for an execution will display its report.
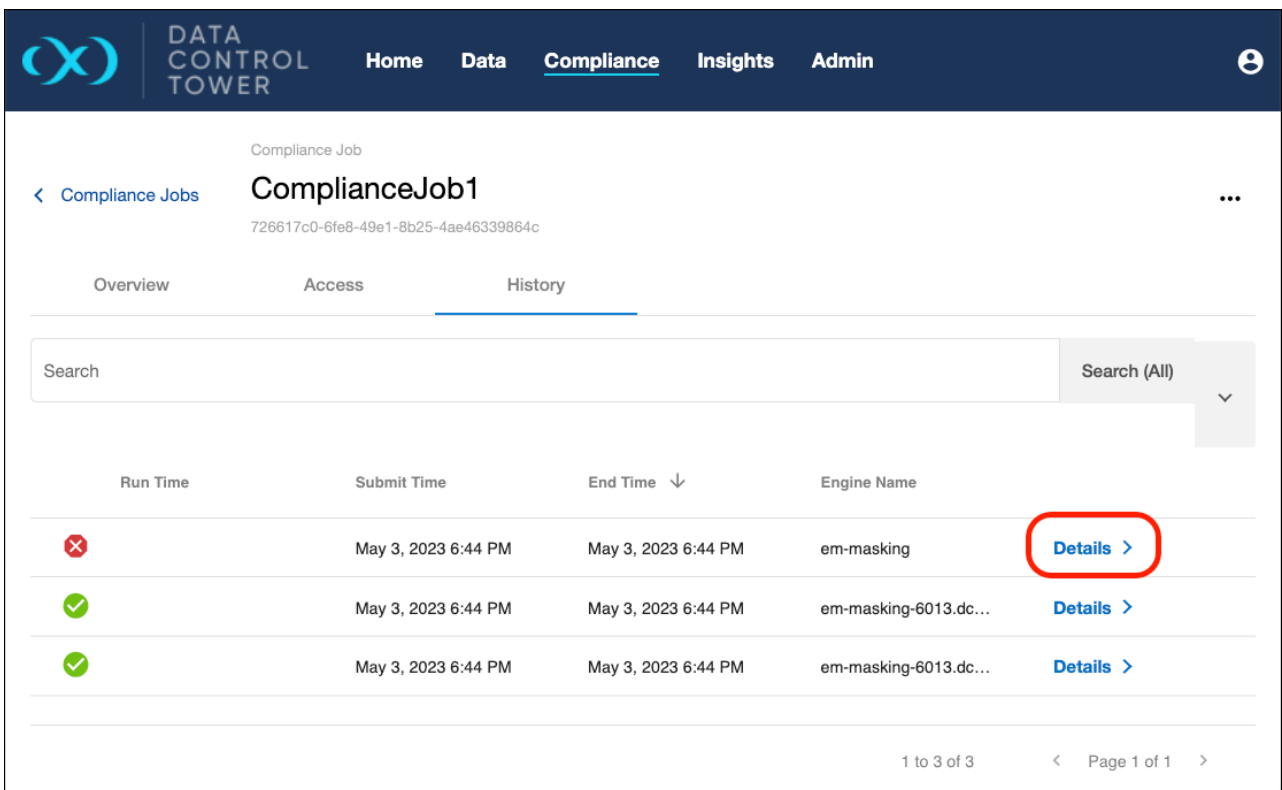
9.4.1.5 Execution details

The execution details view includes a list of execution events and the execution log. This is particularly useful when troubleshooting failed executions.

> 📰 Details for successful executions may also have events and logs which include relevant information, such as warnings.

9.4.2 Compliance jobs (read-only)

This iteration aims to elevate the existing compliance jobs feature in DCT by bringing changes to the existing UI experience. These changes include making additional information on jobs and executions available, several UX enhancements, and introducing a brand-new Compliance Job Execution details page—which offers a more granular view into executions, such as the execution components, the events that occurred during, and the recorded logs.

9.4.2.1 Change list

9.4.2.1.1 Compliance Jobs page

- The **Execution Type** column was previously named *Job Type*. It shows whether the execution type is Standard or Hyperscale.
- The **Type** column is newly introduced. It shows whether the job is for Masking, Discovery, Tokenization, or Reidentification.
- The **Is On The Fly Masking** column is newly introduced, which is hidden by default, but can be manually toggled on.
- The **Source** column is newly introduced. It shows the engine name in the form of a link that redirects the user to the engine's details page.
- The **Start Time** and **Run Time** columns are newly introduced in lieu of the previous *Last Completed Execution Date* column.

9.4.2.1.2 Compliance Job details page

- In the top execution status tile section, the **Start Time** now shows both the date and time (previously it was only showing time).

- The top execution status tile now has a **View Job Execution Details** button, which will redirect to the details page for this job's latest execution (the execution details page is a new UI introduced in this feature).
- The **Details** tile now shows much more metadata for the job. The information shown is as follows:
    - Job ID and Job Name
    - Execution Type (Standard or Hyperscale)
    - Compliance Job Type (Masking, Discovery, Tokenization, or Re-Identification)
    - Creation date
    - Environment name on the engine
    - Application name on the engine
    - Rule Set
    - Source (Engine)
    - Only if this is an On-The-Fly Job, we also show the Source Connector

9.4.2.1.3 Execution History tab

- These new columns were added. Some columns are hidden by default and need to be manually toggled on (noted below).
    - **Rows Processed**: The number of rows masked. This is not applicable to JSON.
    - **Rows Total** (hidden by default): The total number of rows. This is not applicable to JSON.
    - **Bytes Processed**: The number of bytes masked. This is only applicable to JSON files and ASDD profiling jobs on XML files.
    - **Bytes Total** (hidden by default): The total number of bytes. This is only applicable to JSON files and ASDD profiling jobs on XML files.
    - **Started By** (link): DCT user who started this job (from DCT). If the job was not started from DCT, this will not be set.
    - **Queue Duration** (hidden by default): how long this execution spent in the queue (between submit and executing) on the engine.
    - **Run Duration**: How long this execution took to run.
    - **Total Duration** (hidden by default): How long it took when this job was started from DCT to when it completed on the engine.
    - **Start Time**: When the job was started to execute.

9.4.2.1.4 Execution Details page (new)

This feature introduces a new UI for compliance job execution details. In the **Compliance Job Execution History** list, each execution has a View button, which will take you to the details page for that execution.

The **Overview** tab for an execution details page includes a status tile at the top which shows the steps the execution has taken.

It is then followed by a **Details** tile, which includes a few metadata for the execution, including:7

- Name, ID, Compliance Job Type for the parent job of this execution.
- Environment and Application name.
- The number of execution component masked/discovered, yet-to-be masked/discovered, and the total number.

9.4.2.1.5 Execution Components tab

This tab is part of the execution details page and shows a list of execution components for the execution. By default, we show the name, progress (in percentage), status, duration, and rows processed columns. But there are a few additional columns that can be manually toggled on. Including

- Rows Total
- Bytes Processed
- Bytes Total
- Non-Conformant Data

In addition, each execution component in the list has a Details button which will open a dialog showing the events and logs that occurred for this execution component. Please note, the events and logs are only available when the execution is in a final state (succeeded, failed, or warning). Before then, there is no data to retrieve and therefore the execution component will not have the button.

9.4.2.1.6 Execution Logs tab

This tab is part of the execution details page and shows the events and logs for the execution. The events are shown on the left and logs are shown on the right.

Events and logs are only available when the execution enters a final state (i.e. succeeded, failed, or warning). Before then, this tab will show a warning message. Once the execution transitions to a final state, this page should update once the data becomes available.

9.4.3 Discovery jobs (read-only)

DCT has had UI and APIs to expose masking jobs, but not for discovery jobs (profiling jobs). This feature aims to enhance the compliance job entity to incorporate both masking and discovery jobs. With this change, in the compliance job UI, users should now be able to see the discovery jobs alongside the compliance jobs. All existing UI pages that apply to compliance jobs also apply to discovery jobs (certain metadata for read-only may apply/not apply depending on masking vs discovery). And users can also execute the discovery jobs from DCT side in the same way.

The compliance jobs list has been modified to show both the compliance jobs, as well as the discovery jobs. The new **Type** column (which takes the value of Masking, Discovery, Tokenization, or Re-Identification) helps to differentiate the jobs in this list.

Details

Job Name
Profile VSAM

Discovery Policy
ASDD Standard

Job ID
6360e968-0392-441e-9c87-e912955172b4

Rule Set
Profile Simple Mainframe

Execution Type
Standard

Source
gli-me-v25

Compliance Job Type
Discovery

Created
Sep 19, 2024 3:41 PM PDT

Environment
Test Masking Env

Application
Test Application

All the existing pages for compliance jobs also apply for discovery jobs, but discovery jobs still have some unique metadata. In the Overview tab for a compliance job's detail page, if the job is discovery, there will be a link to the Discovery Policy, as shown above.

Table Name	Column Name	File Name	Field Name	Data Class	Algorithm
Cars	mileage			ZIP	RepeatFirstDigit
Driver	age			AGE	dlpx-core:Age SL
Driver	firstname			FIRST_NAME	dlpx-core:FirstName
Insurance	insurance_company_name			COMPANY_NAME	BusinessLegalEntityLookup

Another feature unique to discovery jobs is the Discovery Result tab, which lives inside the execution details page. This tab shows the list of tables/columns or files/fields discovered, and the data classes and algorithms that were assigned to them.

Discovery results are not available until the execution is in a final state (succeeded, failed, or warning). If users enter this tab beforehand, there will be a warning message, and no results will be shown. The page will monitor the status of the execution. After the execution is final and after the data is collected by telemetry, the page will update accordingly.

9.4.4 Copy job

A copy job enables the creation of an independent duplicate of a compliance job, which is essential for supporting Software Development Life Cycle (SDLC) workflows (by enabling the independent management of each job instance). Unique references are maintained for every copied job.

9.4.4.1 DCT UI process

To execute a copy job through the DCT user interface:

1. Navigate to the **Details** page of a compliance job.

2. Click on the ellipsis in the top right corner and select **Copy**.

3. A dialog will appear, prompting you to select the **target engine**, assign a new **name** to the job, specify **source and target environment details**, and add any relevant **tags**.

9.4.4.2 API process

To execute a copy job via the DCT API, the request must include the **target engine and environment details** where the job will be replicated. Using the COPY API with the same target engine and environment acts as a re-sync mechanism, rather than generating a new DCT `MaskingJob` entity.

Example of API usage for copying a `MaskingJob`:

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/d53812ce-9186-485d-a388-44bc52087ead/
copy' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "target_engine_id": "2",
  "target_environment_id": "prod-env"
}'
```

9.4.4.3 Force overwrite

When copying a compliance job, DCT provides a **force overwrite** option designed to override any existing environment based objects on the target engine with the same name, ensuring the latest job is applied without needing to manually resolve naming conflicts or version discrepancies. This can be particularly useful when updating a job with significant changes that need to be reflected immediately across environments.

Connector credentials are not automatically copied to the target engine due to security considerations. Therefore, the credentials must be explicitly set on the target engine's connector. DCT facilitates this by allowing the search, update, and testing of connectors directly through its UI.

Examples of managing connectors for a `MaskingJob`:

9.4.4.3.1 Listing connectors

```
curl -X 'GET' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/d53812ce-9186-485d-a388-44bc52087ead/
connectors' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>'
```

9.4.4.3.2 Updating a connector's credentials

```
curl -X 'PATCH' \
  'https://<APPLIANCE_ADDRESS>/v3/connectors/2-DATABASE-23' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "username": "USER123",
  "password": "password123"
}'
```

9.4.4.3.3 Testing a connector

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/connectors/2-DATABASE-23/test' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>'
```

9.4.5 Execute job

With recent updates, all job executions are now centralized to the source engine. The Execute endpoint has been optimized to support efficient job management directly from the source engine, ensuring any updates or modifications are directly applied to the job on the source engine.

This enables users to directly identify, copy, and run masking jobs without the need to manage multiple job instances across different engines.

Executing a MaskingJob now requires solely the source engine reference. Continuous Compliance will handle all necessary preparations for job execution within the original environment, eliminating the need for syncing operations to other engines.

Example of executing a MaskingJob:

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/d53812ce-9186-485d-a388-44bc52087ead/
execute' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "source_engine_id": "2"
}'
```

For managing MaskingJob credentials securely, users can update job details including connector credentials through the UPDATE API. This ensures all necessary authentication details are correctly configured within the job settings prior to execution.

Example of updating a MaskingJob with connector credentials:

```
curl -X 'PATCH' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/d53812ce-9186-485d-a388-44bc52087ead'
 \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "connector_username": "user123",
  "connector_password": "password123"
}'
```

9.4.6 Migrate job

The Migrate endpoint facilitates the seamless relocation of a masking job from one engine to another, ensuring the job's uniqueness is preserved throughout the process. This is particularly beneficial for organizational tasks, such as transitioning compliance jobs to a new engine in preparation for the retirement of the original engine, or merging jobs from multiple development engines into a single unit for streamlined administration. This operation guarantees that each job retains a singular instance, preventing the creation

of duplicates. Moreover, the job maintains its original reference within DCT, providing a smooth transition with minimal disruption to existing workflows and configurations.
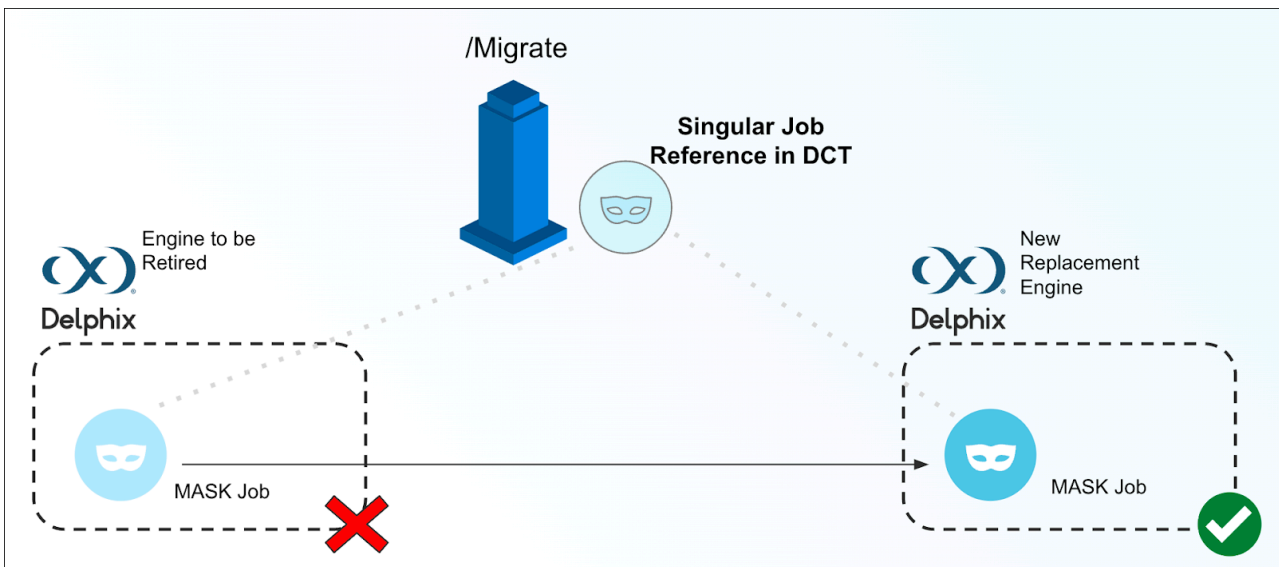
Example of finding all `MaskingJob` originating from engine with ID 2:

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/source-engines/search' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "filter_expression": "source_enigne_id eq '\''2'\''"
}'
```

Example of migrating a `MaskingJob` to new source engine with ID 3 and placing it in the 'prod-env' environment:

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/d53812ce-9186-485d-a388-44bc52087ead/
migrate' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "target_engine_id": "3",
  "target_environment_id": "prod-env"
}'
```

9.4.6.1 Force overwrite

When migrating a compliance job, DCT provides a **force overwrite** option designed to override any existing job on the target engine with the same name, ensuring the latest job is applied without needing to manually resolve naming conflicts or version discrepancies. This can be particularly useful when updating a job with significant changes that need to be reflected immediately across environments.

Connector credentials are not automatically copied to the target engine due to security considerations. Therefore, the credentials must be explicitly set on the target engine's connector. DCT facilitates this by allowing the search, update, and testing of connectors directly through its UI.

Examples of managing connectors for a `MaskingJob` :

9.4.6.1.1 Listing connectors

```
curl -X 'GET' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/d53812ce-9186-485d-a388-44bc52087ead/
connectors' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>'
```

9.4.6.1.2 Updating a connector's credentials

```
curl -X 'PATCH' \
  'https://<APPLIANCE_ADDRESS>/v3/connectors/2-DATABASE-23' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "username": "USER123",
  "password": "password123"
}'
```

9.4.6.1.3 Testing a connector

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/connectors/2-DATABASE-23/test' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>'
```

9.4.7 Delete job

Calling the DELETE API on a MaskingJob will effectively remove the record from DCT (and its execution history) as well as delete the actual masking job on the source engine and on any other engine where the job has been copied to (as a result of execution). The API includes a force option to prevent the action from failing in the event that an engine is unreachable.

Example of **deleting a MaskingJob with the force option**:

```
curl -X 'DELETE' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/d53812ce-9186-485d-a388-44bc52087ead?
force=true' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>'
```

This will return a DCT Job that can be further polled for status updates. Note that if the force option is used and there are ignored errors, details about those errors will be included in the **error_details** and **warning_message** fields of the DCT Job as follows:

```
{
  "job": {
    "id": "722ba51cf70e4e32adbd192b07304bb5",
    "status": "COMPLETED",
    "type": "MASKING_JOB_DELETE",
    "error_details": "Unable to connect to the engine.",
    "warning_message": "Failed to remove local MaskingJob, engineId: 3
localMaskingJobId: 7.",
    "target_id": "d53812ce-9186-485d-a388-44bc52087ead",
    "start_time": "2022-01-02T05:11:24.148000+00:00",
    "update_time": "2022-01-02T06:11:24.148000+00:00"
  }
}
```

9.5 Read-only algorithms

9.5.1 Overview

The DCT 17.0.0 release introduces a new functionality that allows users to explore the masking algorithms imported from a Continuous Compliance Engine.

DCT is equipped with a collection of plugins that support masking algorithms. In DCT version 17.0.0, the shipped algorithm plugin versions range from 1.8.0 to 1.21.0 and from 17.0.0.0 to 22.0.0.0.

When a Continuous Compliance Engine is connected to DCT, DCT imports all algorithms from the connected engine, including built-in algorithms, custom algorithms, and algorithms from any installed custom plugins. Revision hashes of algorithms are also imported to DCT. DCT users can interact with the UI or API to check whether algorithms on different engines are in sync or not.

9.5.1.1 Feature limitations

All algorithms and their revisions are **read-only**. Users cannot create, delete, or modify algorithms, however, tags can be added or removed from the algorithms.

9.5.2 Algorithms page

- **Algorithm list view** – Displays a list of available masking algorithms. Each algorithm entry shows the algorithm name, plugin name, framework name, the source engine name and tags. Algorithm description and revision hash are optional columns that users can show or hide. Clicking an algorithm or the **View** button redirects to its details overview page.
- **Filter and search functionality** – Allows users to filter and search through the list of algorithms based on names, plugin, framework, or tags.

9.5.3 Algorithm details page

- **Algorithm Overview** – Displays the selected algorithm's name and plugin, and shows a brief description of the algorithm and its purpose.
- **Dependencies page** – Available from the algorithm's details page, this will list any files or other algorithms that the selected algorithm depends on.
- **Action buttons** – Buttons for adding/removing tags, with options to change the name and note of revisions (reflecting the read-only nature of the algorithms).

9.6 Data Classes/Domains

Data Classes are known as Domains on the Continuous Compliance Engine, they identify specific types of sensitive data and which algorithms are used to mask it. DCT recreates all Domains discovered through telemetry as Data Classes. This feature allows users to see all the Data Classes on the DCT-registered Continuous Compliance Engines. The Name, Default Algorithm, Source (engine), and Tags columns are shown by default.

> ℹ️ Data Classes are currently read-only, with the exception of updating the description, example, and tags.

This feature allows users to see all the Data Classes on the DCT-registered Continuous Compliance Engines. The Name, Default Algorithm, Source (engine), and Tags columns are shown by default.

There is an additional, optional column for the Default Tokenization algorithm.

9.6.1 Overview tab

This tab displays the metadata for a Data Class.

DCT adds optional Description and Example fields to Data Classes. Domains on Continuous Compliance engines do not have these fields, so although the Data Class feature in DCT is currently read-only, these 2 fields are editable in DCT.

9.6.2 Algorithms tab

This tab lists the algorithms of the Data Class.

9.6.3 Access tab

Standard page for showing users, access groups, and permissions.

9.6.4 API endpoints

- `GET /data-classes` – Retrieve the list of Data Classes
- `POST /data-classes/search` – Search for Data Classes
- `GET /data-classes/{dataClassId}` – Get a Data Class by ID
- `PATCH /data-classes/{dataClassId}` – Update the description and/or example for a Data Class
- `GET /data-classes/{dataClassId}/algorithms` – Retrieve the list of algorithms for a Data Class
- `POST /data-classes/{dataClassId}/algorithms/search` – Search for algorithms of a Data Class
- `GET /data-classes/{dataClassId}/tags` – Get tags for a Data Class
- `POST /data-classes/{dataClassId}/tags` – Create tags for a Data Class
- `POST /data-classes/{dataClassId}/tags/deleteDelete` – Tags for a Data Class

9.7 Masking connectors (read-only)

DCT offers a set of APIs for viewing, updating, tagging, and testing masking connectors; DCT 17.0.0 introduces a new UI for masking connectors that provides visibility on their configuration and metadata, the ability to update and test the connection, and the ability to view compliance jobs related to the connector. Additionally, more metadata fields have been exposed on the connector object to offer details with a higher level of granularity.

9.7.1 Feature limitations

- In DCT 17.0.0, on-the-fly (OTF) jobs in which the connector is a source will not show up in the connector's Jobs tab.
  - In DCT 18.0.0, an **Is On The Fly Masking** column has been added to signify if the job is an OTF job, as well as a **Connector Role** column (which can either be **Source** or **Target**) to represent if the connector being viewed is a source or target to the job.
- The Rulesets tab on a connector details page shows the data elements count, recording it for each rule set. At present, these two columns only have a value if the type is a database connector.

9.7.2 GUI introduction

A new page called **Connectors** is available under the **Compliance** tab from the top navigation bar. This page offers a view of the masking connectors across all connected Continuous Compliance Engines and is updated as connectors are added, removed, or changed on the engine.

The table in this page provides information in columns like connector name, type, platform, host address, source engine, and tags.

There are three potential connector types: **database**, **file**, and **mainframe**. The value under the **Platform** column is different based on the connector type.

- For database connectors, the platform is the database type.
- For file or mainframe connectors, the platform is the connection mode.

See the connector details by clicking on the **View** button in-line for each connector in the list. At the top of this page is an **Actions** menu with two options.

9.7.3 Connectors details page

The top of this page is consistent, showing the name of the connector that was selected and an **Actions** menu with two options.

- **Edit Connector** – Opens a window allowing the user to edit connector configurations, including the password.
- **Test Connection** – Tests the connection using the provided configuration. The result shows up in the form of a Snackbar at the bottom. If it fails, a detailed description of the error appears.

The sub-sections below are named after the tabs found in this page and describe the purpose of each.

9.7.3.1 Overview

This tab lists the details, configurations, and tags related to the connector. The metadata details under the Details tile is based on the connector type (database, file, or mainframe). You can use the Edit button in the Configuration or Tags tiles to modify them as needed.

9.7.3.2 Jobs

This tab shows a list of compliance jobs related to the connector, however, on-the-fly (OTF) jobs that the connector is a source of will not show up in the connector's Jobs tab.

9.7.3.3 Rulesets

This tab shows a list of rulesets related to the selected connector. It includes counts for the data elements and records.

9.7.3.4 Compliance

This tab shows the masking coverage and risk statistics for data of the selected connector.

9.8 Managing classifiers (read-only)

9.8.1 Overview

DCT 19.0.0 introduces a new read-only user interface for users to view metadata and configurations of their classifier objects across all connected engines, in a centralized location.

9.8.2 Accessing the Classifiers UI

1. From the top navigation bar, select **Compliance**.

2. In the side navigation, click on **Data Classifiers**.

3. Select the **Classifiers** tab.

This page provides an overview of all classifiers across all engines.

9.8.3 Classifiers page

The **Classifiers** page displays a paginated list of classifiers. Each entry in the list includes details such as:

- Classifier Name
- Framework
- Data Class Assignment
- Other relevant metadata

9.8.3.1 Classifier details page

Each classifier in the list includes a **View** button, which opens a dedicated details page. This page provides a higher level of granularity on the classifier's metadata and configuration.

The details page contains three main tabs:

1. **Overview**
   - Displays general metadata about the classifier.

2. **Configuration**
   - Shows the configuration for the classifier.
   - The configuration details vary based on the classifier's framework.
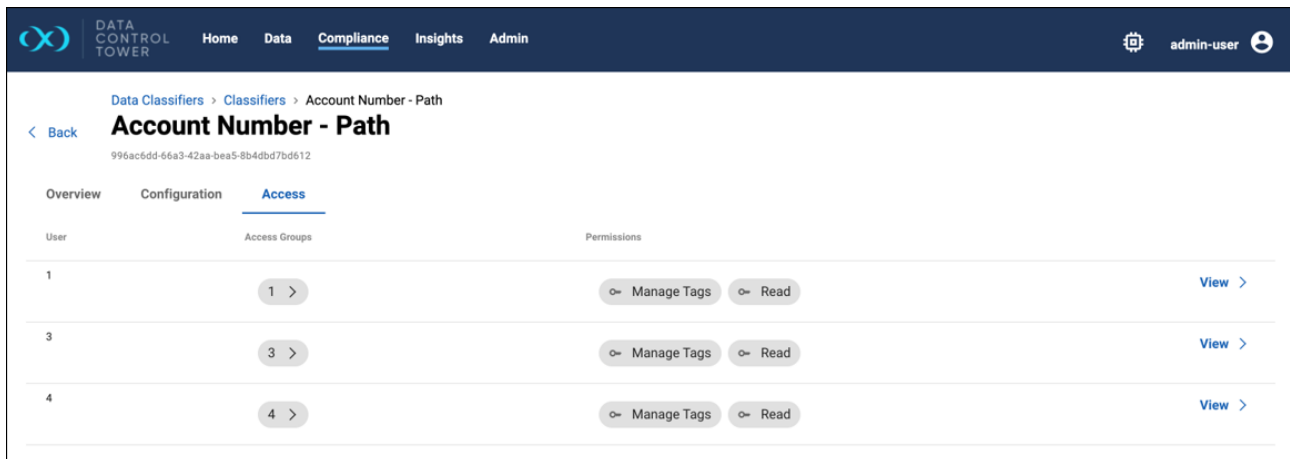   - Examples of configurations for each of the four supported classifier frameworks are provided.

3. **Access**
   - Displays information about users, access groups, and permissions related to the classifier.

9.8.3.2 Tag management

While the experience is primarily read-only, users have the ability to add or remove tags for each classifier.

9.8.4 API Support

Several new APIs have been introduced to support the classifier feature:

- `GET /classifiers` – Retrieve the list of classifiers.
- `POST /classifiers/search` – Search classifiers based on provided filter expressions.
- `GET /classifiers/{classifierId}` – Retrieve a classifier by its ID or name.
- `GET /classifiers/{classifierId}/tags` – Retrieve the list of tags associated with a classifier.
- `POST /classifiers/{classifierId}/tags` – Create new tags for a classifier.
- `POST /classifiers/{classifierId}/tags/delete` – Delete tags from a classifier.

By using these APIs, users can programmatically interact with the classifiers, facilitating integration with other systems and automation processes.

9.9 Discovery policies (read-only)

9.9.1 Overview

DCT 20.0.0 introduces a read-only experience for discovery policies (formerly known as profiling sets on the engine). This feature allows users to view their discovery policies from a central location across all connected engines.

A new top-level page of the same name, **Discovery Policies**, has been created to for this centralized view. Users can access detailed information about each displayed discovery policy, including metadata and relationships with other objects, such as referenced classifiers or expressions.

Additionally, the Classifiers UI has been enhanced so users can now view a list of discovery policies that reference a specific classifier on the classifier's details page.
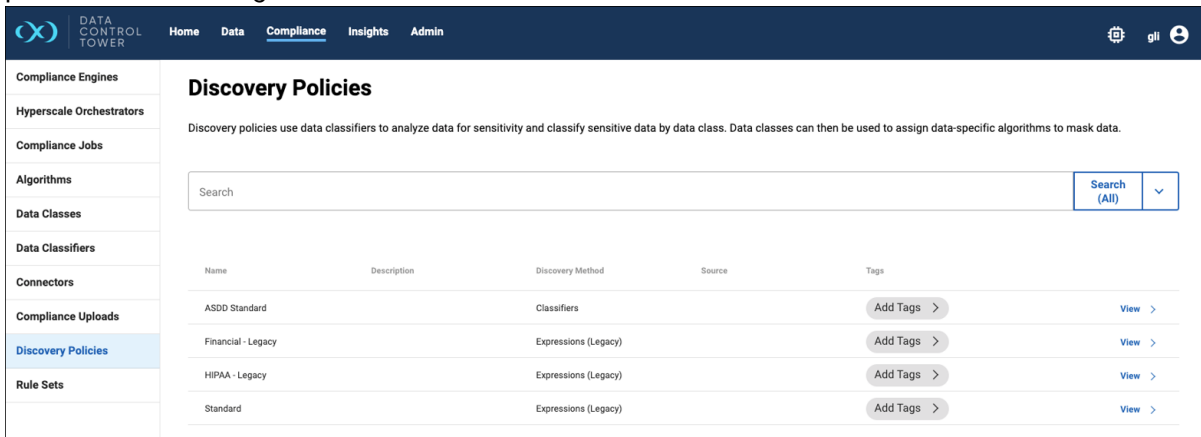
> ℹ️ No editing abilities are supported, with the exception of adding or removing tags on a discovery policy.

9.9.1.1 Limitations for discovery expressions

- Users can view the list of referenced discovery policies for classifiers but not for expressions, due largely to the fact that discovery expressions are considered legacy objects and are discouraged in favor of classifiers. As a result, only a minimal number of features are provided for expressions.

9.9.2 GUI access

- Navigate to the **Compliance** section in the top navigation menu.
- Select **Discovery Policies** from the side navigation menu, which provides an overview of all discovery policies across all engines.


- Each discovery policy on the top-level view includes a **View** button that redirects users to the policy's details page. The **Overview** tab displays a list of metadata.
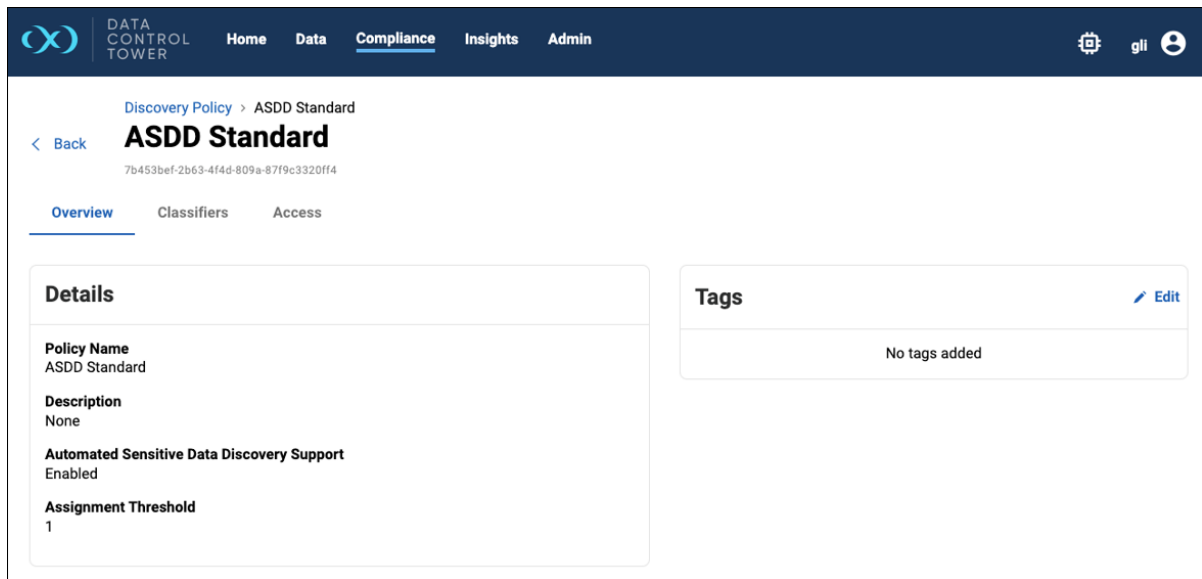
- If a discovery policy uses classifiers as its discovery method, a **Classifiers** tab will be present, listing the referenced classifiers.
    - The classifier details page now includes a new tab titled **Discovery Policies**, which shows a list of policies referencing the classifier object.
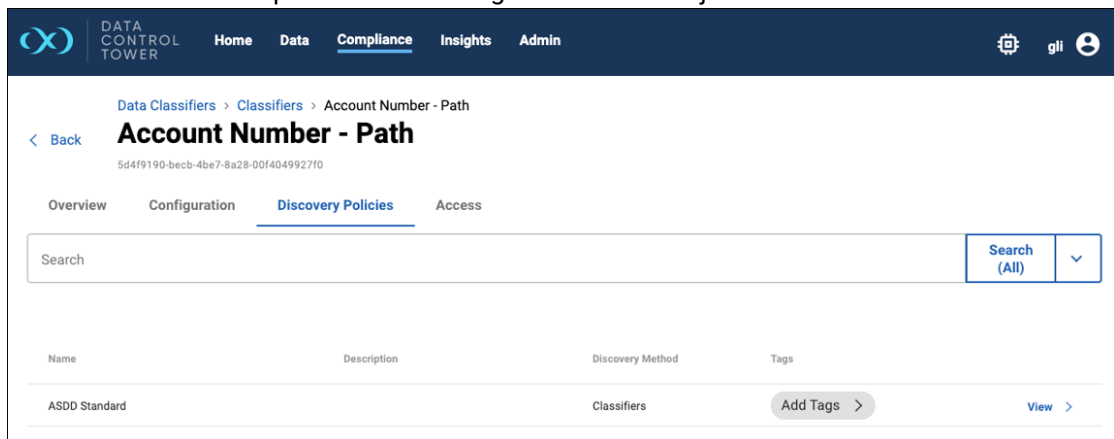
- If a discovery policy uses expressions as its discovery method, an **Expressions** tab will be present, listing the referenced discovery expressions.
- An **Access** tab on the discovery policy details page displays users, access groups, and permissions related to the policy.

9.9.3 API endpoints

The following APIs have been introduced:

- `GET /discovery-policies`
  List all discovery policies.
- `POST /discovery-policies/search`
  Search for discovery policies.
- `GET /discovery-policies/{discoveryPolicyId}`
  Retrieve a discovery policy by ID or name.
- `GET /discovery-policies/{discoveryPolicyId}/tags`
  Get tags associated with a discovery policy.
- `POST /discovery-policies/{discoveryPolicyId}/tags`
  Create tags for a discovery policy.
- `POST /discovery-policies/{discoveryPolicyId}/tags/delete`
  Delete tags from a discovery policy.
- `GET /discovery-policies/{discoveryPolicyId}/classifiers`
  List classifiers associated with a discovery policy.
- `POST /discovery-policies/{discoveryPolicyId}/classifiers/search`
  Search for classifiers related to a discovery policy.
- `GET /discovery-policies/{discoveryPolicyId}/expressions`
  List discovery expressions associated with a discovery policy.
- `POST /discovery-policies/{discoveryPolicyId}/expressions/search`
  Search for discovery expressions related to a discovery policy.
- `GET /classifiers/{classifierId}/discovery-policies`
  List discovery policies associated with a classifier.

- `POST /classifiers/{classifierId}/discovery-policies/search`
  Search for discovery policies related to a classifier.

9.10 Discovery expressions (read-only)

9.10.1 Introduction

In DCT 18, a new read-only page has been introduced for listing discovery expressions, presented as *expressions* within the Continuous Compliance Engine. There are two types of expressions on the engine:

- **Search expressions** – Targeted at finding specific data within data sources.
- **Type expressions** – Used to identify types of data based on schema information.

In DCT, both types of expressions are merged into a single list with an additional column named **Expression Type** to distinguish between them. Despite this primarily being a read-only feature, users are able to add tags to an expression.

> ℹ️ Expressions cannot be assigned to any discovery policies (profile sets).

9.10.1.1 Level column

An enumerative column named **Level** has been introduced in DCT. This column indicates the target level of the expression within a data source:

- **Type:** If the expression is a type expression.

- **Data:** If the expression is a search expression and the `dataLevelProfiling` attribute is `true`. This indicates that the expression targets underlying data (e.g., row values in a database column or field values in a file).
- **Column:** If the expression is a search expression and the `dataLevelProfiling` attribute is `false`. This indicates that the expression targets the schema of the data source (e.g., column names in a database table or field names in a file).

9.10.2 Accessing the Discovery Expressions page

To view the Discovery Expressions page:

1. From the top navigation, select **Compliance**.
2. In the side navigation, click on **Data Classifiers**.
3. At the top of the list, select **Expressions (Legacy)**.

This page provides an overview of all discovery expressions with their respective types and levels.

9.11 Application settings (read-only)

9.11.1 Introduction

DCT 21.0.0 introduces a new UI page and API endpoints that provide a read-only experience for Continuous Compliance Engine application settings. This enables users to view their application settings for each connected compliance engine.

9.11.2 GUI

Select the Compliance tab and click into a compliance engine's details page from the list. In the details page, a new tab has been created for Application Settings.

9.11.3 API

These following APIs were added for this feature:

- `GET /management/engines/{engineId}/compliance-application-settings`
  - Gets a compliance engine's application settings.
- `POST /management/engines/{engineId}/compliance-application-settings/search`
  - Search a compliance engine's application settings.

10 Hyperscale Orchestrator UI

10.1 Overview

The Hyperscale UI is *Controlled Availability*, meaning that Delphix highly encourages interested customers to reach out to Delphix Product and Engineering for a guided onboarding. Please contact your account team if interested.

To disambiguate object nuances between Compliance Engines, Hyperscale Orchestrators, and DCT, please refer to the following definitions:

- **Hyperscale dataset**
    - A Hyperscale dataset is a granular unit that defines source and target connector information alongside a defined database schema to be masked (the inventory on a Compliance Engine).
- **Hyperscale job**
    - A Hyperscale-specific object that is comprised of a rule set along with additional running parameters such as engines to be used and memory settings.
- **Hyperscale job execution**
    - The execution information such as duration, rows masked, etc. that is sent back to DCT for reporting.

Data Control Tower has the ability to connect to one or more [Hyperscale Compliance Orchestrators](#)[123], to serve as a complimentary user interface. The current Compliance Engine/Hyperscale Orchestrator/DCT workflow is as follows:

1. Build the foundational objects (rule sets, connectors, inventories, etc.) within the Compliance Engine.

2. Build a data set on the Hyperscale cluster by either:
    a. Importing a compliance job from a Compliance Engine.
    b. Building the dataset via the connector, dataset, and jobs API on the Hyperscale cluster (DCT will read the new Hyperscale jobs and display them.)

3. Set up the DCT-to-Hyperscale connection by following simple steps in the [Connecting/ authenticating](#)[124] page. DCT will then create references for any data sets (registered as Hyperscale jobs in DCT), cluster details (e.g. mount points, etc.), and pre-existing engine connections to the Hyperscale Orchestrator.

4. Create a Hyperscale job on DCT by selecting an existing masking job and importing the job's inventory and connector information into DCT to represent a hyperscale job. Users have the option to further refine the job details such as defining table-specific masking parameters. A job will appear as a special Hyperscale job on the compliance jobs page.

5. Execute a Hyperscale job by selecting the **run** option on the Hyperscale jobs detailed view.

---

123 https://hyperscalemasking.delphix.com/docs/latest/
124 https://dct.delphix.com/docs/10.0.0/connecting-authenticating

6. DCT will recognize this job execution and will present the execution details under the specific Hyperscale Orchestrator's **Details** page.

10.2 Implementation introduction

The first step in working with Hyperscale is to register a Hyperscale Orchestrator with DCT. DCT connects to all Hyperscale Orchestrators over HTTPS, some configurations might be required to ensure DCT can communicate successfully.

Begin the Hyperscale Orchestrator registration flow, as well as view currently registered Hyperscale Orchestrators under the **Compliance** -> **Hyperscale Orchestrators** section. The registration wizard will guide you through the steps of the process, some of which are described in further detail below.

10.2.1 Truststore for HTTPS

If the CA certificate that signed the Hyperscale Orchestrator's HTTPS certificate is not a trusted root CA certificate present in the JDK, then custom CA certificates can be provided to DCT. If these certificates are not provided, a secure HTTPS connection cannot be established and registering the Hyperscale Orchestrator will fail. The `insecure_ssl` Hyperscale Orchestrator registration parameter available as a checkbox option in the registration wizard can be used to bypass the check, however, this should not be used unless the risks are understood.

Get the public certificate of the CA that signed the Hyperscale Orchestrator's HTTPS certificate in PEM format. Your IT team might be required to get the correct certificates. Base64 encode the certificate with:

```
cat mycertfile.pem | base64 -w 0
```

Copy the Base64 encoded value from the previous step and configure in the values.yaml file under the `truststoreCertificates` section. For example, the section might look like this:

```
truststoreCertificates: <certificate_name>.crt: <base64 encode certificate string
value in single line>
```

`<certificate_name>` can be any logically valid string value, like "hyperscale".

All certificates configured in the `truststoreCertificates` section will be read and included in the trustStore, which would then be used for SSL/TLS communication between DCT and Hyperscale.

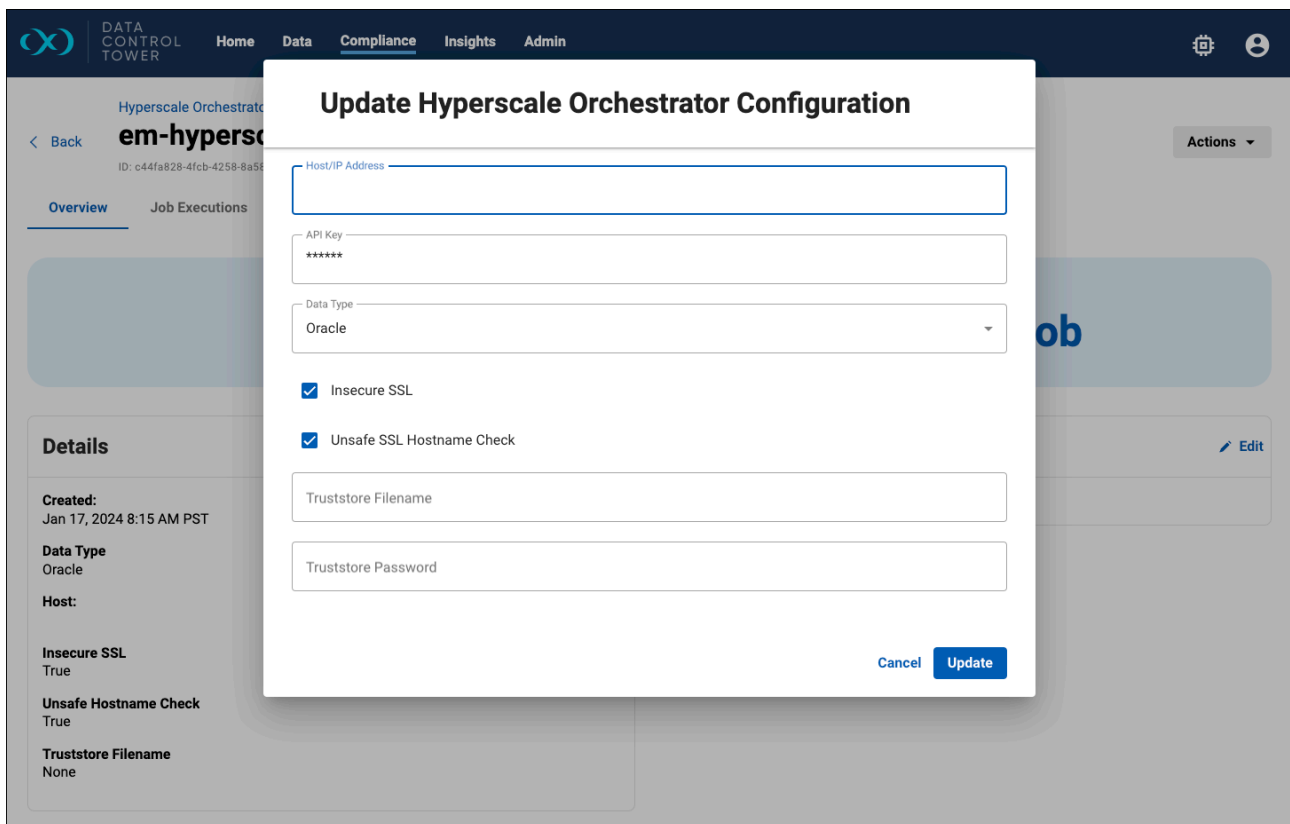## 10.2.2 Authentication with Hyperscale Orchestrators

All authentication with a Hyperscale Orchestrator is done with an API Key corresponding to an admin Hyperscale user. The API will be stored and encrypted on DCT itself. The use of a vault to store Hyperscale credentials is currently not supported.

10.2.3 Hyperscale deployment type

A deployment type designation must be made for a Hyperscale Orchestrator at the time of registration. The registration UI wizard will present the data type options that are supported. This type must be in line with the type of data source the Hyperscale Orchestrator has been set up to mask (eg, Oracle).

10.2.4 Editing and unregistering Hyperscale Orchestrators

Making edits to a Hyperscale Orchestrator's configuration can be done via the UI, in the Orchestrator's overview page via the Edit action in the Details tile.

Unregistering a Hyperscale Orchestrator and removing all its data from DCT can be done via the `DELETE` API or directly in the UI via the Actions menu.

10.3 Managing Hyperscale objects

10.3.1 Introduction

After registering a Hyperscale Orchestrator, DCT will begin to ingest all relevant data discovered on that Hyperscale instance. This data, in turn, becomes modeled in the DCT world with APIs and a UI to (and manage to a limited extent). Currently, a lot of this data can be navigated to and viewed from the main overview details page of a Hyperscale Orchestrator.

10.3.2 Hyperscale executions

A view of all current and past Hyperscale job executions present on a particular Hyperscale Orchestrator can be found under the Job Executions tab. The **View** link will display additional information, especially for in-progress or failed executions.

10.3.3 Hyperscale jobs

A view of all Hyperscale jobs discovered on a Hyperscale Orchestrator can be found under the **Associated Jobs** tab.

The **View** link will take you to the **Compliance Jobs** details page where more information about the job can be found.

> Hyperscale jobs are not separate entities in DCT, but rather are combined with standard Compliance jobs. Hence, this Hyperscale job details page can be navigated to from **Compliance** -> **Compliance Jobs**. There is a `type` attribute (Hyperscale or standard) which helps differentiate between the different types.

Configuration and dataset details can be edited directly in the job's Overview page.

10.3.4 Hyperscale job engine selection

The Engine Selection tab allows you to manage the compliance engines that the job can use for masking. The table will list **all** compliance engines that are part of the Hyperscale Orchestrator's engine pool. Each compliance engine will either be Selected or Not Selected for the job. The state of each engine is indicated by the Selection column and can be changed via the Actions menu.
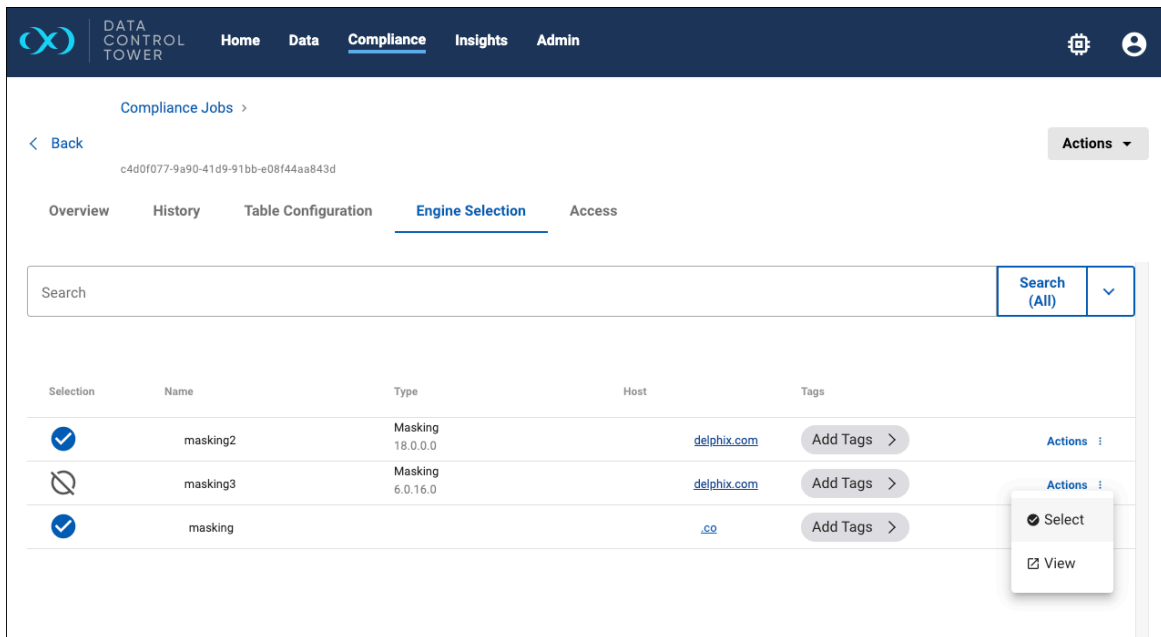
10.3.5 Hyperscale job table configuration

The Table Configuration tab contains the early stages of inventory management. You can see the Hyperscale dataset tables and their settings in the screenshot below. The left side shows the list of tables and allows you to search and paginate through the results.
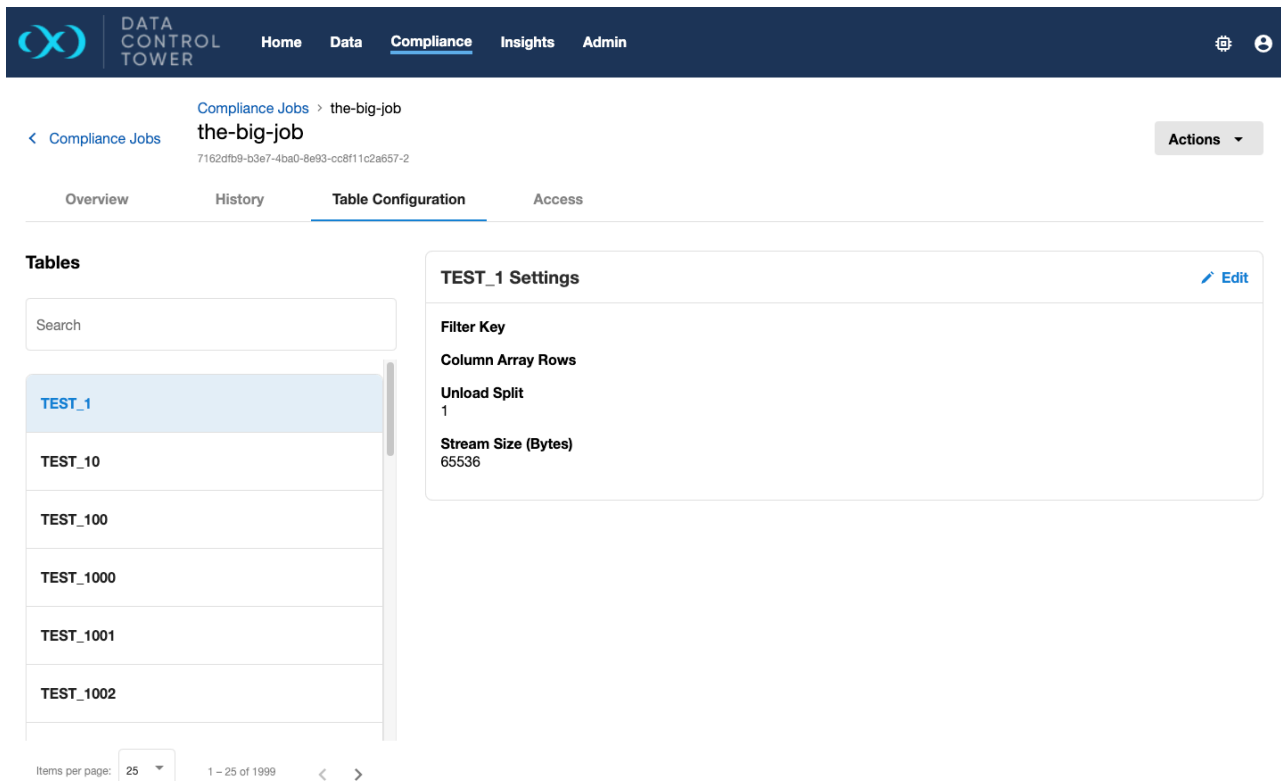
The right side shows you the settings and allows you to make changes via the **Edit** action.
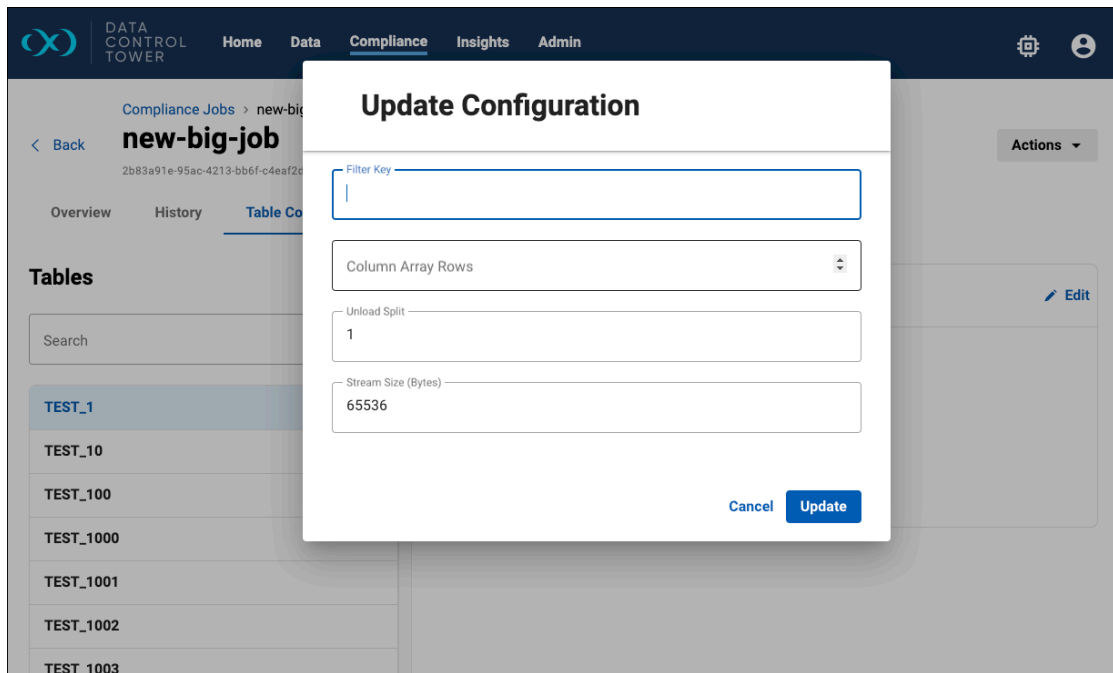
10.3.6 Executing Hyperscale jobs

A Hyperscale job can be executed via the Actions menu.
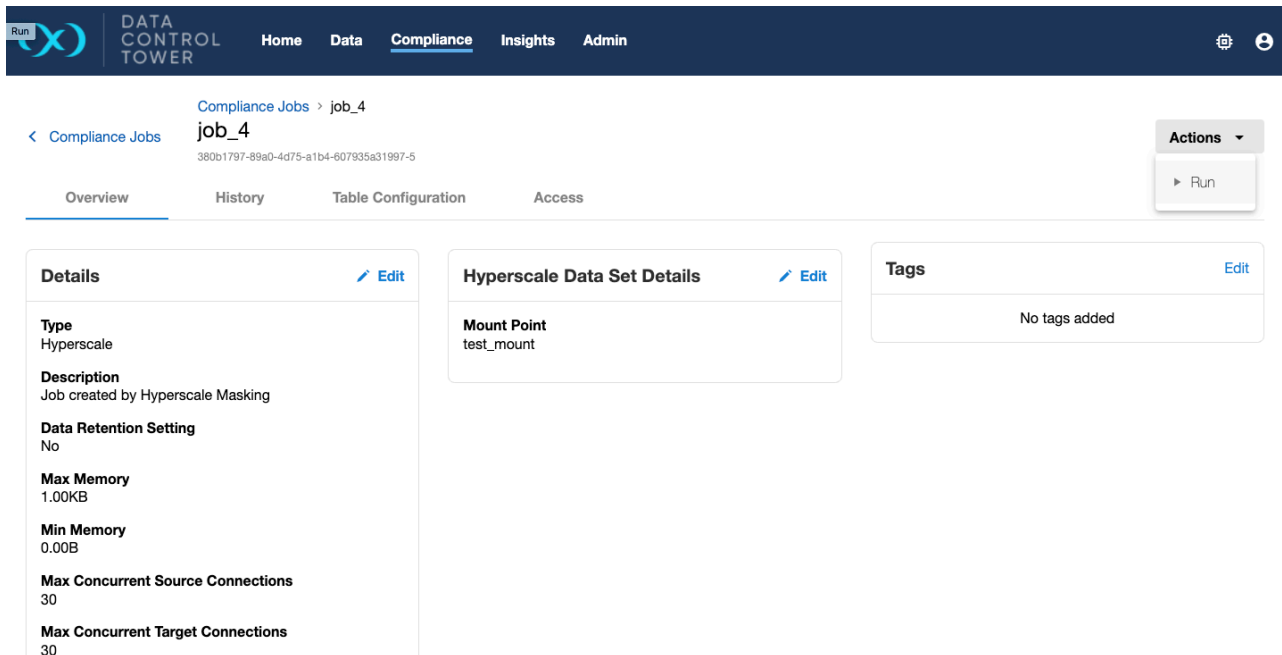
Once a job has started, a progress bar with details will appear in the Overview page.

Click the **View Job Execution Details** link to go to the Execution Details page to see job execution progress, as well as an Actions menu to **Stop** or **Re-Run** a job.

Details of a failed job will also be displayed in the Execution Details page. If the job was configured to retain execution data, the Clean option in the **Actions** menu can be used to discard this data.

Data Control Tower – Data Control Tower Home

10.3.7 Creating Hyperscale jobs

New Hyperscale jobs can be created via DCT when you have a database compliance job setup on a Compliance Engine and need to use the same masking inventory in a Hyperscale job. DCT will export the masking job details from the Compliance Engine and import them into the Hyperscale Orchestrator. The result is a new set of Hyperscale connectors, a dataset, and a Hyperscale job.

> ℹ️ Hyperscale job creation is currently only supported on Hyperscale Orchestrators of type ORACLE and MSSQL.

The **+ Hyperscale Job** button in the Actions menu will initiate a wizard that walks you through the creation process.

After setting the new job name, description, and tags, the wizard will prompt you to select a **source** compliance job. This compliance job must be the source job on your compliance engine whose inventory you

want to import. You must then explicitly select a Hyperscale Orchestrator to create the job on, along with the **MountPoint** to use. Finally, set any and all configuration settings that are relevant to the job you want to create.

By default, DCT will create the Hyperscale job using all available Compliance Engines in the orchestrator's pool. If changing this selection is required, it can be done after job creation in the Engine Selection tab of the job's details page.

Currently DCT does not validate or enforce required input parameters. The requirements to create a Hyperscale job will differ depending on the Hyperscale Orchestrator version and database type.

As a reminder, the import process will not include any secure credentials for connectors. You have the option to set the connector credentials up front when creating the job (see screenshot above). Otherwise, you must independently find the imported connectors and explicitly set the credentials after the job has been created.

If your source compliance job is using any global objects such as Algorithms, those global objects must be pre-synced to all engines in the orchestrator's engine pool. This can be done via the **Manually Deploy Job Dependencies** option in the Hyperscale Orchestrator's **Action** menu.

The wizard will walk you through the necessary steps which includes selecting the source compliance engine along with one or more target engines.

Please refer to the Hyperscale product documentation[125] for the Hyperscale Orchestrator version you are running to learn all about the specifics of creating jobs in Hyperscale.

---

125 https://documentation.delphix.com/docs/hyperscale-compliance

Updates to the original source compliance job can be done anytime after initial import and a re-import can be done via DCT. This will update the Hyperscale job's existing dataset with the refreshed ruleset from the source compliance job.
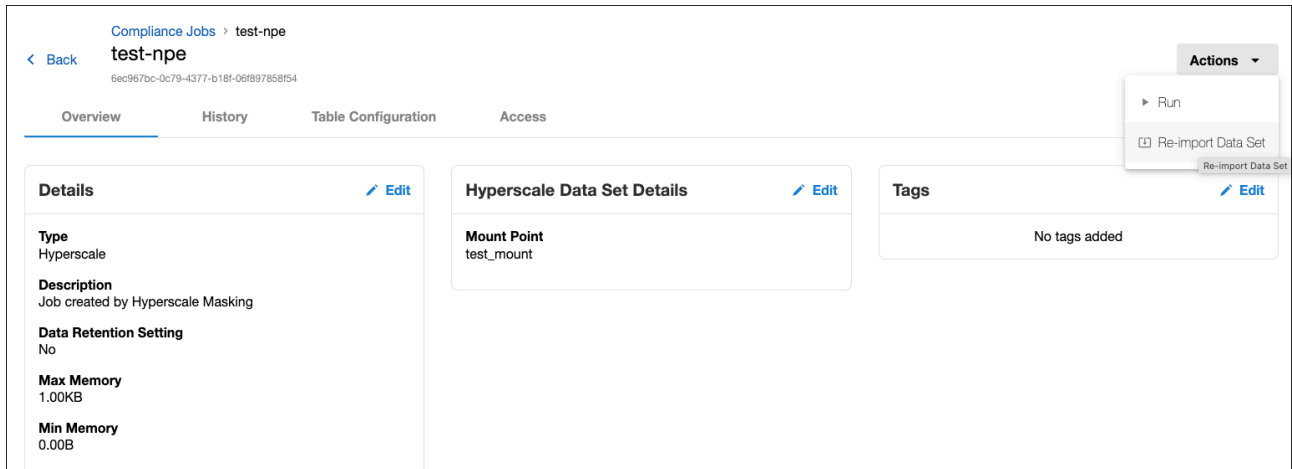
Note that this action will keep existing table configurations intact and the initial default settings for **unload split** and **stream size** will apply only to new tables. These settings can be changed for any table afterwards in the job's **Table Configuration** details.

10.3.8 Hyperscale Compliance Engines

A view of all Compliance Engines registered with a Hyperscale Orchestrator can be found under the **Engine Pool** tab.
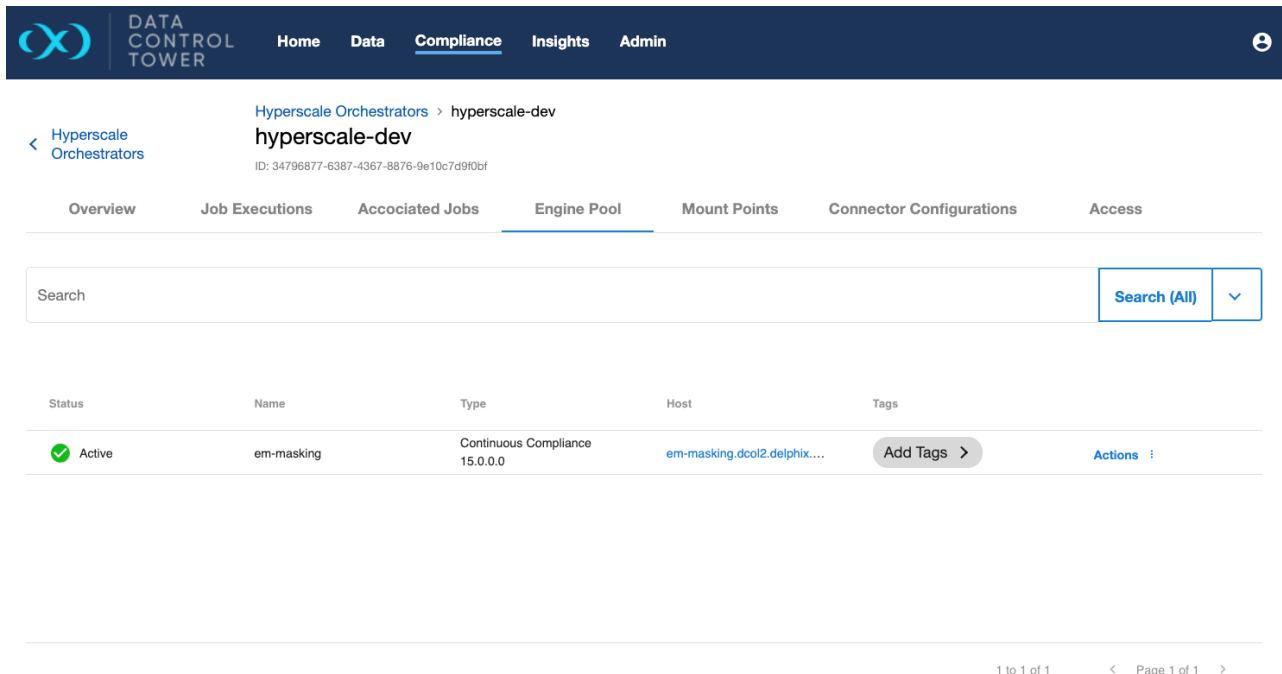
When DCT discovers the engines registered with a Hyperscale Orchestrator, it will create DCT `RegisteredEngine` entities out of them (if they do not already exist in DCT, as uniquely identified by the hostname). The result is a unified model where DCT `RegisteredEngine` objects are what make up the

engine pool in a Hyperscale Orchestrator. The same engines will appear under the **Compliance** -> **Compliance Engines** page.

[≡] Engine credentials will not be retrieved from the Hyperscale Orchestrator, so the created engines must be updated with credentials, along with any relevant security settings. The discovered engines will remain in the OFFLINE status until updated.

[≡] The created DCT engine entities are permanent, in the sense that even if the origin Hyperscale Orchestrator is unregistered, the engines in DCT remain as if they had been registered independently of Hyperscale.

Adding new engines to the engine pool can be done by clicking the **+ Engine** button. Only Compliance Engines that have already been registered with DCT can be added to a Hyperscale Orchestrator's engine pool:
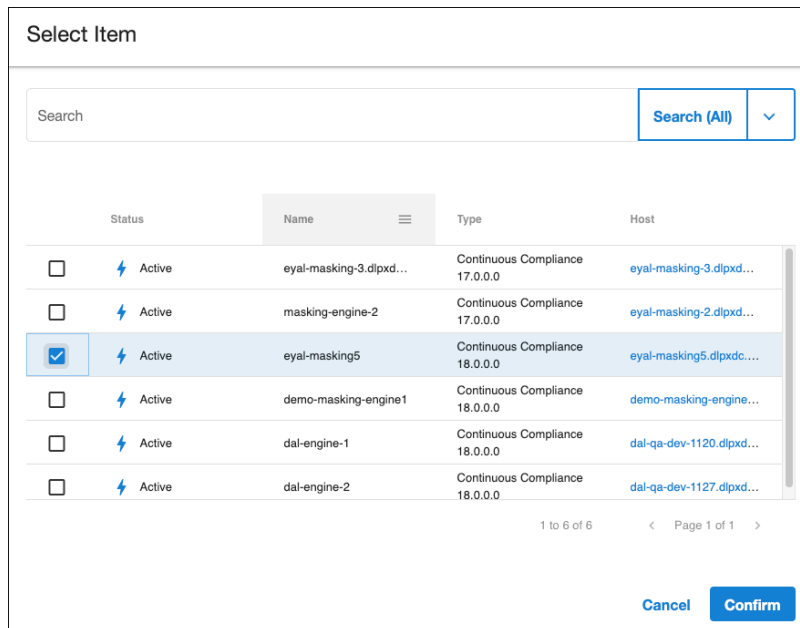
Note that this operation will, in turn, register the Compliance Engine with the Hyperscale Orchestrator. The Compliance Engine name, credentials, and configuration settings will be set according to DCT's record.

Removing an engine from the engine pool can be initiated with the **Remove** button via the Actions menu:
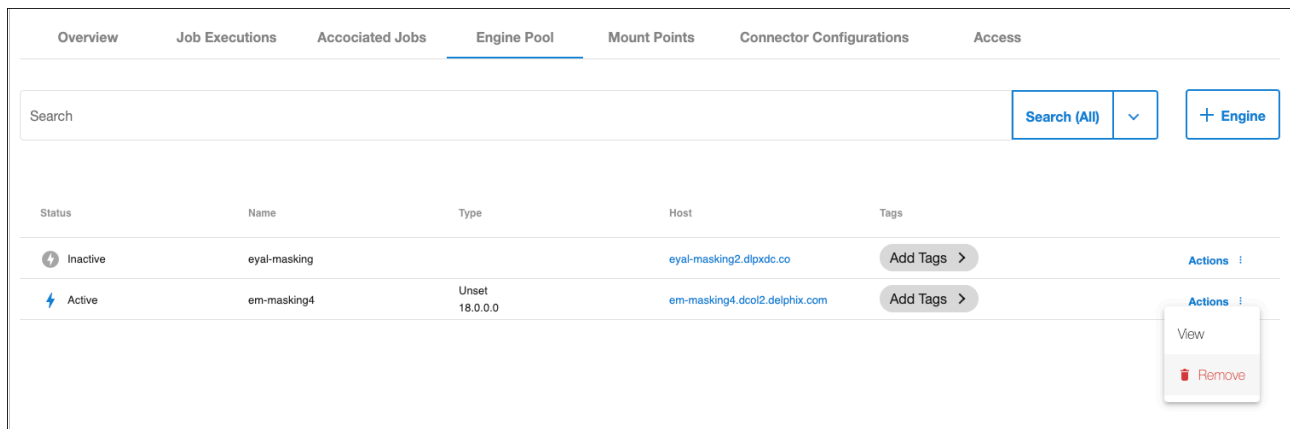
This will completely unregister the Compliance Engine from the Hyperscale Orchestrator.

10.3.9 Hyperscale mount points

A view of all mount points on a Hyperscale Orchestrator can be found under the **Mount Points** tab.
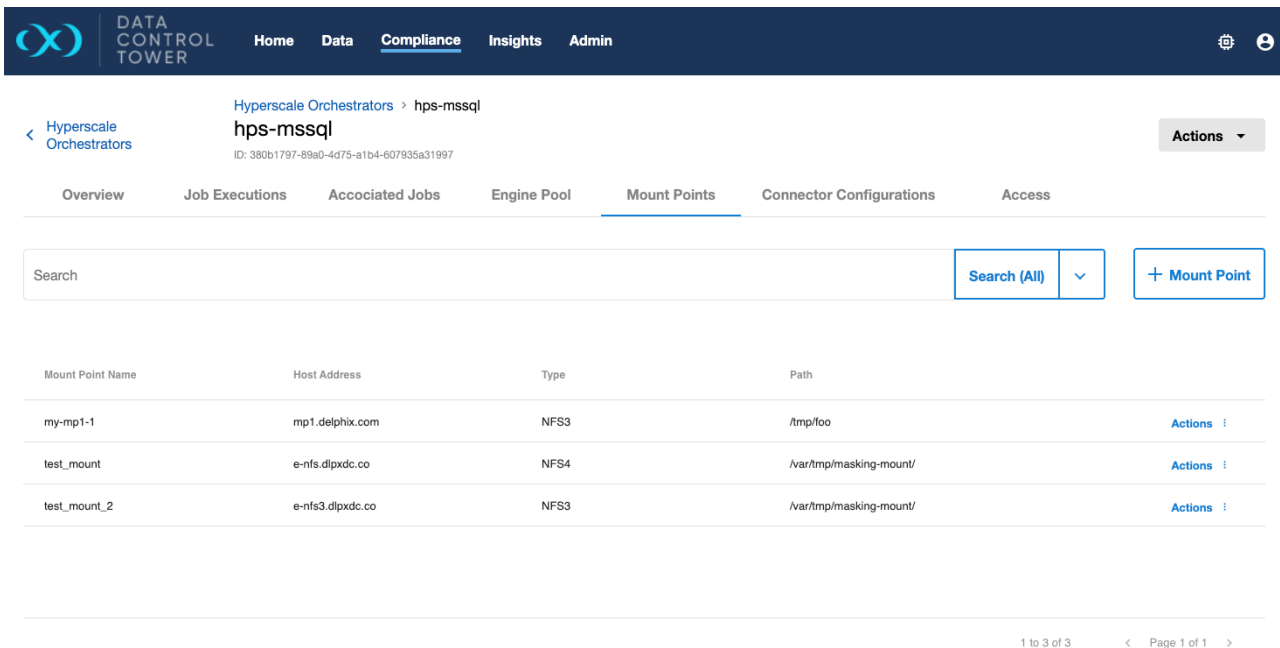
Creating new mount points can be done via the **+ Mount Point** button.
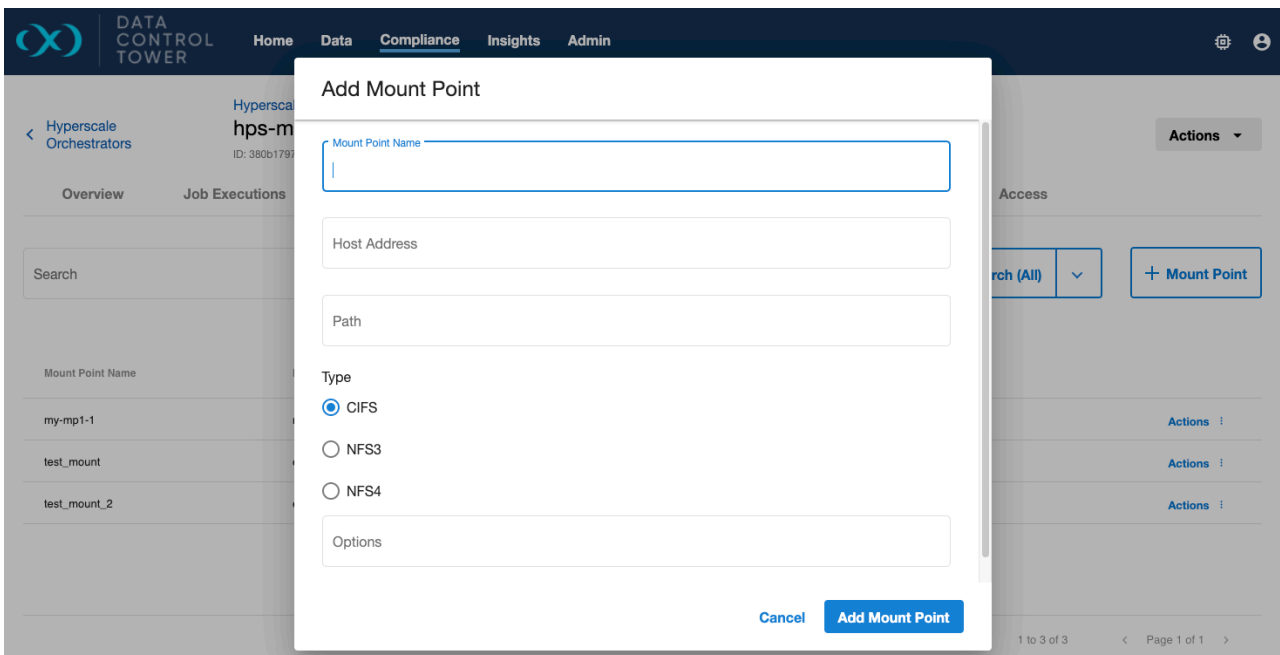
This will result in a new mount point being created directly on the Hyperscale Orchestrator.

Editing and **Deleting** a mount point can be done for a particular row via the Action menu in the last column of the table.

10.3.10 Hyperscale connector configurations

A view of all connectors on a Hyperscale Orchestrator can be found under the **Connector Configurations** tab.

Creating new connectors can be done via the **+ Connector** button.
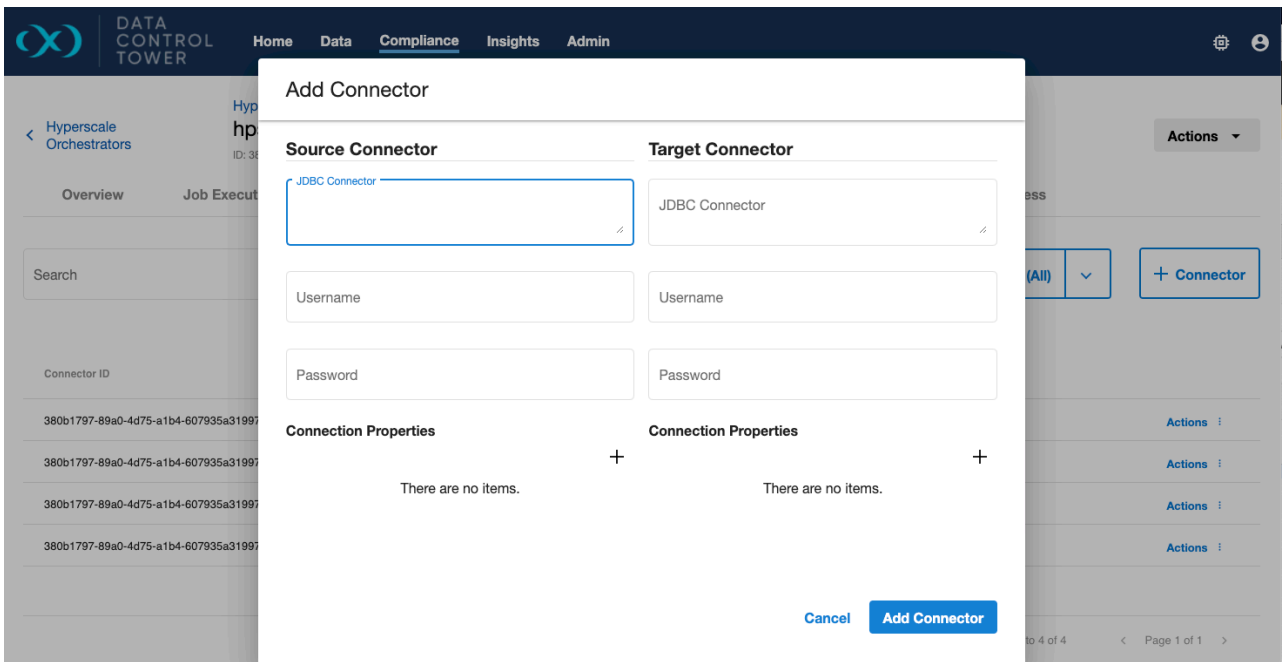
Editing and **Deleting** a connector can be done for a particular row via the Actions menu in the last column of the table.
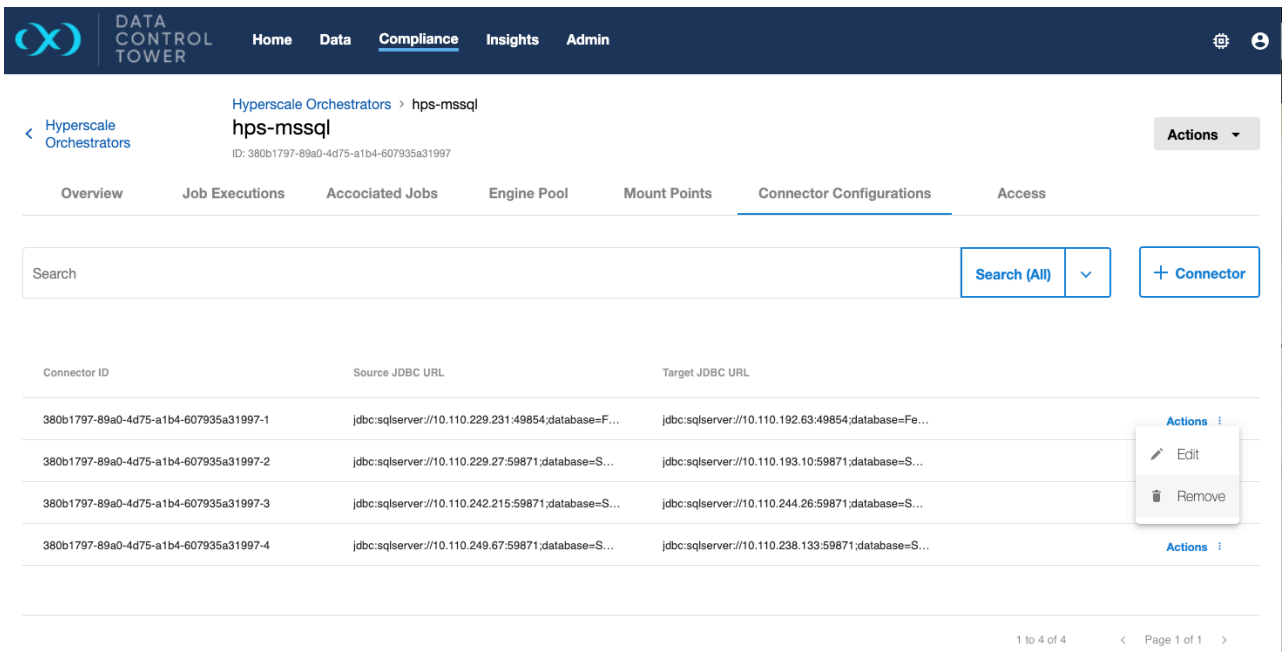
11 Integrations

Data Control Tower provides a global integration layer for a connected Delphix ecosystem, whether that is a single or dozens of globally distributed engines, DCT drives a scalable approach to integrating Delphix into any custom script or automation toolchain.

Aside from the comprehensive API layer (see API references _(see page 380) for more detail), DCT powers automation through Delphix-built and supported integrations with popular applications such as Terraform, ServiceNow, etc.

To see a current list of Delphix integrations, please visit Delphix Integrations[126] for more detail.

---

[126] http://ecosystem.delphix.com

12 DCT concepts

12.1 Introduction

Data Control Tower (DCT) provides new and novel approaches to general Delphix workflows, delivering a more streamlined developer experience. This article will introduce these concepts to Delphix and how they work with DCT.

> For **VDB Provisioning**, the UI supports these data platforms:
>
> - Oracle Single Instance Single Tenant
>
> - Oracle Single Instance Multi Tenant (for Linked CDB only)
>
> - MSSql Single Instance
>
> For **Infrastructure Connection Wizard**, only UNIX standalone environments can be added via the UI.

12.2 Concepts

12.2.1 Virtual Database (VDB) groups

Virtual Database (VDB) groups are a new concept to Delphix, which enable the association of one or more VDBs as a single VDB group. This allows for bulk operations to be performed on the grouped VDBs, such as bookmark, provision, refresh, rewind, and others. This will assist in complex application testing scenarios (e.g. integration and functional testing) that require multiple data sources to properly complete testing.

With VDB groups, developers can now maintain data synchronicity between all grouped VDBs, which is particularly useful for complex timeflow operations. For example, updating VDBs to reflect a series of schema changes across data sources, or to reflect an interesting event in all grouped datasets. In order to maintain synchronicity among grouped datasets, timeflow operations (refresh, rewind, etc.) must use a bookmark reference.
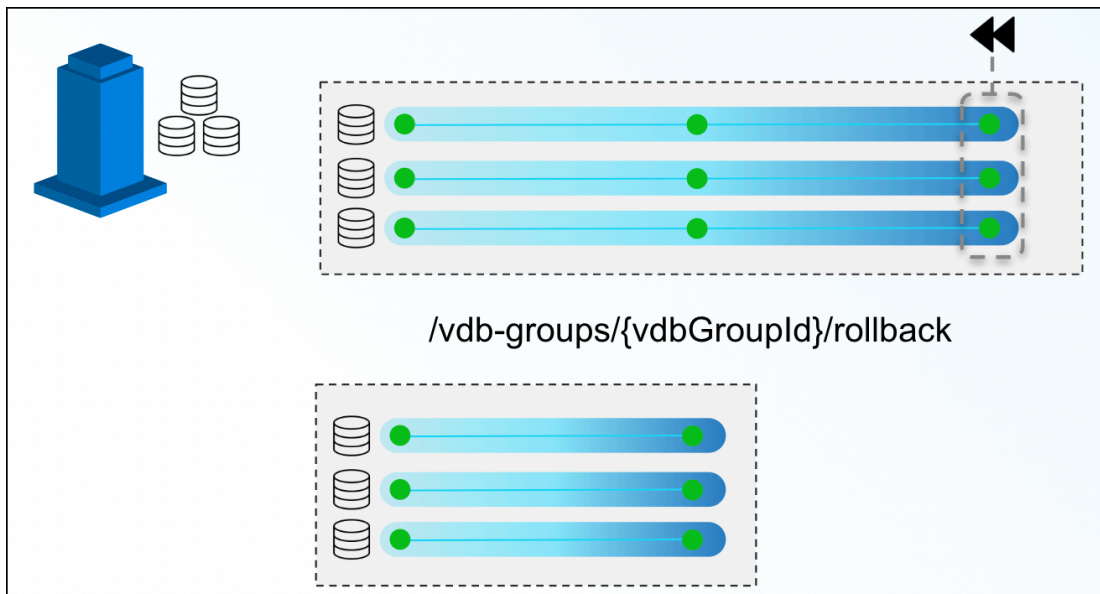
In the above example, a VDB Group reference is created for three VDBs. At the end of the above timeline group, a developer decides to rollback those VDBs to a previous snapshot. By issuing a single command via the VDB groups endpoint, DCT will move all three back, ensuring that they all maintain referential synchronicity.

Bookmarks and VDB groups are loosely related; a VDB group can exist in the absence of any bookmarks, and a bookmark can exist without any VDB group. It is important to note that the bookmark represents data, while the VDB group represents the databases to make this data available.

> DCT will automatically stop an operation from executing if one or more objects are incompatible (e.g. provisioning a VDB group into a set of environments, where one of the VDBs is incompatible, such as an Oracle on Linux VDB provisioned onto a Windows environment).
>
> VDB groups based operations will return a single job to monitor the overall status of the series of individual VDB operations. If one of those individual operations is unable to complete, DCT will report a "fail", but any individual operations that are able to successfully complete will still do so.

12.2.2 Comparing Self-Service containers to VDB groups

As mentioned above, VDB groups are a crucial DCT concept that enable Self-Service functionality outside of the Self-Service application. Consider VDB groups acting similarly to Self-Service containers, in that it provides grouping and synchronization among VDBs, but VDB groups can provide a more flexible approach for users. Here are some additional points for example:

- The same VDB can be included in multiple VDB groups
- Including a VDB in a VDB group does not prevent operations on the VDB individually
- VDBs can be added to or removed from VDB groups
- VDB groups do not have their own timeline

12.2.3 Bookmarks

DCT Bookmarks are a new concept that represents a human-readable snapshot reference that is maintained within DCT. This is not to be confused with Self-Service bookmarks, maintained separately within the Self-Service application. With DCT Bookmarks, developers can now reference meaningful data (e.g. capturing a schema version reference to pair with an associated code version, capturing test failure data so that developers can reproduce the error in a developer environment, etc.) and use those references for any number of use-cases (e.g. versioning data as code, quickly provisioning a break/fix environment with relevant data, etc.). DCT Bookmarks are compatible with both VDBs and VDB groups, and can be used as a reference for common timeflow operations such as:

- Provisioning a VDB or VDB group from a bookmark
- Refreshing a VDB or VDB group to a bookmark
- Rewinding a VDB or VDB group to a bookmark

> ℹ️ DCT Bookmarks have associated retention policies, the default value is 30 days, but policies can be customized anywhere from a day to an infinite amount of time. Once the Bookmark expires, DCT will delete the bookmark.
>
> Bookmarks are compatible with individual VDBs and VDB groups. Bookmark Sharing is only available for engines on version 6.0.13 and above.
>
> DCT Bookmarks, when created, initiate a snapshot operation on each and every VDB in order to maintain synchronicity between each VDB. In that same vein, bookmark-based VDB group operations will have each VDB-specific sub-process run in parallel (as opposed to sequentially) to reduce drift between grouped VDBs.

12.2.4 Jobs

Jobs in DCT are the primary means of providing operation feedback (PENDING, STARTED, TIMEDOUT, RUNNING, CANCELED, FAILED, SUSPENDED, WAITING, COMPLETED, ABANDONED) for top-level operations that are run on DCT. Top-level operations represent the parent operation that may have one or more child-based jobs (e.g. refreshing a VDB group is the parent job to all of the individual refresh jobs for the grouped VDBs under the VDB group reference).

> 🗒️ Top-level jobs will report a "FAILED" status if one or more child jobs fail. For child jobs that can complete, DCT will continue to complete those jobs even if a parent job reports a failure.

12.2.5 Tags

DCT Tags enable a new business metadata layer for users and consumers to filter, sort, and identify common Delphix objects, to power any number of business-driven workflows. A tag is comprised of a (Key:Value) pair that associates business-level data (e.g. location, application, owner, etc.) with supported objects. DCT 2.0 and above support the following Tags:

- Continuous Data Engines
- Environments
- dSources
- VDBs

Developers and administrators add and remove tags using tag-specific object endpoints (e.g. `/vdbs/{vdbId}/tags`) and can leverage tags as search criteria when using the object-specific search endpoints (e.g. using filtering language to narrow results).

Some sample tag-based use-cases include:

- Refreshing all the VDBs owned by a specific App Team using an "Application: Payment Processing" tag. This would be accomplished by querying "what VDBs have the (Application: Payment Processing) tag" and feeding those VDB IDs into the refresh endpoint.
- Driving accountability for VDB ownership by tagging primary and secondary owners for each VDB (e.g. (primary_owner: John Smith), (secondary_owner: Jane Brown)). That way, if a VDB is overdue for a refresh, tracking down an owner is a simple tag query.

> Tags are registered as an attribute that is specific to an object as opposed to a central tagging service. As a result, tag-based querying can only be done on a per-object type basis.
>
> A supported object can contain any number of tags.

12.2.6 Tag-based filtering

All taggable objects support tag-based filtering for API queries that adhere to the search standards documented in API References (see page 380). A few examples of how tag-based filtering can be used are as follows:

List all VDBs of type `'Oracle'` , of which IP address contains the `'10.1.100'` string and which have been tagged with the `'team'` tag, `'app-dev-1'` .

```
database_type EQ 'Oracle' AND ip_address CONTAINS '10.1.100' and tags CONTAINS { key
EQ 'team' AND value EQ 'app-dev-1'}
```

12.3 Nuances

12.3.1 Stateful APIs

All applicable DCT APIs are stateful so that running complex queries against a large Delphix deployment can be done rapidly and efficiently. DCT accomplishes this by periodically gathering and hosting telemetry-based Delphix metadata from each engine.

12.3.2 Local data availability

DCT currently relies on existing Continuous Data and Compliance constructs around data-environment-engine relationships. This means that DCT operations require VDBs to live on the engine where the parent dSource lives and so on.

12.3.3 Engine-to-DCT API mapping

Wherever possible, DCT has looked to provide an easier-to-consume developer experience. This means that in some cases, an API on DCT could have an identical API on an engine. However, there are many instances of providing a higher level abstraction for ease of consumption; one example is the data inventory APIs on DCT (sources, dSources, VDBs), which are a simplified representation of data represented by the source, sourceconfig, and repository endpoints on the local engine (source, dSource, and VDB detail are all combined under those three endpoints).

12.3.4 Local references to global UUIDs

In order to avoid collision of identically-named and referenced objects, DCT generates Universally Unique IDentifiers (UUID) for all objects. For existing objects on engines like dSources and VDBs, DCT will concatenate the local engine reference with the engine UUID (e.g. `'Oracle-1'` on engine `'3cec810a-ee0f-11ec-8ea0-0242ac120002'` will be represented as `'Oracle-1-3cec810a-ee0f-11ec-8ea0-0242ac120002'` on DCT).

12.3.5 Environment representations

Environments within Delphix serve as a reference for the combination of a host and instance. This is coupled with the fact that environments can be leveraged by multiple engines at the same time and that engines often have a specific context to some of the elements that comprise an environment. For example, an environment could have both an Oracle and ASE instance installed and that Engine A leverages an Oracle-based workflow and Engine B leverages an ASE workflow. DCT will create two identifiers to represent the specific host and instance combinations. Thus, in DCT, Engine A will be connected to a different uniquely identified Environment than Engine B.

As mentioned earlier with Engine-to-DCT API mapping, DCT aims to simplify the user experience with Delphix APIs by combining different Continuous Data endpoints into a simplified DCT API. The Environment API does this by combining environment, repository, and host endpoints so that writing queries against Delphix data is

a much simpler process. One example would be identifying all environments that have a compatible Oracle home for provisioning:

```
repositories CONTAINS { database_type EQ 'Oracle' and allow_provisioning EQ true AND
version CONTAINS '19.2.3'}
```

12.3.6 Supported data sources/configurations

DCT is compatible with all Delphix-supported data sources and configurations.

12.3.7 Process feedback

Whenever a DCT request completes, it will return a JOB ID as its response. This Job ID can be used in conjunction with the jobs endpoint to query the operation status.

13 DCT Toolkit

13.1 Introduction

DCT Toolkit (dct-toolkit) is the command line application specifically designed to interact with a particular DCT instance. It aims to simplify and ease out interacting with DCT instances while making the process intuitive, with minimal configuration and setup. dct-toolkit abstracts all of the API level implementation complexities and provides a user friendly LINUX-like interface, which can be run from any terminal.

13.2 Compatibility

dct-toolkit is designed to be compatible with any DCT product version, but is fully supported from DCT version 8.0.0 onwards. While dct-toolkit can be used in testing or development environments against any DCT versions, Delphix highly recommends at least DCT version 8.0.0 for using it in the production environment. Some of the value added features are only supported from DCT 8.0.0 version onwards.

dct-toolkit does not need to be upgraded with every new version of DCT. All of the new APIs that are part of the newer DCT releases will automatically be available in dct-toolkit. This is possible because dct-toolkit reads the api specification from the DCT instance (configurable) it is configured to connect to and dynamically generates the list of commands and their respective options.

13.3 New features

13.3.1 Version 1.4.0

- **Enhanced search experience**
  Users can now employ the search command with a more straightforward syntax, in addition to the existing option for filter_expression use. This command structure will search for VDBs that have a name matching `<vdb_name>` and database type matching `<database_type>` :

  ```
  ./dct-toolkit search_vdb name "<vdb_name>" database_type <database_type>
  ```

  - The name flag allows users to specify the VDB name, while the database_type flag lets them indicate the database type. This command structure aims to improve efficiency and ease of use for the search functionality within dct-toolkit.
- **Simplify syntax for value array input**
  Users can now provide array values using more natural and versatile formats, such as comma-separated values or repeated options. Additionally, to handle cases where a comma needs to be included within a value, users can escape the comma using a (\"). Here are the updated examples:

- **Comma-separated values**

  This format allows users to list multiple values for `vdb_ids` in a straightforward, comma-separated manner.

  ```
  vdb_ids=VCDO_ZKJ,ABCD_JHI
  ```

- **Repeated options**

  Users can now repeat the `vdb_ids` option to specify multiple values individually, providing a clear and concise representation.

  ```
  vdb_ids=ABCD_JHI vdb_ids=XYZ_ABC
  ```

- **Escaping Commas Inside Values:**

  ```
  vdb_ids=VCDO_ZKJ,\"AB,CD_JHI\"
  ```

> ⓘ The Simplify syntax for value array input feature in dct-toolkit is not backward compatible with previous versions. As a result, users are unable to use the older method of providing values in string format to array input, as exemplified below:
>
> ```
> vdb_ids='["VCDO_ZKJ", "ABCD_JHI"]'
> ```
>
> This is true for all options where the type is array of value.

13.3.2 Version 1.2.0

- **--json/-js option**

  dct-toolkit now supports a `--json/-js` option for all of the commands. If this option is specified, all properties from the DCT response are printed in JSON format. This option is mutually exclusive with the `--all-columns/-A` and `--columns/-c` options but can be used along with the `--jsonpath/-jp` and `--expand/-ex` options.

- **--csv/-cs option**

  dct-toolkit now supports a `--csv/-cs` option for all of the commands. If this option is specified, all properties from the DCT response are printed in CSV format. This option can be used along with the `--columns/-c` option only to output particular properties from the DCT response. This option can also be specified along with the `--jsonpath/-jp` and `--expand/-ex` options.

  - This option is only applicable to list API responses where the response is printed in table format by default. For all other APIs that return a single object, this option is ignored.

- *create_config* and *encrypt_api_key* commands are now interactive

While providing apiKey value in `create_config` and `encrypt_api_key` command, there is a risk that plaintext `apiKey` can leak via shell history. To protect against this, both of these commands are now made interactive. Providing an apiKey interactively does not leak it via shell history. Following examples demonstrates providing `apiKey` interactively. Once following commands are executed, `dct-toolkit` will ask and wait for the apiKey to be provided by the user before executing the command. The DCT instance URL must contain the protocol and suffix. For example, dct.company.co should be https://dct.company.co/dct/.

```
./dct-toolkit encrypt_api_key dctUrl=<DCT instance URL> apiKey
```

```
./dct-toolkit create_config dctUrl=<DCT instance URL> apiKey
```

13.4 Installation and setup

13.4.1 Installation

dct-toolkit is currently supported on Linux, Windows, and macOS platforms. There are no special system requirements for running it.

To install dct-toolkit, download the executable binaries from the [Delphix download site](127) and extract it to the machine from which it needs to run. Once the executable binary is extracted, grant the execute permission to the binary file.

On **Linux** and **macOS** this can be done from Terminal with the following command:

```
chmod 777 ./dct-toolkit
```

For **Windows** this can be done from cmd with the following command:

```
CACLS files /e /p <username>:F
```

Where `<username>` is the user to whom the permission must be granted and `F` is the **Full Control** permission that needs to be granted.

> The binary is not currently verified for macOS and will give a warning that the binary is untrusted. To get around this issue, you will need to open the binary and explicitly whitelist it. One easy way to achieve this is by pressing the ctrl key and clicking on the binary file, then select Open. On the resulting warning message box, select Open. Now the binary is allowed to be executed from the terminal.

---

127 https://download.delphix.com/folder/4721/Delphix%20Product%20Releases/DCT/Toolkit%20(CLI)

13.4.2 Setup

A plain text configuration file is needed to get started. The recommended way to create this configuration file is via the `create_config` command, available in dct-toolkit.

For creating a typical configuration with only the required properties, use the following command:

```
dct-toolkit create_config dctUrl=<URL of DCT instance> apiKey=<api key value>
```

This will create the configuration file in the default location under the user's home directory. The full path for the configuration file will be `<users home directory>/.dct-toolkit/dct-toolkit.properties`.

The `create_config` command takes in the options mentioned below:

Option name	Required/optional	Description
`dctUrl`	Required	HTTP/S URL of the DCT instance.
`apiKey`	Required	API key of the account used to connect to the DCT instance pointed by the `dctUrl` option.
`configFileOutputLocation`	Optional	Fully qualified path of the properties file. If this option is not specified, the properties file will be created in the `.dct-toolkit` folder under the user's home directory. If this option is specified, the `DCT_TOOLKIT_CONFIG_FILE` environment variable needs to be exported to point to the config file, in order for the config file to be effective.
`apiVersion`	Optional	API version to be used for DCT APIs. If this is not set, the latest API version from the DCT instance will be used.
`apiYamlLocation`	Optional	Location of the local api.yaml file. If this is not set, the latest api.yaml file will be downloaded from the DCT instance that is configured.
`--insecureSSL`	Optional	Whether to use insecure SSL connection to the DCT instance. By default, this will be set to false.

`--unsafeHostnameCheck`	Optional	Whether to disable hostname verifier checks for SSL connection to the DCT instance. By default, this will be set to false.
`sslCertificate`	Optional	PEM format SSL certificate path to be used for SSL connection to DCT instance.
`logLevel`	Optional	Log level to set. Can be one of OFF, ERROR, WARN, INFO, DEBUG, TRACE, ALL. By default, the log level will be INFO.
`logDir`	Optional	Directory where the logs should be written. By default, logs will be written to the logs directory created under the `.dct-toolkit` folder in the user's home directory.

All of these options and help can be requested with the following command:

```
./dct-toolkit create_config -h
```

All of the above properties can also be individually specified using environment variables. Environment variable mappings for each of the above properties are as follows:

Property name	Environment variable name
`dctUrl`	DCT_TOOLKIT_DCT_URL
`apiKey`	DCT_TOOLKIT_API_KEY
`apiVersion`	DCT_TOOLKIT_API_VERSION
`apiYamlLocation`	DCT_TOOLKIT_API_YAML_LOCATION
`--insecureSSL`	DCT_TOOLKIT_SSL_INSECURE
`--unsafeHostnameCheck`	DCT_TOOLKIT_SSL_UNSAFE_HOSTNAME_CHECK

sslCertificate	DCT_TOOLKIT_SSL_CERT
logLevel	DCT_TOOLKIT_LOG_LEVEL
logDir	DCT_TOOLKIT_LOG_DIR

Configuring non-sensitive properties in a properties file and sensitive properties (like `apiKey`) via an environment variable is acceptable. Properties set via environment variable will take precedence over the ones specified in the properties file.

Once the required properties are available via either the properties file or the environment variable, dct-toolkit is ready for use.

13.5 Usage guide

All of the DCT APIs are available as commands in dct-toolkit. To find the list of all commands which are available, use the following command:

```
./dct-toolkit -h
```

This will print the list of all commands grouped by the appropriate entity names.

13.5.1 Examples

- The list of available options for a particular command can be requested with the following command:

```
./dct-toolkit <command_name> -h
```

- There are currently many options for provisioning a VDB, some options are only valid for a particular DB type. For example, the `unique_name` option is only applicable to Oracle DBs. For better ease of use, dct-toolkit has subcommands created under the provisioning command, with options that are relevant to that particular subcommand. For example, these are the subcommands under the `provision_vdb_by_snapshot` command:

```
oracle
sybase
mssql
appdata
postgres
```

For provisioning an Oracle VDB, use the following command:

```
dct-toolkit provision_vdb_by_snapshot oracle source_data_id=<dsourceId> --
auto_select_repository unique_name=<uniqueName>
```

For provisioning a Sybase-ASE VDB, use the following command:

```
dct-toolkit provision_vdb_by_snapshot sybase source_data_id=<dsourceId> --
auto_select_repository --truncate_log_on_checkpoint
```

- For getting the list of registered engines, use the following command:

```
dct-toolkit get_registered_engines
```

The above command will only print important fields in the response.
For printing all of the fields in the response, use the following command:

```
-A/--all-columns
```

```
dct-toolkit get_registered_engines -A
```

To list only the selected columns that are useful to the user, use the following command:

```
--columns/-c
```

```
dct-toolkit get_registered_engines --columns=id,name,cpu_core_count...
```

For advanced use cases, dct-toolkit also provides an option to specify a jsonpath[128] expression to extract required objects from the JSON response. This can be requested with the following command:

```
--jsonpath/-jp
```

```
dct-toolkit get_registered_engines --jsonpath=<json path expression starting
with $>
```

Note, these options are available for all of the commands available in dct-toolkit.

- For commands that use a request body, dct-toolkit provides `-body`, which can be used to specify a JSON body (instead of using individual options).

---

[128] https://github.com/json-path/JsonPath

```
dct-toolkit token_info --body=<JSON body to POST to DCT>
```

Help regarding the exact structure of JSON request and response body can be requested for all of the commands with the following command:

```
-jh/--jsonhelp
```

```
dct-toolkit token_info -jh
```

- dct-toolkit provides an easy to use and understand format for specifying tags. For example, the following command adds two tags – `{key=purpose, value=testing}` and `{key=key1, value=value1}`:

```
dct-toolkit create_environment_tags environmentId=<envId> tags purpose=testing
key1=value1
```

- Some of the DCT APIs trigger an asynchronous job and return a jobId in response. dct-toolkit, by default, will wait for the asynchronous job to be completed and will report the end state of the job.
    - There is an option to **not** wait for the completion of an asynchronous job. Use the following command with the `--no-wait` option:

```
dct-toolkit refresh_environment environmentId=<envId> --no-wait
```

- If the user intends on specifying the API key via an environment variable, dct-toolkit provides a command to get the encrypted version of the API key.

```
dct-toolkit encrypt_api_key apiKey=<DCT api key> dctUrl=<DCT instance URL>
```

13.5.1.1 `--json/-js` and `--csv/-cs` option examples

- For the `--json/-js` option, to print the DCT response as it is received from DCT instance users can use following command:

```
dct-toolkit get_environments --json
dct-toolkit get_environment_by_id environmentId=<environment Id> --json
```

- To print the repositories object from environment object in JSON format users can use following command:

```
dct-toolkit get_environments --json --expand=repositories
```

- For the `--csv/-cs` option, to print the DCT response in CSV format users can use the following command:

```
dct-toolkit get_environments --csv
```

- To print the repositories object from the environment object in CSV format users can use the following command:

```
dct-toolkit get_environments --csv --expand=repositories
```

13.5.2 Object IDs and names

Object IDs in DCT are opaque Strings which are globally unique and thus unambiguously represent exactly one object in the system. However, since Object IDs are often not known ahead of time (and not friendly to remember or work with), *dct-toolkit* will accept an object name wherever an object id is expected in any of the command parameters, as long as the object name is globally unique. For example, the following dct-toolkit commands are equivalent if `AppFS_DEV` is the name corresponding to the VDB with id `MSSQL_DB_CONTAINER-10` and is globally unique.

```
dct-toolkit get_vdb_by_id vdb_id="2-MSSQL_DB_CONTAINER-10"
dct-toolkit get_vdb_by_id vdb_id="AppFS_DEV"
```

However, if the object name is not unique, in this example if 2 or more VDBs have the same name, using the VDB name would fail:

```
dct-toolkit get_vdb_by_id vdb_id="AppFS_DEV"
Multiple objects match the provided ID.
```

The solution would then be to search for the VDBs with that name and identify the ID of the VDB:

```
dct-toolkit search_vdbs name=AppFS_DEV -c
id,database_type,name,engine_name,group_name

          id              | database_type |  name       |      engine_name     |
group_name     |
--------------------------------------------------------------------------------
---------------
  2-MSSQL_DB_CONTAINER-10 |         MSSql  | AppFS_DEV   |     dev-engine-1     |
dev-group      |
```

```
  3-MSSQL_DB_CONTAINER-127   |             MSSql   |   AppFS_DEV   |   dev-engine-2        |
qa-group      |
---
response_metadata:
  total: 2
```

Here is another example where names are used instead of IDs for multiple parameters:

```
dct-toolkit provision_vdb_by_snapshot appdata source_data_id="AppFS_MASK" name="AppFS
_DEV" environment_id="Postgres_Target" environment_user_id="delphix" --
auto_select_repository target_group_id='DEV' mount_point="/mnt/provision/AppFS_DEV"
```

Note that object names in DCT are case sensitive, meaning that `AppFS_DEV` is not equivalent to
`appfs_dev` .

13.6 Auto-completion support

To enhance your experience with `dct-toolkit` , an auto-completion feature is implemented to streamline
command entry. This functionality facilitates the auto-completion of commands and options with a simple
press of the Tab key.

To enable this feature, first generate the auto-completion script by executing the following command:

```
dct-toolkit generate_completion >> dct_toolkit_auto_completion_script
```

For immediate use in your current shell session, activate the script with the following command:

```
source dct_toolkit_auto_completion_script
```

Alternatively, for a swift activation within the same session, the following command can be utilized:

```
source <(dct-toolkit generate_completion)
```

Please note that these methods only apply the auto-completion feature for the duration of the current shell
session. For a more permanent solution that ensures auto-completion is available in all future sessions, you
must append a command to your shell's configuration file (`~/.bashrc` for Bash or `~/.zshrc` for Zsh).
Use one of the following commands for Bash or Zsh, respectively.:

- For Bash:

```
echo "source <(dct-toolkit generate_completion)" >> ~/.bashrc
```

- For Zsh:

```
echo "source <(dct-toolkit generate_completion)" >> ~/.zshrc
```

> ℹ️ Please be aware that the `dct-toolkit` auto-completion feature is currently supported only on Bash-type shells.

13.7 API key encryption

13.7.1 Overview

dct-toolkit currently authenticates HTTP requests with an API key. The API key can be configured in plain text or specified via an environment variable.

13.7.2 Implementation

Starting from 1.2.0 version, dct-toolkit supports encrypting an API key via a separate command and also via the `create_config` command used to generate the configuration file. The `create_config` command now stores the API key in encrypted format using the AES/GCM algorithm, where the key and IV are generated from the following two inputs:

1. Local hostname of the machine from where the dct-toolkit command is run.
2. DCT URL used to connect to the DCT instance.

For users who want to supply the API key via an environment variable, dct-toolkit provides the `encrypt_api_key` command which takes in the API key and DCT URL as inputs and returns an encrypted API key, which can then be exported to an environment variable. As an example, please refer to the section below.

> ℹ️ The encryption key (not to be confused with the DCT API key) used for encryption is reproducible, hence, it is possible to encrypt and decrypt the DCT API key without storing the encryption key on the file system. Anyone with access to the logic to create an encryption key and knows the inputs for that logic will be able to generate the encryption key, and consequently, will be able to decrypt the DCT API key.

13.7.3 Backward compatibility

The dct-toolkit API key encryption feature is fully backward compatible with previous versions of dct-toolkit. dct-toolkit first tries to decrypt the API key; if it fails to decrypt the API key, then the key is used as configured.

13.7.4 Example

If the user intends on specifying the API key via an environment variable, then dct-toolkit provides a command to get the encrypted version of the API key:

```
./dct-toolkit encrypt_api_key apiKey=<DCT api key> dctUrl=<DCT instance URL>
```

13.8 Configure multiple DCT instances in dct-toolkit

13.8.1 Overview

This page describes a setup for a case where users need to connect to multiple DCT instances via dct-toolkit on a single engine. This would be useful in a case like upgrade testing. The user would upgrade the DCT instance in a lower environment (like a development environment) first, then run test APIs with dct-toolkit. Once all of the tests are successful, the user can upgrade the higher environment (like a production environment) and run checks via dct-toolkit. Being able to switch between DCT instances without hassle can be easily configured in dct-toolkit, as described below.

13.8.2 Example

The user will first need to create separate dct-toolkit configuration files for the environments that need to connect via dct-toolkit. This can be done by using the `configFileOutputLocation` option available for the `create_config` command:

```
./dct-toolkit create_config url=https://prod.... api_key=....
configFileOutputLocation=<prod config file path>
./dct-toolkit create_config url=https://dev.... api_key=....
configFileOutputLocation=<dev config file path>
```

Once the configuration files are created, users can create separate aliases for the config files generated above:

```
alias dct-toolkit-prod="DCT_TOOLKIT_CONFIG_FILE=<prod config file path>"
alias dct-toolkit-dev="DCT_TOOLKIT_CONFIG_FILE=<dev config file path>"
```

Once the aliases are set up, users can then switch between production and development environments by invoking the corresponding aliases.

To get sources from the production environment, use the following command:

```
dct-toolkit-prod
./dct-toolkit get_sources
```

To get sources from the development environment, use the following command:

```
dct-toolkit-dev
./dct-toolkit get_sources
```

13.9 Logging

dct-toolkit has inbuilt logging support, which will export all of the logs to a default location under the user's home directory. The full path will be `<user home directory>/.dct-toolkit/logs`. dct-toolkit creates a separate log file for every date.

Log level and log directory are both configurable and can be specified in the configuration file or via the environment variable. Please refer to the Installation and setup page for more information.

14 Developer resources

14.1 API summary

14.1.1 Introduction

This page provides an overview of the various data APIs supported by Data Control Tower (DCT), along with example requests and guidance on using them.

DCT offers support for a handful of primary API options, based on development needs and preferences:

1. **REST API**
2. **Command Line Interface**
3. **Go SDK**
4. **Python SDK**
5. **Integrations**

14.1.2 1. REST API

The most popular option for interacting with DCT APIs.

- **Interactive API documentation:** You can explore and experiment with the REST API directly in your web browser using the Swagger interface. To access it, enter your DCT hostname (with the protocol and suffix included) followed by the `/api` path in the address bar. For example, if DCT is running on `gateway.example.com`, navigate to `https://gateway.example.com/dct/api`.
- **API references:** For a detailed list of available REST API endpoints, see our API references[129] page.
- **Authentication with Swagger:** When authenticating with the Swagger API, be sure to prefix your API key with `apk`. For instance, if your API key is `1.2345`, you should use `apk 1.2345` as the authentication header.

It is important to recognize that the REST APIs are the foundation for all other integrations.

---

[129] https://dct.delphix.com/docs/latest/api-references

14.1.3 2. Command Line Interface (CLI)

The second most popular option for interacting with DCT APIs, done through a command line called the `dct-toolkit` .

- **DCT Toolkit documentation**: The DCT Toolkit commands for DCT can be found here[130], in our documentation.

14.1.4 3. Go SDK

For developers who prefer to work with Go, DCT provides a Go SDK to help simplify development and integration.

- **Go SDK documentation:** The Go bindings for DCT can be found here[131], on go.dev.

14.1.5 4. Python SDK

The Python SDK is another option for interacting with DCT, offering a Pythonic way to automate tasks within DCT.

- **Python SDK documentation:** The Python bindings can be found here[132], on PyPi.
  - You can install the latest version with the following command:

```
pip install delphix-dct
```

14.1.5.1 4a. Querying engines example for Python

This exemplifies how to use the Python SDK to query DCT for information about connected Continuous Data Engines.

```python
import delphix.api.gateway
import delphix.api.gateway.configuration
import delphix.api.gateway.api.management_api

cfg = delphix.api.gateway.configuration.Configuration()
cfg.host = "<https://localhost/v2>"

# For example purposes
cfg.verify_ssl = False

# Replace the string with your own API key
```

---

130 https://dct.delphix.com/docs/latest/dct-toolkit
131 https://pkg.go.dev/github.com/delphix/dct-sdk-go
132 https://pypi.org/project/delphix-dct-api/

```
cfg.api_key['ApiKeyAuth'] = 'apk 3.tEd4DXFce'

api_client = delphix.api.gateway.ApiClient(configuration=cfg)
engines_api = delphix.api.gateway.api.management_api.ManagementApi(api_client)

print(engines_api.get_registered_engines())
```

The result should appear similar to the following:

```
{
    "items": [
        {
            "connection_status": "ONLINE",
            "cpu_core_count": 2,
            "data_storage_capacity": 23404216320,
            "data_storage_used": 11589626880,
            "hostname": "avm.delphix.com",
            "id": 1,
            "insecure_ssl": True,
            "memory_size": 8589934592,
            "name": "vmname",
            "password": "******",
            "status": "CREATED",
            "tags": [],
            "type": "UNSET",
            "unsafe_ssl_hostname_check": False,
            "username": "admin",
            "uuid": "ec2fbfea-928b-07f8-94c4-29fea614624f",
            "version": "6.1.0.0"
        }
    ]
}
```

14.1.6 5. Integrations

If you prefer a more out-of-the-box approach for your DevOps tooling, Delphix offers a wide array of integration options, outlined more in the Ecosystem hub for integrations[133] documentation. There, you can find information on Delphix-supported integrations, such as ServiceNow, Jenkins, Terraform, or Kubernetes.

14.2 API references

> ℹ **API documentation for DCT version 20.0.0 and ABOVE can be found by expanding a section and clicking a link, where the * version coincides with this version of documentation.**

---

[133] https://ecosystem.delphix.com/docs/main/integrations

DCT CD APIs (Continuous Data)

- 22.0.0[134]*
- 21.0.0[135]
- 20.0.0[136]

DCT CC APIs (Continuous Compliance)

- 22.0.0[137]*
- 21.0.0[138]
- 20.0.0[139]

---

[134] https://help-api.delphix.com/dct/cd/22.0.0/
[135] https://help-api.delphix.com/dct/cd/21.0.0/
[136] https://help-api.delphix.com/dct/cd/20.0.0/
[137] https://help-api.delphix.com/dct/cc/22.0.0/
[138] https://help-api.delphix.com/dct/cc/21.0.0/
[139] https://help-api.delphix.com/dct/cc/20.0.0/