Data Control Tower Home

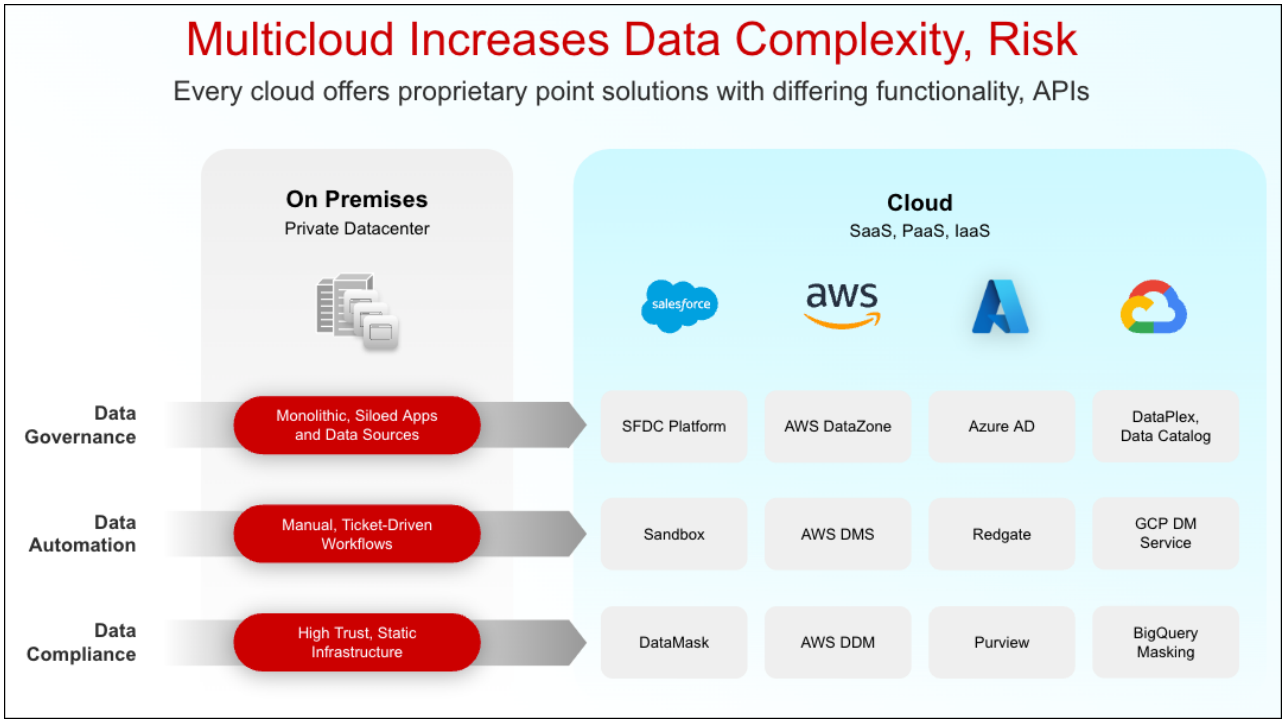
Data Control Tower

Exported on 08/15/2023

Table of Contents

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.

2 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.

2.1 New features

2.1.1 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 the case of parent and replicated engines, both are registered with DCT.

2.1.2 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.

2.1.3 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.

2.1.4 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.

2.1.5 Release 5.0.1

2.1.5.1 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 187).
  - **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 187).
- **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**
    - **Add scope** to an Access Group
      `POST: /access-groups/{accessGroupId}/policies → POST /access-groups/{accessGroupId}/scopes`
    - **Remove scope** from Access Group
      `DELETE /access-groups/{accessGroupId}/policies/{policyId} → DELETE /access-groups/{accessGroupId}/scopes/{scopeId}`
    - **Get** Access Group scope
      `GET /access-groups/{accessGroupId}/policies/{policyId} → GET /access-groups/{accessGroupId}/scopes/{scopeId}`
    - **Update** Access Group scope
      `PATCH /access-groups/{accessGroupId}/policies/{policyId} → PATCH /access-groups/{accessGroupId}/scopes/{scopeId}`
    - **Add object tags** to Access Group scope
      `POST /access-groups/{accessGroupId}/policies/{policyId}/object-tags → POST /access-groups/{accessGroupId}/scopes/{scopeId}/object-tags`
    - **Remove object tags** from Access Group scope
      `POST /access-groups/{accessGroupId}/policies/{policyId}/object-tags/delete → POST /access-groups/{accessGroupId}/scopes/{scopeId}/object-tags/delete`

    - **Add objects** to Access Group scope
      `POST /access-groups/{accessGroupId}/policies/{policyId}/objects → POST /access-groups/{accessGroupId}/scopes/{scopeId}/objects`
    - **Remove objects** 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

2.1.5.2 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).

2.1.6 Release 4.0

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

2.1.7 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

2.1.8 Release 2.2

2.1.8.1 Deployment

- Introducing Kubernetes and OpenShift support

2.1.8.2 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

2.1.8.3 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

2.2 Fixed issues

2.2.1 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.

2.2.2 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.

2.2.3 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.
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.

2.2.4 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.

2.2.5 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.

2.2.6 Release 6.0.0 changes

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

2.2.7 Release 5.0.3 changes

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

2.2.8 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.

2.2.9 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.

2.2.10 Release 3.0.0 changes

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

2.3 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	6.0.0.1 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).

Version	Release date	Can upgrade to
2.0.0	Jun 27th, 2022	2.1.0 - 9.0.x
2.1.0	Sep 8th, 2022	2.2.0 - 9.0.x
2.2.0	Oct 17th, 2022	3.0.0 - 9.0.x
3.0.0	Dec 15th, 2022	4.0.0 - 9.0.x
4.0.0	Jan 19th, 2023	5.0.0 - 9.0.x
5.0.x	Feb 16th, 2023	6.0.0 - 9.0.x
6.0.x	Mar 29th, 2023	7.0.0 - 9.0.x
7.0.x	May 9th, 2023	8.0.0 - 9.0.x
8.0.x	Jun 22nd, 2023	9.0.x
9.0.x	Jul 26th, 2023	N/A

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

3 Getting started

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.

> ⓘ 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.**

3.1 Container Platform Support

Data Control Tower (DCT) supports the most popular distributions of Kubernetes and Openshift. If you do not see your distribution or platform of choice, please reach out to your account team for more details.

3.1.1 Kubernetes Support

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.

3.1.2 Openshift Support

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.

3.1.3 Docker Compose

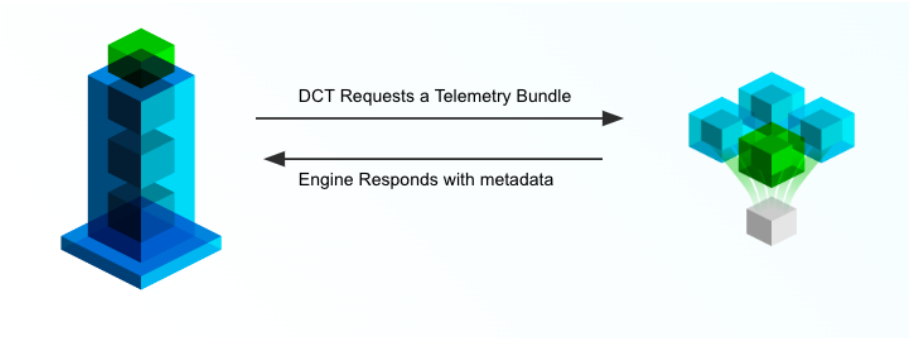
DCT supports Docker Compose but only recommends using this platform for testing/non-production purposes due to the inherent limitations to deployment scalability. Note - DCT has documentation on migrating deployments from Docker Compose to Kubernetes and Openshift.

3.2 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. Note: 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 of the heavy lifting.

3.3 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 can be used to 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 what tags to implement, please consider who will have access to creating tags (developer access versus admin only), which will impact how teams are able to collaborate with one another.

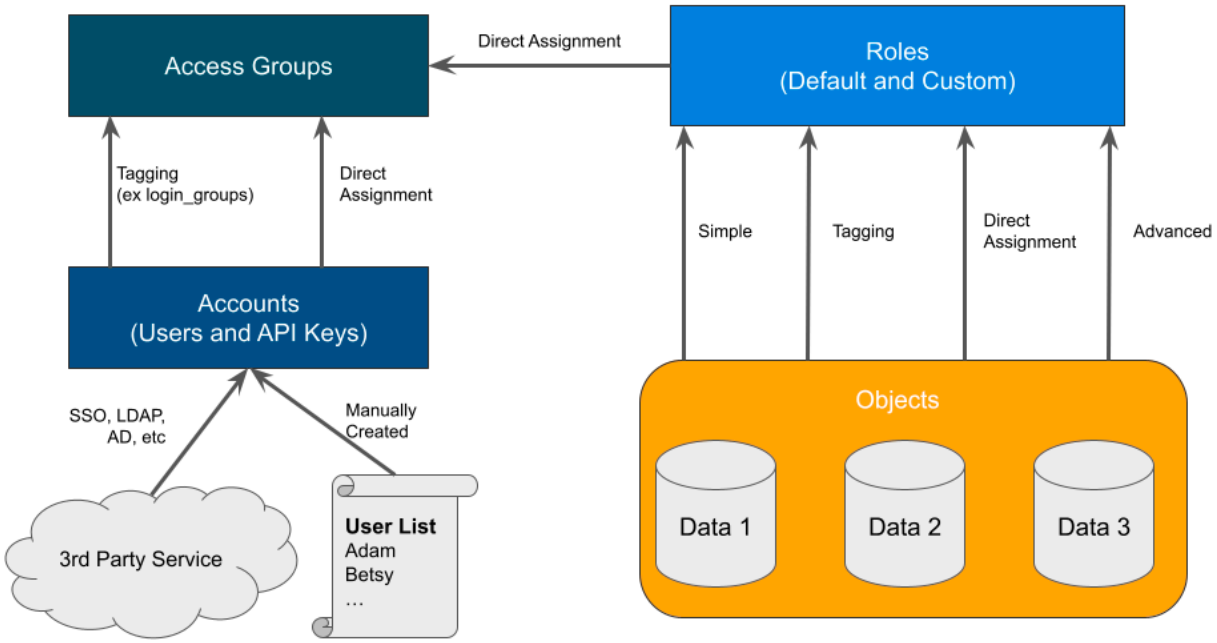
Also, Delphix recommends that the DCT administrative team creates Delphix-wide documentation on these tagging standards to reduce the risk of deviation.

3.4 Plan your Access Control Strategy

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 DCT's 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.

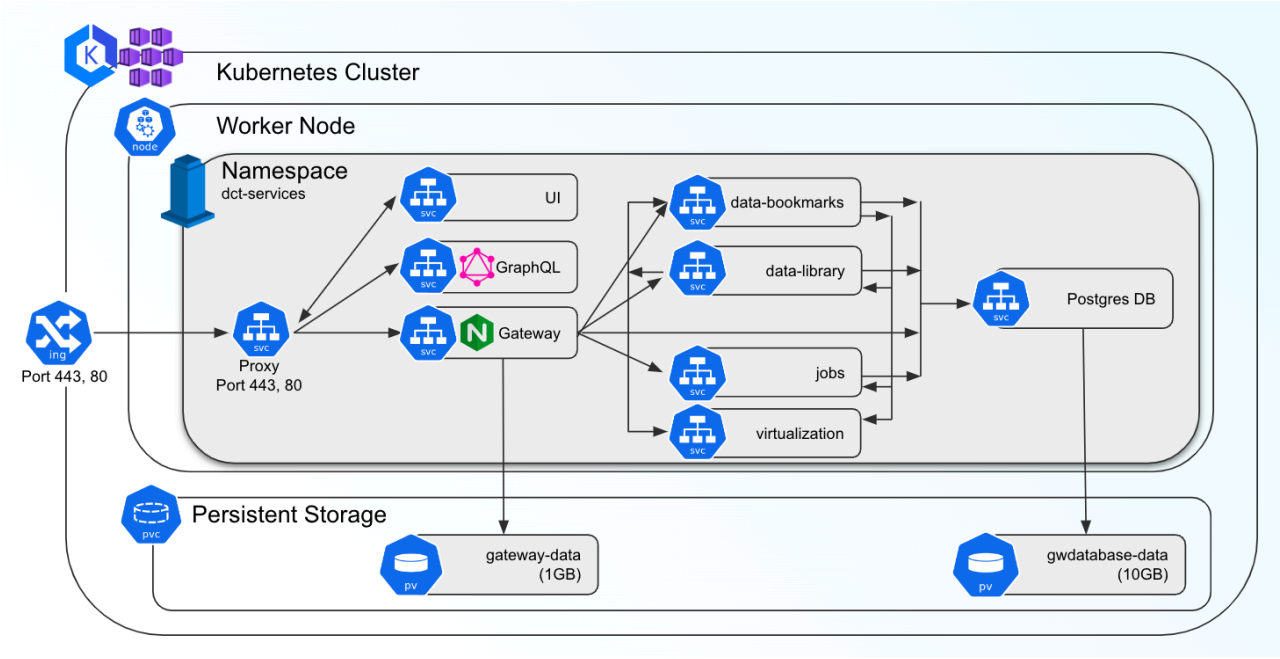


Understanding your team structure is imperative to identify the best access model. Usually, organizations have existing groupings defined in their Identify Provider (IdP). These groups are typically organized in one of two ways (a) a team dedicated towards a central goal (such as a product development 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[1], will produce a streamlined rollout.

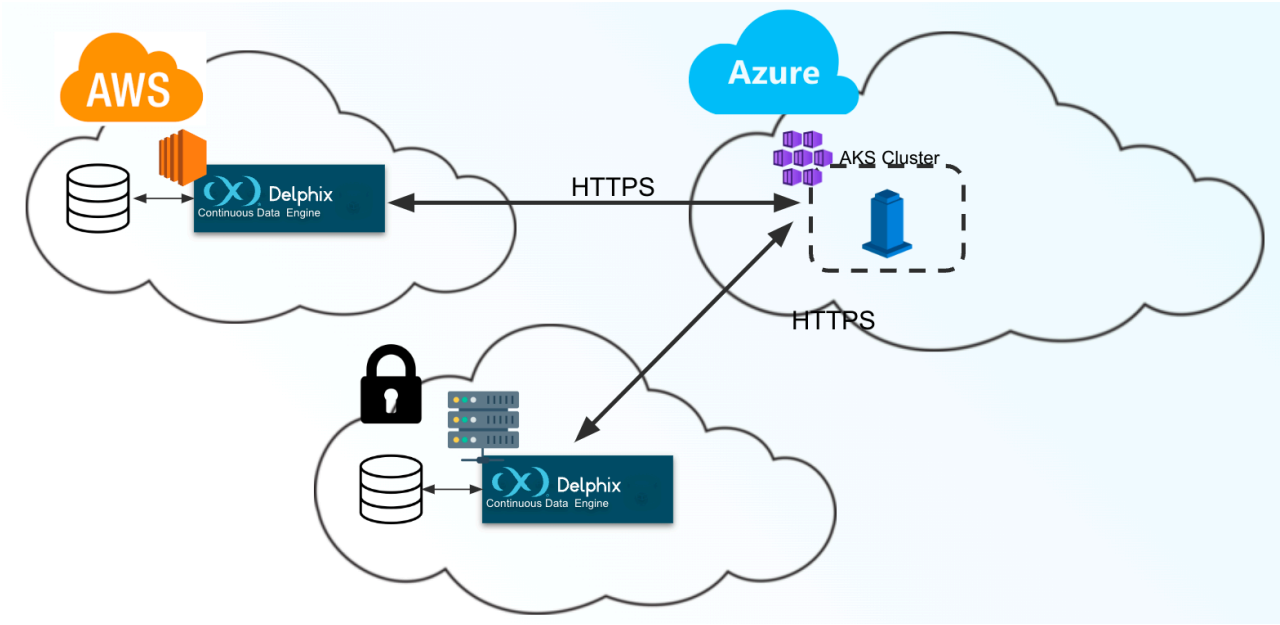
1 https://en.wikipedia.org/wiki/Principle_of_least_privilege

4 Deployment

Data Control Tower is a container-based architecture and 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 and resilient 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.



This section will explain all of the required steps to deploy DCT on your container platform of choice.

4.1 Kubernetes

4.1.1 Installation and setup for Kubernetes

> ⓘ Before getting started, Delphix recommends that you engage your Kubernetes Admin for deployment guidance on available Kubernetes deployment platforms (AKS, EKS, etc.), node sizing, and persistent volume settings.

4.1.1.1 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 **values.yaml** file and can be overridden during deployment.

CPU: 4-Core
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. If the persistent volume is mounted externally, the host requires 39 GB of storage, since the default storage required by the database (10 GB) and gateway (1 GB) draws from the external storage. The default storage configuration for the database and gateway can be modified in values.yaml.

4.1.1.2 Kubernetes Overview

Data Control Tower can be deployed in a manner 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 you will deploy DCT.
- 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[2] (Note: 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[3])
- kubectl - is a command line tool that enables administrative communication with the deployed pods (most useful post-deployment or after an upgrade)

4.1.1.3 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.

Requirement	DCT Recommended Version	Comments
Kubernetes Cluster	1.25 or above	
HELM	3.9.0 or above	DCT also requires access to the HELM repository from where DCT charts can be downloaded. The HELM repository URL is https://dlpx-helm-dct.s3.amazonaws.com[4]. If an intermediate (local) 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.

2 https://dlpx-helm-dct.s3.amazonaws.com/
3 http://download.delphix.com
4 https://dlpx-helm-dct.s3.amazonaws.com/

Requirement	DCT Recommended Version	Comments
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/ .

4.1.1.4 Installing DCT

The latest version of the chart can be pulled locally with the following command (where `X.0.0.tgz` should be changed to the version of DCT being installed):

```
curl -XGET https://dlpx-helm-dct.s3.amazonaws.com/delphix-dct-X.0.0.tgz -o delphix-dct-X.0.0.tgz
```

This command will download a file with the name **delphix-dct-7.0.0.tgz** in the current working directory. The downloaded file can be extracted using the following command (where `X.0.0.tgz` should be changed to the version of DCT being installed):

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

This will extract into the following directory structure:

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

For pulling the Docker images from the registry, temporary credentials would need to be configured/overridden in the **values.yaml** file. For getting the temporary credentials, visit the Delphix DCT Download[5] page and login with your customer login credentials. Once logged in, select the **DCT Helm Repository** link and accept the Terms and Conditions.

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

Once accepted, login credentials will be presented. In version 8.0.0 and above, instead of username/password, an option is available to use credentials from a pre-existing Kubernetes Secret[6]. 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 like this:

```
imageCredentials:
  # registry to pull docker images from.
  registry: 762392488304.dkr.ecr.us-west-2.amazonaws.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 extracting the chart, 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. In the above directory structure, the **values.yaml** file contains all of the configurable properties with their default values. These default values can be overridden while deploying DCT, as per the requirements. If the values.yaml file needs to be overridden, create a copy of values.yaml and edit the required properties. While deploying DCT, values.yaml file can be overridden using the following command:

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

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 https://kubernetes.io/docs/concepts/configuration/secret/

4.1.2 Bootstrapping API keys

4.1.2.1 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[7] with type **APK**. 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 you haven't performed this configuration step[8] yet, and understand the risk, you may disable the check in the HTTP client. For instance, this can done with cURL using the `--insecure` flag.

> cURL version >7.43 is recommended.

Bootstrap First API Key

There is a special process to bootstrap the creation of the first API key. This first API key should only be used to create another key and then promptly deleted, since the bootstrap API will appear in the logs. 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 application is started, edit the **values.yaml** file and modify the following lines, to set the `apiKeyCreate` to the string value `true`. Toggle this value to create/seed bootstrap API key:

```
apiKeyCreate: true
```

Upgrade DCT with:

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

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

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

You will see the following output in the logs for the *gateway* pod (the key will be different from this example):

---

7 https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization
8 https://docs.delphix.com/display/dctmc/Authentication?src=contextnavpagetreemode#Authentication-replace-https

```
NEWLY GENERATED API KEY: 1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWaj
uKcBT3
```

Logs for a gateway pod can be accessed using:

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

`gateway-pod-name` will be of the format `gateway-xxx` and can be found using the following command:

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

Copy the API Key, it can now be used to authenticate with DCT. Remember that the API Key value must be prefixed with **APK**. An example cURL command with the above API Key appears as follows:

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

Edit the **values.yaml** file to set the `apiKeyCreate` environment variable value back to `false` and upgrade DCT again with:

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

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

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

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.

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

```
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
}'
```

A response should be received similar to the lines below:

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

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>/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>/v2/management/accounts/' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>'
```

4.1.3 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.

4.1.4 Admin topics for Kubernetes

4.1.4.1 Deployment upgrade for Kubernetes

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

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

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

Download the new version of chart using the following command in tandem with the newly created folder (where `x.0.0` should be changed to the version of DCT being installed):

```
$cd dct-x.0.0
$curl -XGET https://dlpx-helm-dct.s3.amazonaws.com/delphix-dct-x.0.0.tgz -o delphix-dct-x.0.0.tgz
```

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

Copy the values.yaml file from the previous version parallel to the dct-x.0.0 folder.

This values.yaml file contains modified values from the existing previous version of deployment.

Since the Docker Registry (AWS ECR) expires after 12 hours, the Docker Registry should be modified in the values.yaml (from the previous existing version) with the latest password. It can be obtained from https://download.delphix.com[9]. Here are some notes in regards to this step in the process:

9 https://download.delphix.com/

- This 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).
- In case a user is using their internal Docker Registry, they should first pull the next version of the Docker images from the Delphix provided registry, using a new password.
- Steps to pull Docker images from the Docker Registry:

Docker login command (password from https://download.delphix.com[10]):

```
$docker login --username AWS --password [PASSWORD] 762392488304.dkr.ecr.us-west-2.ama
zonaws.com/delphix-dct
```

Pull Docker images of DCT Services:

```
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:nginx-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:app-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:data-bookmarks-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:delphix-data-library-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:graphql-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:ui-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:jobs-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:postgres-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:virtualization-x.0.0
```

The last step is to run the helm upgrade command:

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

4.1.4.2 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.

10 https://download.delphix.com/

4.2 OpenShift

4.2.1 Installation and setup for OpenShift

4.2.1.1 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
Memory: 16GB
Storage: 50GB
Port: 443

4.2.1.2 Installation requirements (OpenShift)

DCT requires a running OpenShift cluster to run, `oc` command line tool to interact with OpenShift cluster, and HELM for deployment on to 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/ . DCT also requires access to the HELM repository from where DCT charts can be downloaded. The HELM repository URL is https://dlpx-helm-dct.s3.amazonaws.com[11].
oc	4.11.3 or above	To install oc follow the instructions at https://docs.openshift.com/container-platform/4.8/cli_reference/openshift_cli/getting-started-cli.html .

11 https://dlpx-helm-dct.s3.amazonaws.com/

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.

4.2.1.3 Installation process

Jumpbox setup

OC login

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

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

Verify KubeConfig

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

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

```
oc config current-context
```

Create a new project

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

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

Installing Helm

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

DCT also requires access to the HELM repository from where DCT charts can be downloaded. Run the following commands to add the repository (where `x.0.0` should be changed to the version of DCT being installed):

```
curl -XGET https://dlpx-helm-dct.s3.amazonaws.com/delphix-dct-x.0.0.tgz -o delphix-
dct-x.0.0.tgztar -xvf delphix-dct-x.0.0.tgz
```

Deploy DCT chart

Find and update fsGroup 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
```

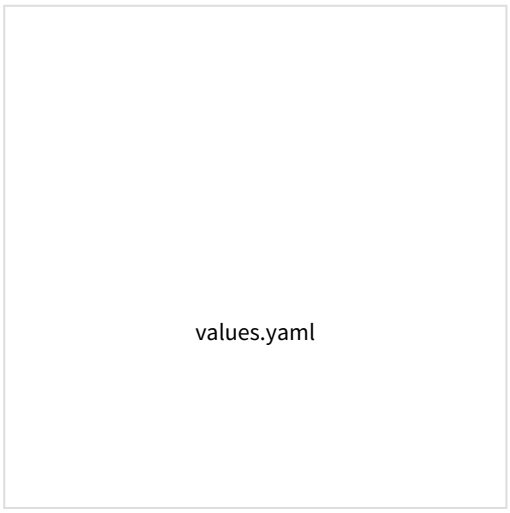
Copy the first value from the `openshift.io/sa.scc.supplemental-groups` line, before the slash (e.g. 1001000000).

Paste this value in the values.yaml file:

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

Create values.yaml file

Create a values.yaml file and update the properties according to your environment. A sample values.yaml file can be downloaded below.



(see page 35)

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 -f <path to edited values.yaml> dct-services apigw-repo/delphix-dct -
version=x.0.0
```

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
```

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
```

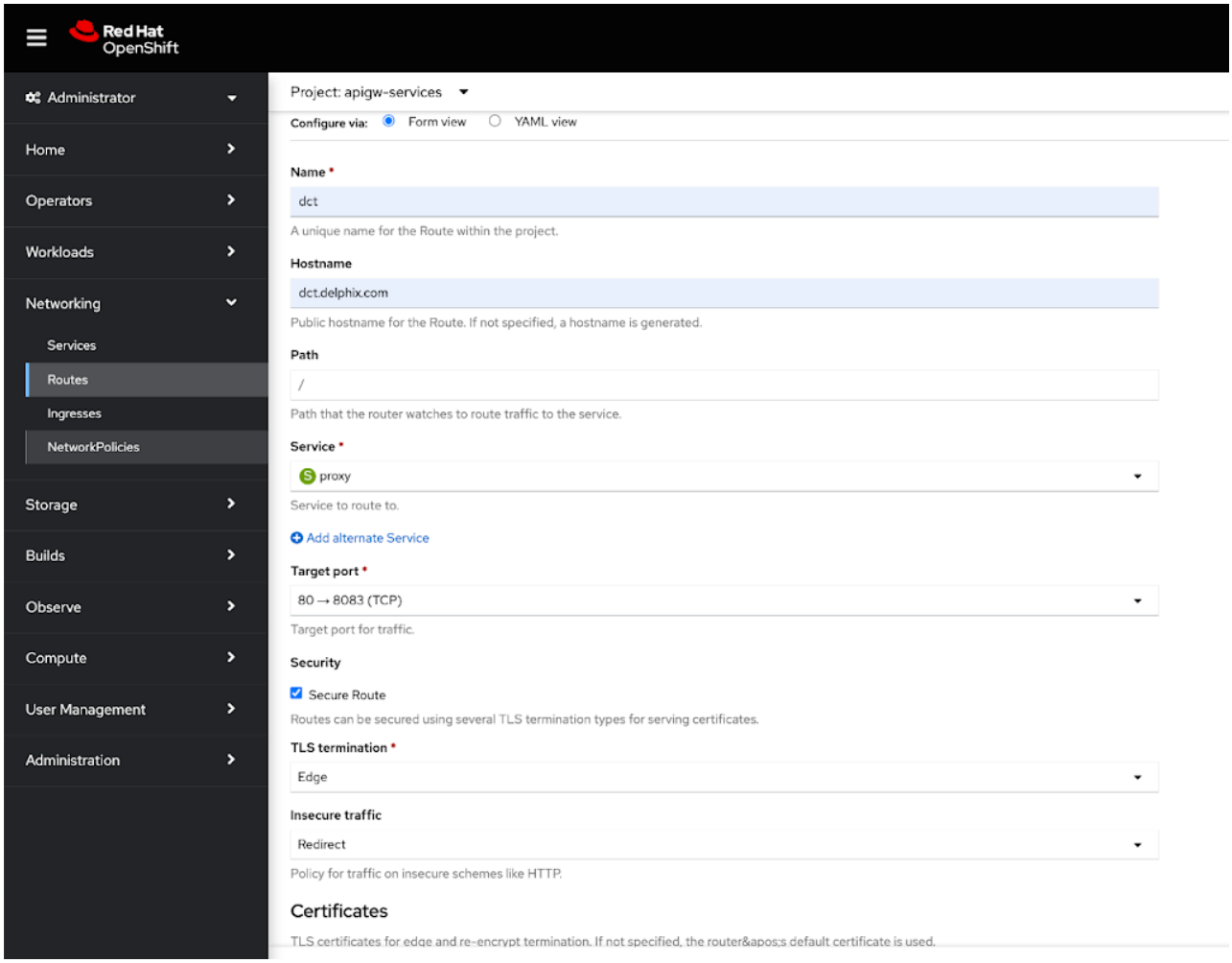
4.2.1.4 Configure Ingress

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

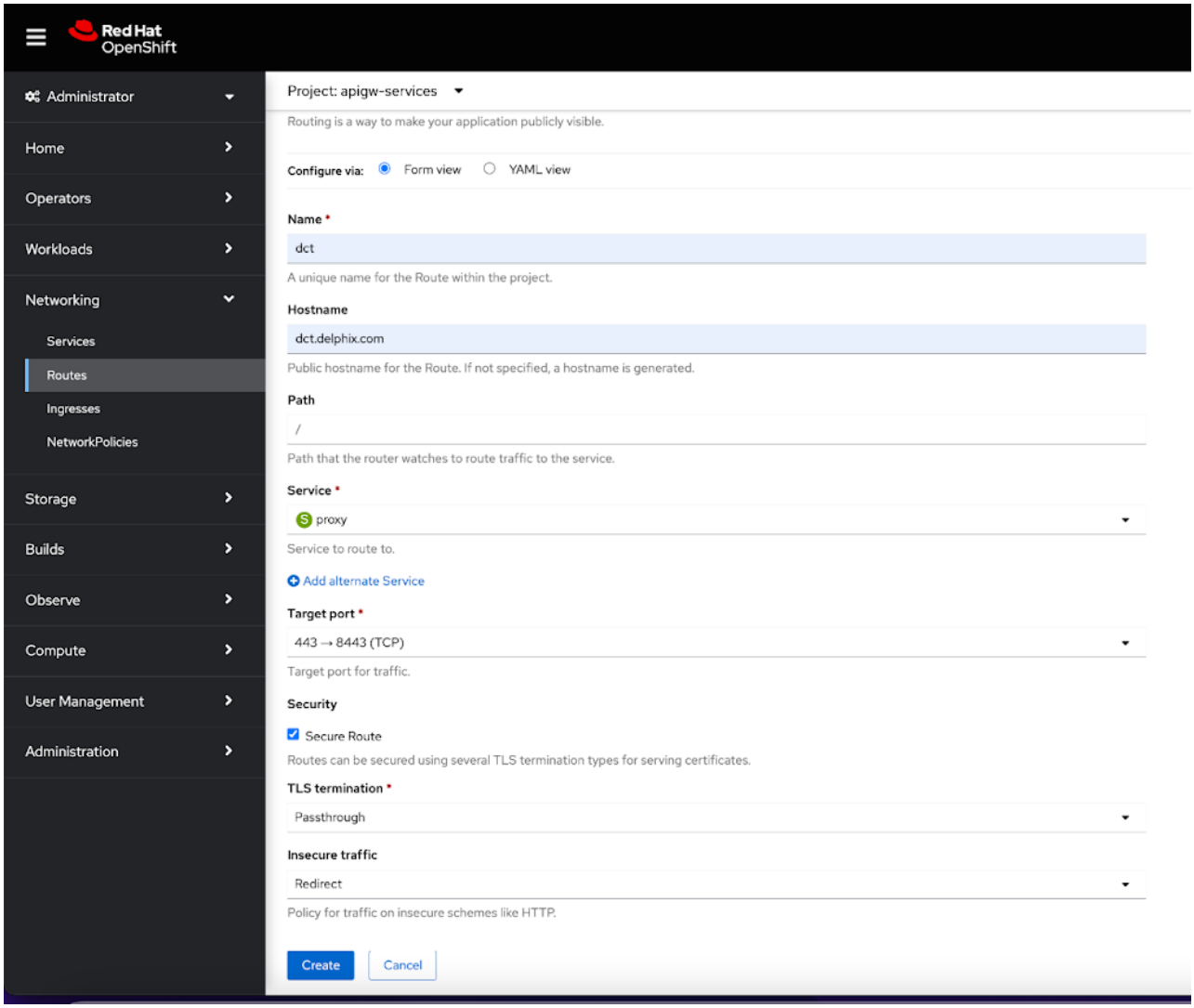
Creating route

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:



4.2.2 OpenShift authentication

4.2.2.1 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.

4.2.2.2 Enable OAuth2 authentication

By default APIKey authentication will be enabled and when DCT starts it will generate a new API key(see page 59) 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
```

4.2.3 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.

4.2.4 Admin topics for OpenShift

4.2.4.1 Deployment upgrade for OpenShift

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

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

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

Download the new version of chart using the following command in tandem with the newly created folder.

```
$cd dct-x.0.0
$curl -XGET https://dlpx-helm-dct.s3.amazonaws.com/delphix-dct-x.0.0.tgz -o delphix-
dct-x.0.0.tgz
```

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

Copy the values.yaml file from the previous version parallel to the dct-x.0.0 folder.

This values.yaml file contains modified values from the existing previous version of deployment.

Since the Docker Registry (AWS ECR) expires after 12 hours, the Docker Registry should be modified in the values.yaml (from the previous existing version) with the latest password. It can be obtained from https://download.delphix.com[12]. Here are some notes in regards to this step in the process:

- This 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).
- In case a user is using their internal Docker Registry, they should first pull the next version of the Docker images from the Delphix provided registry, using a new password.
- Steps to pull Docker images from the Docker Registry:

Docker login command (password from https://download.delphix.com[13]):

```
$docker login --username AWS --password [PASSWORD] 762392488304.dkr.ecr.us-west-2.ama
zonaws.com/delphix-dct
```

Pull Docker images of DCT Services:

```
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:nginx-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:app-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:data-bookmarks-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:delphix-data-library-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:graphql-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:ui-x.0.0
$ docker pull
```

12 https://download.delphix.com/
13 https://download.delphix.com/

```
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:jobs-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:postgres-x.0.0
$ docker pull
762392488304.dkr.ecr.us-west-2.amazonaws.com/delphix-dct:virtualization-x.0.0
```

The last step is to run the helm upgrade command:

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

4.2.4.2 Factory reset DCT for OpenShift

To clean DCT installation run following command:

```
helm delete dct-services:
```

⚠ This process will delete both services pod and database.

4.3 Docker Compose

4.3.1 Installation and setup for Docker Compose

⚠ Docker Compose should only be used to deploy DCT in an evaluation/testing capacity, and production DCT workloads in Docker Compose are not fully supported. Installations starting on Docker Compose may be migrated to Kubernetes or OpenShift by using the steps in the technical documentation. In-place upgrades from Docker Compose to Kubernetes or OpenShift are not supported.

4.3.1.1 Hardware requirements

The hardware requirements for Data Control Tower are listed below. In addition to these requirements, inbound port 443 must be open for API clients, and outbound port 443 to engines.

CPU: 4-Core
Memory: 2GB
Storage: 50GB
Port: 443

4.3.1.2 Installation requirements (Docker Compose)

DCT **requires** Docker and Docker Compose to run, thus, Linux versions and distributions that have been verified to work with Docker are supported. To see a list of supported distributions, please reference this Docker article[14].

14 https://docs.docker.com/engine/install/#server

This example uses a Docker installation[15] and is completed on an Ubuntu 20.04 VM.

To begin, uninstall any old versions of Docker.

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

Next, update the package lists and install Docker.

```
sudo apt-get update
sudo apt-get install docker.io
```

Last, install Docker Compose[16].

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

Docker-Compose is packaged with Docker engine version 20.10.15 and up.

Running Docker as non-root (optional)

To avoid prefacing the Docker command with sudo, create a Unix group called docker and add users to it. When the Docker daemon starts, it creates a Unix socket accessible by members of the Docker group. See Docker Post Installation[17] documentation for details.

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

4.3.1.3 Unpack and install DCT

Once Docker and Docker Compose are installed, DCT can be installed. Begin by downloading the latest version of the tarball from the Delphix Download site[18]. Next, transfer the file to the Linux machine where Docker is installed. Run the following commands to extract the containers and load them into Docker (where `x.0.0` should be changed to the version of DCT being installed):

```
tar -xzf delphix-dct-x.0.0.tar.gz
for image in *tar; do sudo docker load --input $image; done
```

15 https://docs.docker.com/engine/install/
16 https://docs.docker.com/compose/install/
17 https://docs.docker.com/engine/install/linux-postinstall/
18 https://download.delphix.com/folder

4.3.1.4 Run DCT

To run DCT, navigate to the location of the extracted **docker-compose.yaml** file from the tarball and run the following command. Using `-d` in the command will start up the application in the background.

```
sudo docker-compose up -d
```

Running `docker ps` should show 9 containers up and running:

```
sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED
STATUS         PORTS                    NAMES
75a9df0cae07   delphix-dct-proxy:x.0.0              "/sbin/tini -- /boot…"   7 seconds
ago   Up 4 seconds   0.0.0.0:443->8443/tcp   delphix-dct-proxy:x.0.0
a23f4fbe0220   delphix-dct-app:x.0.0                "java -jar /opt/delp…"   7 seconds
ago   Up 5 seconds                            delphix-dct-app:x.0.0
96ba8018fa03   delphix-dct-data-library:x.0.0       "/usr/bin/tini -- ./…"   7 seconds
ago   Up 5 seconds                            delphix-dct-data-library:x.0.0
8e5b1e671acc   delphix-dct-jobs:x.0.0               "/usr/bin/tini -- ./…"   7 seconds
ago   Up 5 seconds                            delphix-dct-jobs:x.0.0
96049058f025   delphix-dct-data-bookmarks:x.0.0     "/usr/bin/tini -- ./…"   7 seconds
ago   Up 5 seconds                            delphix-dct-data-bookmarks:x.0.0
20d1782cb3bb   delphix-dct-ui:x.0.0                 "node ./index.js"        7 seconds
ago   Up 5 seconds                            delphix-dct-ui:x.0.0
4fae43c79e8d   delphix-dct-virtualization:x.0.0     "/usr/bin/tini -- ./…"   7 seconds
ago   Up 5 seconds                            delphix-dct-virtualization:x.0.0
83d7d661d8a0   delphix-dct-graphql:x.0.0            "/bin/sh -c 'BASE_UR…"   7 seconds
ago   Up 6 seconds                            delphix-dct-graphql:x.0.0
3dded474e28b   delphix-dct-postgres:x.0.0           "docker-entrypoint.s…"   7 seconds
ago   Up 6 seconds   5432/tcp                 delphix-dct-postgres:x.0.0
```

4.3.2 Bootstrapping API Keys

> ⚠ Docker Compose should only be used to deploy DCT in an evaluation/testing capacity.

There is a special process to bootstrap the creation of the first API key. This first API key should only be used to create another key and then promptly deleted, since the bootstrap API will appear in the logs. 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. It also means that the Linux users with permissions to edit the docker-compose file implicitly have the ability to get an API key at any time. There is no mechanism to lock this down after the first bootstrap key is created.

Begin by stopping the application with the following command:

```
sudo docker-compose stop
```

Once the application is stopped, edit the docker-compose.yaml file and modify the following lines to the DCT section, to set the `API_KEY_CREATE` to the string value "true":

```
services:
  gateway:
    environment:
      API_KEY_CREATE: "true"
```

Start DCT again with `sudo docker-compose up`. You will see the following output in the logs for the app container (the key will be different from this example):

```
NEWLY GENERATED API KEY: 1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWaj
uKcBT3
```

Copy the API Key and shut down the DCT app. The API key can now be used to authenticate with DCT. Remember that the API Key value must be prefixed with apk. An example cURL command with the above API Key appears as follows:

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

Edit the docker-compose.yaml file to set the `API_KEY_CREATE` environment variable value back to "false" and restart DCT again with `sudo docker-compose up -d`.

4.3.3 Custom configuration

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

4.3.3.1 Introduction

DCT was designed for users to configure Delphix applications in a way that would meet their security requirements, which handled with a custom configuration. This article provides background information on performing custom configurations, which are referenced throughout DCT articles and sections.

4.3.3.2 Bind mounts

Configuration of DCT is achieved through a combination of API calls and the use of Docker bind mounts[19]. A bind mount is a directory or file on the host machine that will be mounted inside the container. Changes made to the files on the host machine will be reflected inside the container. It does not matter where the files live on the host machine, but the files must be mounted to specific locations inside the container so that the application can find them.

The DCT and proxy containers can both be configured via separate bind mounted directories. Each container requires all configuration files to be mounted to the `/etc/config` directory inside the container. Therefore, it is

19 https://docs.docker.com/storage/bind-mounts/

recommended to create a directory for each container on the host machine to store all of the configuration files and mount them to `/etc/config`. This is done by editing the `docker-compose.yaml`. Under **proxy services**, add a **volumes** section if one does not already exist; this is used to mount the configuration directory on the host to `/etc/config`. For example, if `/my/proxy/config` is the directory on the host that contains the configuration files, then the relevant part of the compose file would look like this:

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

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

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

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

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

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

```
$ docker ps

CONTAINER ID   IMAGE                              COMMAND                  CREATED
STATUS         PORTS                                    NAMES
ac343412492a   delphix-dct-proxy:x.0.0   "/bootstrap.sh"          8 minutes ago   Up
8 minutes   0.0.0.0:443->443/tcp, :::443->443/tcp   dct-packaged_proxy_1
```

In the above example, ac343412492a is the **id**. Run the following command to copy the default files to the bind mount.

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

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

4.3.4 Docker logs

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

DCT leverages the Docker logging[20] infrastructure. All containers log to stdout and stderr so that their logs are processed by Docker. Docker supports logging drivers for a variety of tools such as Fluentd, Amazon CloudWatch, and Splunk to name a few. See Docker documentation here[21] on how to configure them. These changes will need to be made to the docker-compose.yaml file. This link[22] explains how to alter the compose file to adjust the logging driver. For example, if you want to use syslog for the proxy container then it would look like this:

```
services:
    proxy:
        logging:
            driver: syslog
            options:
                syslog-address: "tcp://192.123.1.23:123"
```

4.3.5 Migration topics

4.3.5.1 Migrate to Kubernetes

Overview

Installations starting on Docker Compose may be migrated to Kubernetes by moving the persistent data store using the following steps. In-place upgrades from Docker Compose to Kubernetes are not supported.

During the migration process, there will be a downtime period where the service cannot be used.

Migration Process

Stop DCT services. In order to avoid a situation of losing data, stop serving the upcoming traffic with:

```
~$ docker-compose stop
```

Copy the Postgres Docker volume folder data on a local machine with:

```
~$ mkdir database
~$ docker cp {dbcontainer_Id}:/var/lib/postgresql/data ./database
```

Copy the encryption key Docker volume folder data on a local machine with:

---

20 https://docs.docker.com/config/containers/logging/
21 https://docs.docker.com/config/containers/logging/configure/
22 https://docs.docker.com/compose/compose-file/compose-file-v3/#logging

```
~$ mkdir data_key
~$ docker cp {gateway_container_id}:/data ./data_key
```

Mounted Docker volume folder content for database is copied in database folder on local machine.

Mounted Docker volume folder content for encryption key is copied in the data_key folder on local machine.

Move the copied volume folders (**database** and **data_key** from the previous step) to the Kubernetes host machine where DCT is up and running.

Update the **values.yaml** file to add the list of certificates which were used in the previous DCT version (present in mounted trustStore). Update the deployment with the new **values.yaml** file.

Terminate the proxy pod to stop serving external traffic with:

```
~$ kubectl scale --replicas=0 deployment/proxy -n dct-services
```

Terminate the database to stop internal threads using the database with:

```
~$ kubectl scale --replicas=0 deployment/database -n dct-services
```

Create a dummy pod to access the Persistent Volume. Use the Pod.yaml as an example:

```
apiVersion: v1
kind: Pod
metadata:
Namespace: dct-services
name: dummy-pod
  labels:
    app: dummy-pod
spec:
  containers:
    - image: ubuntu
      command:
        - "sleep"
        - "604800"
      imagePullPolicy: IfNotPresent
      name: ubuntu
  restartPolicy: Always
  volumes:
    - name: gwdatabase-data
      persistentVolumeClaim:
        claimName: gwdatabase-data
```

Followed by this command to actually create the dummy pod:

```
~$ kubectl apply -f pod.yaml -n dct-services
```

Restore previous DCT version volume data with DCT deployed on the Kubernetes setup (in Persistent Volume).

Move the encryption key with:

```
~$cd data_key
~$ kubectl cp data dct-services/{gateway_pod_name}:/
```

Move the Postgres data with:

```
~$cd database
~$ kubectl cp data dct-services/{dummy_pod_name}:/var/lib/postgresql
```

Delete the dummy pod with:

```
~$ kubectl delete pod dummy-pod -n dct-services
```

Start the database pod (scale to 1) with:

```
~$ kubectl scale --replicas=1 deployment/database -n dct-services
```

Delete or patch the gateway pod with:

```
~ % kubectl delete pod {gateway_pod_name} -n dct-services
```

Delete or patch the data-library pod with:

```
~ % kubectl delete pod {data-library_pod_name} -n dct-services
```

Delete or patch the jobs pod with:

```
~ % kubectl delete pod {jobs_pod_name} -n dct-services
```

Delete or patch the data-bookmarks pod with:

```
~ % kubectl delete pod {data-bookmarks_pod_name} -n dct-services
```

Start the proxy service to serve the external service:

4.3.5.2 Migrate to OpenShift

Overview

Installations starting on Docker Compose may be migrated to OpenShift by moving the persistent data store using the following steps. In-place upgrades from Docker Compose to OpenShift are not supported.

> During the migration process, there will be a downtime period where the service cannot be used.

Migration Process

Stop DCT services. In order to avoid a situation of losing data, stop serving the upcoming traffic with:

```
~$ docker-compose stop
```

Copy the Postgres Docker volume folder data on a local machine with:

```
~$ mkdir database
~$ docker cp {dbcontainer_Id}:/var/lib/postgresql/data ./database
```

Copy the encryption key Docker volume folder data on a local machine with:

```
~$ mkdir data_key
~$ docker cp {gateway_container_id}:/data ./data_key
```

> - Mounted Docker volume folder content for database is copied in database folder on local machine.
> - Mounted Docker volume folder content for encryption key is copied in the `data_key` folder on local machine.

Move the copied volume folders (**database** and **data_key** from the previous step) to the Kubernetes host machine where DCT is up and running.

Update the **values.yaml** file to add the list of certificates which were used in the previous DCT version (present in mounted trustStore). Update the deployment with the new **values.yaml** file.

Terminate the proxy pod to stop serving external traffic with:

```
~$ oc scale --replicas=0 deployment/proxy -n dct-services
```

Terminate the database to stop internal threads using the database with:

```
~$ oc scale --replicas=0 deployment/database -n dct-services
```

Create a dummy pod to access the Persistent Volume. Use the Pod.yaml as an example:

```
apiVersion: v1
kind: Pod
metadata:
Namespace: dct-services
name: dummy-pod
  labels:
    app: dummy-pod
spec:
  containers:
    - image: ubuntu
      command:
        - "sleep"
        - "604800"
      imagePullPolicy: IfNotPresent
      name: ubuntu
  restartPolicy: Always
  volumes:
    - name: gwdatabase-data
      persistentVolumeClaim:
        claimName: gwdatabase-data
```

Followed by this command to actually create the dummy pod:

```
~$ oc apply -f pod.yaml -n dct-services
```

Restore previous DCT version volume data with DCT deployed on the Kubernetes setup (in Persistent Volume).

Move the encryption key with:

```
~$cd data_key
~$ oc cp data dct-services/{gateway_pod_name}:/
```

Move the Postgres data with:

```
~$cd database
~$ oc cp data dct-services/{dummy_pod_name}:/var/lib/postgresql
```

Delete the dummy pod with:

```
~$ oc delete pod dummy-pod -n dct-services
```

Start the database pod (scale to 1) with:

```
~$ oc scale --replicas=1 deployment/database -n dct-services
```

Delete or patch the gateway pod with:

```
~ % oc delete pod {gateway_pod_name} -n dct-services
```

Delete or patch the data-library pod with:

```
~ % oc delete pod {data-library_pod_name} -n dct-services
```

Delete or patch the jobs pod with:

```
~ % oc delete pod {jobs_pod_name} -n dct-services
```

Delete or patch the data-bookmarks pod with:

```
~ % oc delete pod {data-bookmarks_pod_name} -n dct-services
```

Start the proxy service to serve the external service:

```
~$ oc scale --replicas=1 deployment/proxy -n dct-services
```

4.3.6 Admin topics for Docker Compose

4.3.6.1 Backup DCT on Docker Compose

⚠ Docker Compose should only be used to deploy DCT in an evaluation/testing capacity.

This article discusses how to backup DCT. The data that needs to be backed up is the Docker volumes used by the DCT container, gwdatabase container, and the configuration directories on the host that are bind mounted to the containers.

The Docker volumes named `{xxx}delphix-dct-data` and `{xxx}delphix-dct-database-data` should be backed up to prevent data loss. This Docker article[23] explains how to backup a data volume.

The bind mount directories containing the configuration files are standard directories that can be backed up as desired. A simple approach would be to create a tar file of the contents. If **/my/config** is the bind mount directory on the host, then this can be done with the following command:

```
tar -czf gateway-backup.tgz /my/config
```

23 https://docs.docker.com/storage/volumes/#backup-restore-or-migrate-data-volumes

4.3.6.2 Deployment upgrade for Docker Compose

⚠ Docker Compose should only be used to deploy DCT in an evaluation/testing capacity.

⚠ DCT versions 2.0.0 through 6.0.2 running on Docker Compose, that are being upgraded to DCT 7.0.0 or later, may experience potential failure to start post-upgrade, resulting in a "permission denied" error in the logs. Operations post-upgrade may also fail with internal errors.
The issue is due to the UID running the application containers changing from UID 1000 (in DCT 2.0.0 through 6.0.2) to UID 1010 (in DCT 7.0.0 and later). Resolving the issues requires the following one-time change and no container restart is required:

1. Change ownership of the volume associated to the gateway container to the new UID:
   `docker exec -u 0 -it <gateway-container-name> chown delphix:delphix /data`
2. If bind mounts have been used to configure DCT, they must grant permission to the user with UID 1010 (GUID 1010) to read/write files, for example:
   `chown 1010:1010 /path/to/nginx/bind/mount`

Introduction

This article describes the procedure to upgrade the DCT version without losing any data. Docker Compose uses the concept of 'project' to create unique identifiers for all of a project's containers and other resources (like volumes, etc.).

Get the current project name and note it down using the following command:

The volume name would be of the format **{project-name}_gateway-data** and **{project-name}_gwdatabase-data.** In the below example, the project name is **delphix-dct**.

```
docker volume ls
DRIVER    VOLUME NAME
local     delphix-dct_gateway-data
local     delphix-dct_gwdatabase-data
```

Bring down DCT services using the following command:

```
docker compose down
```

Refer to the **Installation and Setup** article to download and extract the new release tarball, then load Docker images.

Navigate to the extracted directory which contains the **docker-compose.yaml** file. By default, Docker Compose uses the extracted folder name as **project-name**.

⚠ Edit the docker-compose.yaml file. Changes made to the docker-compose.yaml prior to upgrade file must be applied to the newly extracted docker-compose.yaml file.

With that, either rename the extracted folder to match the project-name and run:

```
docker compose up -d
```

OR run the below command with the project-name noted above from step #1 above

```
docker compose -p <project-name> up -d
```

If the -p argument is used to deploy DCT services, then the corresponding command to bring down the DCT services would be:

`docker compose -p <project-name> down`

4.3.6.3 Factory reset DCT for Docker Compose

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

This article explains how to factory reset DCT. Factory resetting means deleting all of the configuration and data associated with DCT. Perform this step only if you are absolutely sure about this and understand the implications.

Bring all of the DCT services down with this command:

```
docker compose down
```

List all Docker volumes being used and note down the volume names:

```
docker volume ls
DRIVER  VOLUME NAME
local   dct_gateway-data
local   dct_gwdatabase-data
```

Delete the Docker volumes that are listed from the previous command:

```
docker volume rm dct_gateway-data
docker volume rm dct_gwdatabase-data
```

4.4 Engines: connecting/authenticating

4.4.1 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.

4.4.2 Truststore for HTTPS

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.

4.4.3 Authentication with engine

All authentication with the Delphix Engine is done with the username and password of a domain admin engine user. There are two methods of storing these credentials with DCT. They can either be stored and encrypted on DCT itself or retrieved from a password vault. We recommend fetching the credentials from a vault. Currently only the HashiCorp vault is supported.

4.4.4 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.

4.4.4.1 Vault authentication and registration

First, DCT needs to be able to authenticate with the vault. DCT supports the Token[24], AppRole[25], and TLS Certificates[26] authentication methods. This is done by passing a command to the HashiCorp CLI[27]. 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.

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"
   }
```

4.4.4.2 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>/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.

---

24 https://www.vaultproject.io/docs/auth/token
25 https://www.vaultproject.io/docs/auth/approle
26 https://www.vaultproject.io/docs/auth/cert
27 https://www.vaultproject.io/docs/commands

4.4.4.3 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>/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"
   }
}
```

4.4.5 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.

4.4.5.1 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.

4.5 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[28] 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).

4.5.1 API keys

4.5.1.1 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[29] with type **apk**. 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(see page 75) has not been performed yet and the risk is comprehended, you may disable the check in the HTTP client. For instance, this can done with cURL using the **--insecure** flag. The cURL version must be 7.43 or higher.

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

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>/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"
}
```

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>/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>/v2/management/accounts/' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk <your API key>'
```

4.5.2 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
OiI4IiwiZXhwIjoxNjYyNTUyMzI3LCJpYXQiOjE2NjI0NjU5MjcsInVzZXJuYW1lIjoic29tZS11c2VybmFtZ
SJ9.Cx_hGU9noyWS6mtK6gjsA85FTgJRQgyJizR5t_akNps",
    "token_type":"Bearer",
    "expires_in":86400
}
```

The access token can be used as HTTP Authorization request header[30] with type **Bearer**. A cURL example using the access token retrieved above would appear as:

30 https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization

```
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”.

4.5.2.1 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).

4.5.2.2 Understanding password policies

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

- **min_length**: A password must be longer than this length.
- **reuse_disallow_limit**: 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**: A password must have at least one capital letter.
- **lowercase_letter**: A password must have at least one lower case letter.
- **digit**: A password must have at least one digit.
- **special_character**: A password must have at least one special character, such as #, $, !
- **disallow_username_as_password**: A password should not be the same as the user name.
- **maximum_password_attempts**: The number of allowed attempts for incorrect password, after which the account gets locked.

4.5.2.3 Default password policy

By default, DCT does not enforce any password policy.

4.5.2.4 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.

4.5.2.5 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}'
```

4.5.3 LDAP/Active Directory

4.5.3.1 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.

Configuring authentication

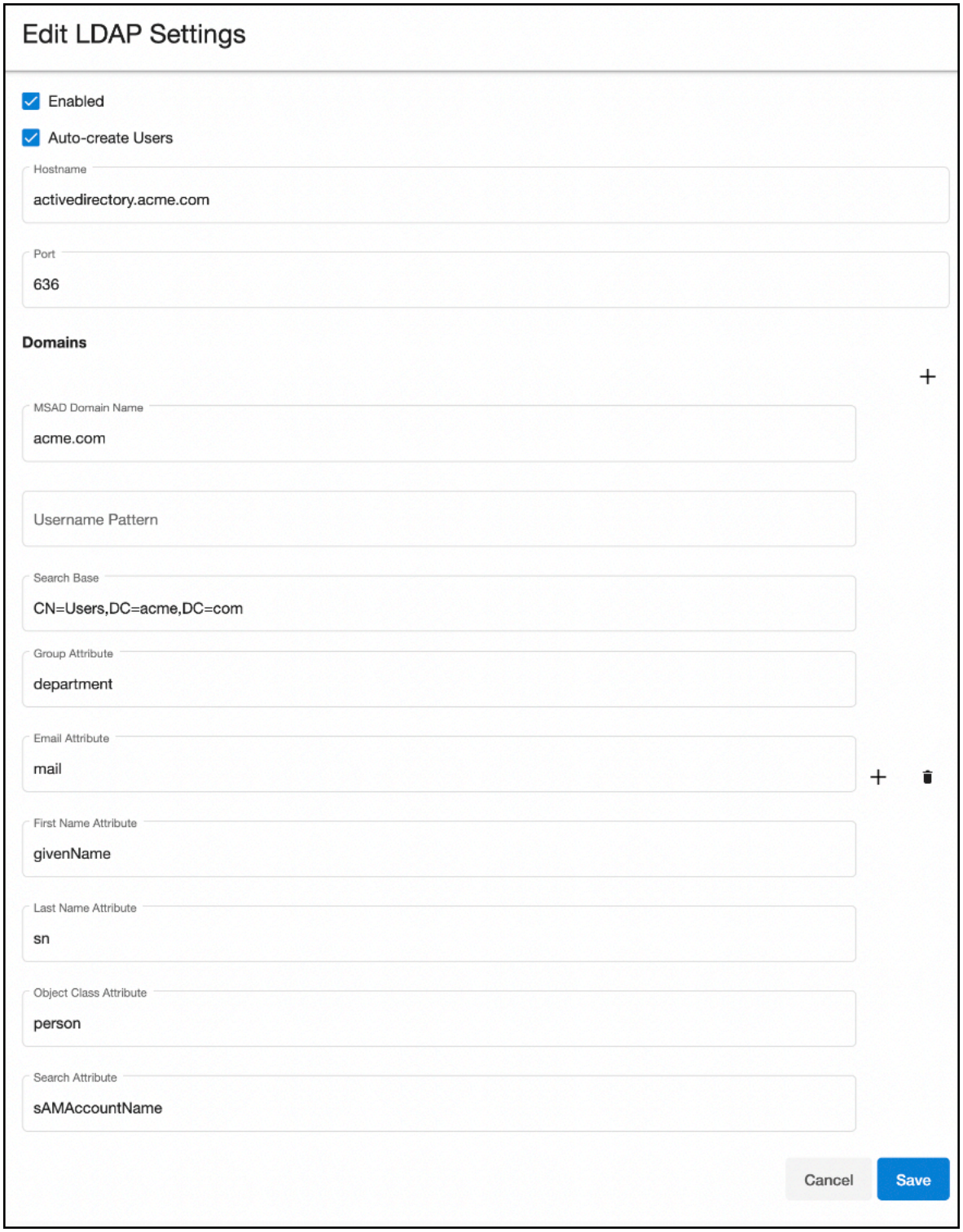
The following attributes must be set to configure LDAP/Active Directory authentication.

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, trustore_file_name, 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[31].
[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.

31 mailto:john@mycompany.co



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"
 }]
}'
```

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.

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.

4.5.4 SAML/SSO

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, Ping federate, Okta, OneLogin, etc.). It only applies to web browser based interaction, and cannot be used for API access (scripting, integration).

Setting up SAML/SSO requires configuration changes both in the Identity Provider and DCT, so that trust can be established across both products.

When using SAML/SSO, DCT will uniquely identify accounts by email address, so make sure that 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 authentication method for web browser interaction, perform the SAML/SSO configuration steps below and disable LDAP/Active Directory and Username/Password authentication. Note that API Key based authentication cannot be entirely disabled, but only administrators can create accounts with API keys.

4.5.4.1 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	• Alternative name depending on 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	https://<dct-hostname>/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.

4.5.4.2 DCT SAML/SSO setup

Once the configuration has been performed at the Identity provider, use the saml-config API endpoint to configure DCT accordingly. If DCT has network access to the Identity Provider server, and the Identity Provider provides a "metadata URL", you can point DCT directly to the metadata URL. Otherwise, for instance when a firewall blocks network access from DCT to the Identity Provider, 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.

The metadata URL is sometimes called "App Federation Metadata URL", and is sometimes only known by reading the Identity Provider's product documentation (for instance Active Directory Federation Services, or ADFS, publishes the metadata URL at https://<hostname>/federationmetadata/2007-06/federationmetadata.xml).

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).

Example 1: With network access, point DCT to the metadata URL.

```
curl --location --request GET '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_url": "<idp-metadata-url>",
}'
```

Example 2: 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[32] installed, this can be achieved with the following command: `jq --slurp --raw-input <<< 'xml-metadata-here'`

4.5.4.3 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 and the the navigation bar can be used to go to the desired page. The session expires 24 hours after login.

4.5.4.4 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.

32 https://stedolan.github.io/jq/

4.5.4.5 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://<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 version="1.0" encoding="UTF-8"?>
<saml2:Assertion ID="id979239831676038211571805l6" 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="2022-12-01T1
0:12:12.857Z">
        <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="id166
9889232855.2084756273">
        <saml2:AuthnContext>
            <saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Passwo
rdProtectedTransport</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:st
ring">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:st
ring">Doe
            </saml2:AttributeValue>
        </saml2:Attribute>
        <saml2:Attribute Name="http://schemas.xmlsoap.org/claims/Group" NameFormat="u
rn: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:st
ring">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:st
ring">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"
        }
    ]
}
```

4.5.4.6 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**

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[33] 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.

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.

4.6 Replace HTTPS certificate for DCT

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

```
proxy:
    crt:<certificate_value>
    key:<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 mycertfile.pem | base64 -w 0`
- To generate the Base64 encoded value of the key:
  `cat mykey.key | base64 -w 0`

Generating a new TLS certificate and 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).

4.7 External database support

4.7.1 Overview

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

33 mailto:john@mycompany.co

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.

4.7.2 Requirements

- **Database type:** PostgreSQL
- **Min supported version:** 13.4
- **Max supported version:** 14.7

4.7.3 Setup

4.7.3.1 PostgreSQL database setup

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

> ⚠ 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.

Shell

```
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;
```

4.7.3.2 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.

Shell

```
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
```

4.7.4 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` 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`, as DCT would not be able to decrypt some of the data in the database. The encryption key in `gateway-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.

4.7.5 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 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.

4.7.6 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": "14.7",
  "database_product_name": "PostgreSQL",
  "major_version": 14,
  "minor_version": 7,
  "min_supported_major_version": 13,
  "min_supported_minor_version": 4,
  "max_supported_major_version": 14,
  "max_supported_minor_version": 7,
  "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).

4.8 DCT data backup and recovery

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

4.8.1 Data backup of Persistent Volumes used by DCT

Based on customer need or backup policy, they need to take a backup of data in Persistent Volumes. In DCT, gateway and database services are using Persistent Volume hence customer need to take backup of data folder mounted on mounted path.

Take the backup (copy) of the data folders for the **gateway** Persistent Volume:

```
kubectl cp {gateway_pod_name}:/data data -n dct-services
```

Take the backup (copy) of the data folders for the **database** Persistent Volume:

```
kubectl cp {database_pod_name}:/var/lib/postgresql/data/ dct_database -n dct-services
```

4.8.2 Restore data backup in a new DCT setup

Deploy DCT services with the same version for which data back-up was taken. This deployment will create a new Persistent Volume data folder for the cluster.

Use the following steps to copy the backup data in this new deployment.

First, stop all services.

Terminate `proxy` pod to stop serving external traffic:

```
kubectl scale --replicas=0 deployment/proxy -n dct-services
```

Terminate the database to stop internal threads using the database:

```
kubectl scale --replicas=0 deployment/database -n dct-services
```

Terminate the gateway to stop using Persistent Volume data:

```
kubectl scale --replicas=0 deployment/gateway -n dct-services
```

Create a `dummy` pod to access the **Persistent Volume** using pod.yaml, as shown below.

```
apiVersion: v1
kind: Pod
metadata:
Namespace: dct-services
name: dummy-pod
  labels:
    app: dummy-pod
spec:
  containers:
    - image: ubuntu
      command:
        - "sleep"
        - "604800"
      imagePullPolicy: IfNotPresent
```

```
      name: ubuntu
  restartPolicy: Always
  volumes:
    - name: gwdatabase-data
      persistentVolumeClaim:
        claimName: gwdatabase-data
```

Create the `dummy` pod with:

```
kubectl apply -f pod.yaml -n dct-services
```

Copy the backup Persistent Volume data into a new cluster.

Move the encryption key:

```
kubectl cp data dct-services/{gateway_pod_name}:/data
```

Move the Postgres data:

```
kubectl cp dct_database dct-services/{dummy_pod_name}:/var/lib/postgresql/data
```

Delete the `dummy` pod.

```
kubectl delete pod dummy-pod -n dct-services
```

Start the `database` pod (scale to 1):

```
kubectl scale --replicas=1 deployment/database -n dct-services
```

Restart all services.

Delete or patch the `gateway` pod:

```
kubectl delete pod {gateway_pod_name} -n dct-services
```

Delete or patch the `data-library` pod:

```
kubectl delete pod {data-library_pod_name} -n dct-services
```

Delete or patch the `jobs` pod:

```
kubectl delete pod {jobs_pod_name} -n dct-services
```

Delete or patch the `data-bookmarks` pod:

```
kubectl delete pod {data-bookmarks_pod_name} -n dct-services
```

Start the proxy service to serve the external service:

```
kubectl scale --replicas=1 deployment/proxy -n dct-services
```

4.9 Exporting DCT logs to Splunk

4.9.1 Overview

This article provides some tips for configuring DCT (running on Kubernetes) to send logs to Splunk and extract useful information in Splunk.

4.9.2 Setting up a Splunk instance

Authenticate with Splunk via the web portal and install the third-party Monitoring Kubernetes[34] app directly via the Splunk UI, then enable HTTP Event Collector in Splunk and save the HTTP Event Collector token for future use.

4.9.3 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[35] 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.

```
global:
 logLevel: info
 splunk:
   hec:
```

34 https://splunkbase.splunk.com/app/3743
35 https://faun.pub/logging-in-kubernetes-using-splunk-c2785948fdc0

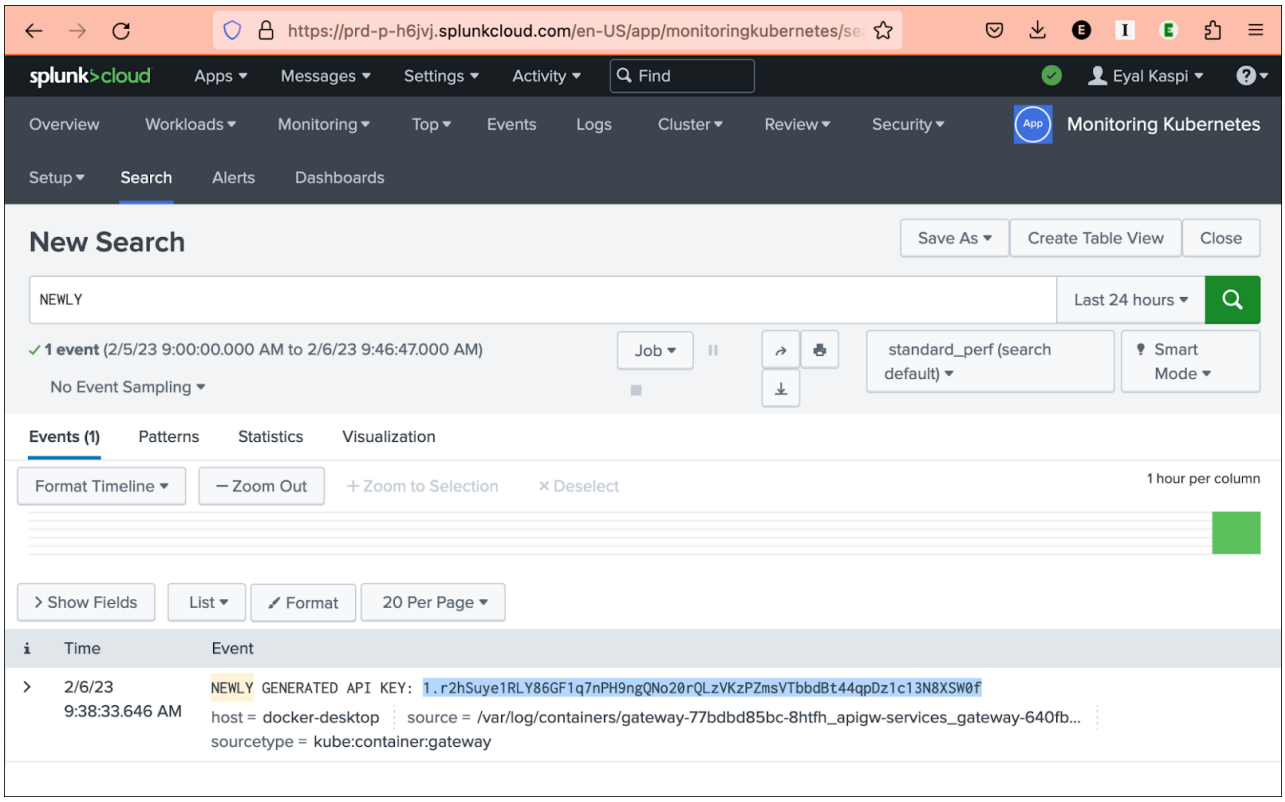
```
    # host is required and should be provided by user
    host: <insert-splunk-http-event-collector-hostname-here>
    # port to HEC, optional, default 8088
    port:
    # token is required and should be provided by user
    token: <insert-token-here>
```

Install the helm chart and and after a few minutes DCT logs will be visible in Splunk.

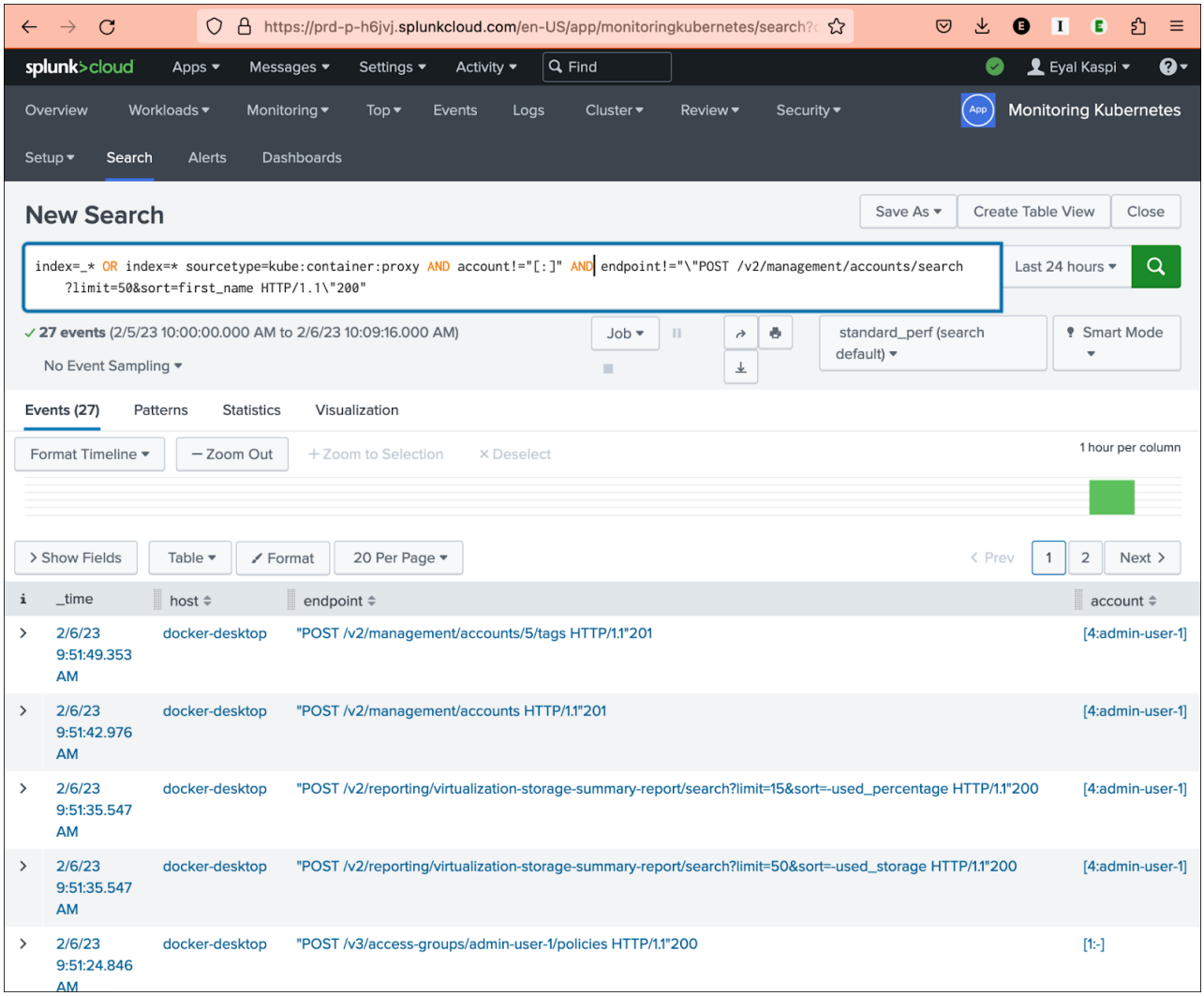
```
helm install splunk-connect-for-kubernetes . -f values.yaml --set splunk-kubernetes-logging.fullnameOverride=splunk-logging
```

4.9.4 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.



4.10 Generating a support bundle

4.10.1 Find the “collect_bundle.sh” script

- The support bundle tar file is available on the downloads site[36].
- 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
```

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

4.10.2 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...
....
```

4.10.3 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...
....
```

4.10.4 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:

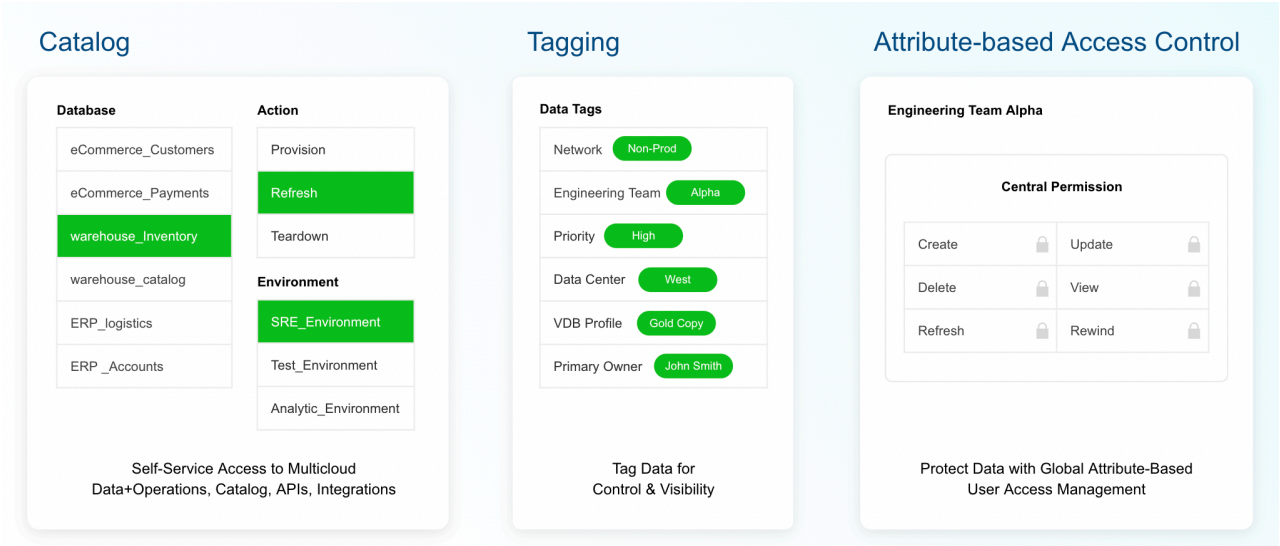
- DCT logs for all of the containers.
- A java heap dump, .hprof, if one exists.
- A java thread dump and memory stats.
- The output of **docker stats**, if running in Docker-Compose.
- The output of **cpuinfo**, **meminfo**, and **mpstat** for each container, if running in Kubernetes.
- The output of **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.

5 Data governance

5.1 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.



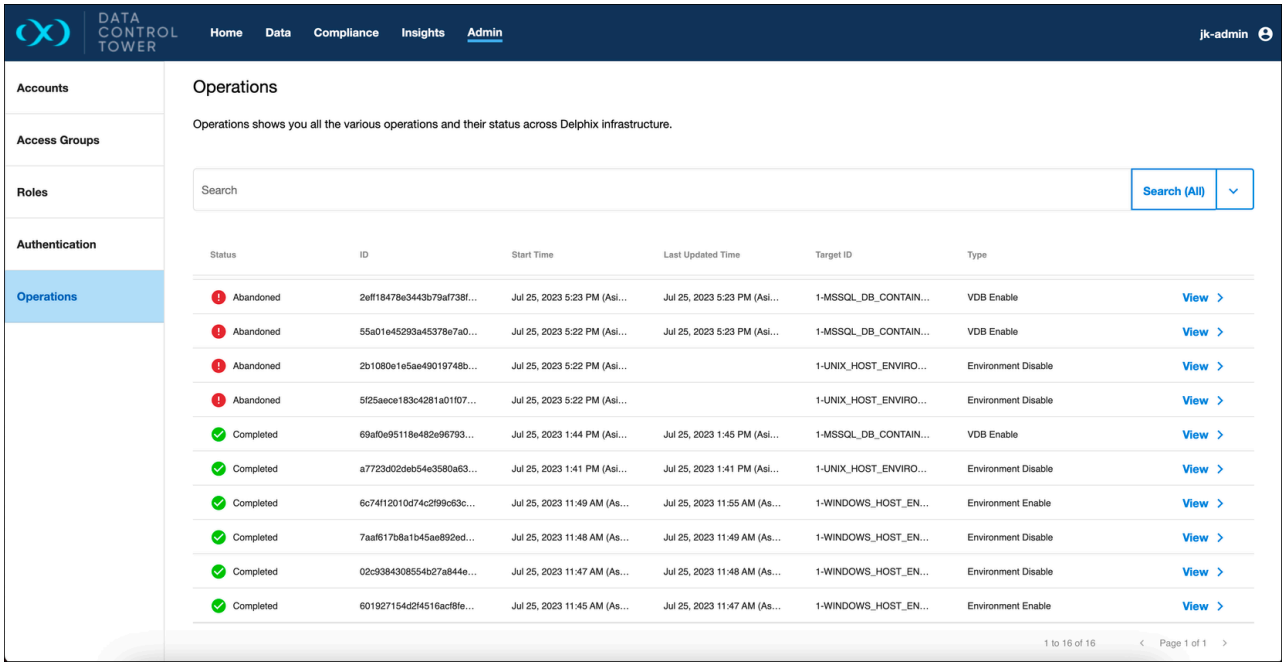
This section contains configurations handled under the **Admin page** in the DCT interface.

5.1.1 Operations page

5.1.1.1 Operations

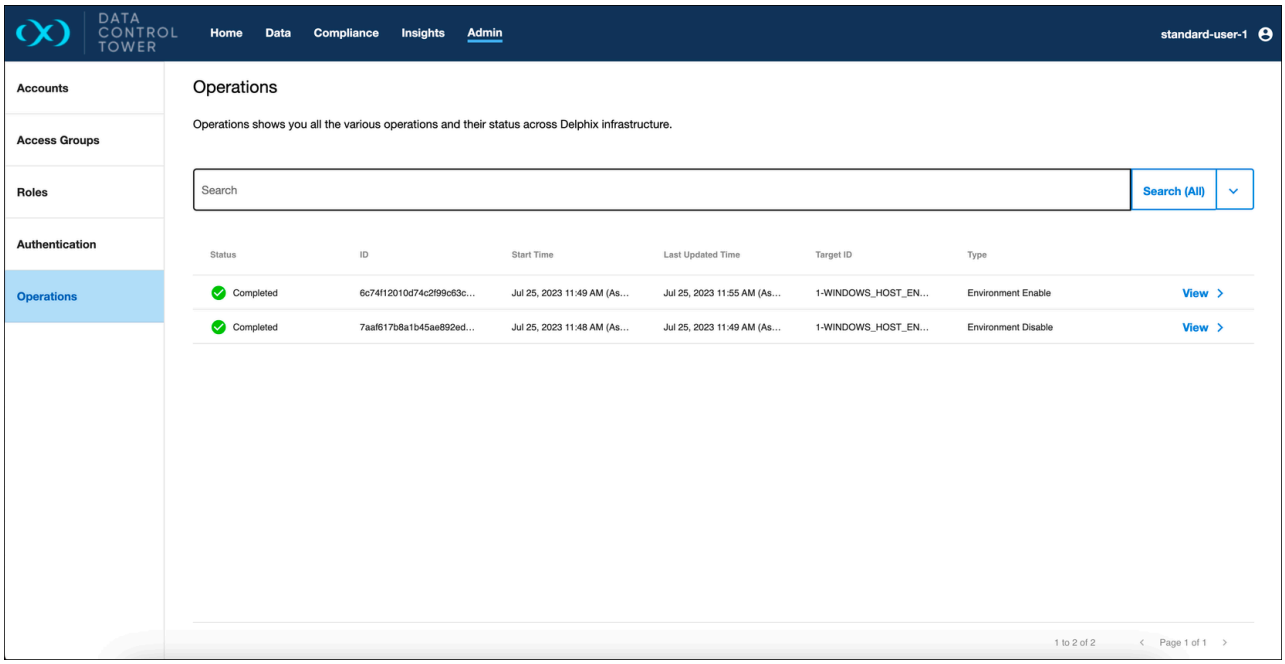
An Operations page is available under the **Admin** menu to display the list of all the DCT Jobs across Delphix infrastructures. It only shows operations that the user has access to.

This page displays the relevant details for the operation, such as status, 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.

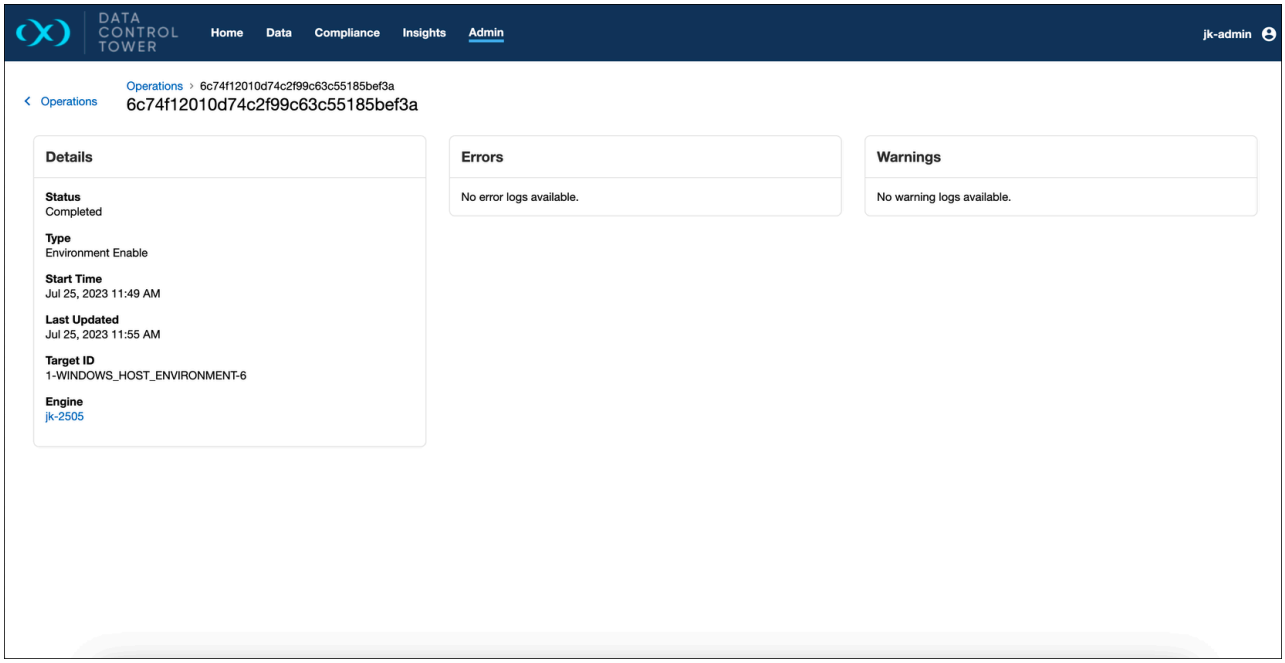


The capture **above** represents an **admin user** view of the Operations page.

The capture **below** represents a **non-admin user** view of the same page.



Further details regarding the job, such as target id, error, or warning logs can be viewed by clicking on the **View** link, which navigates to the details page.



5.1.2 Tags

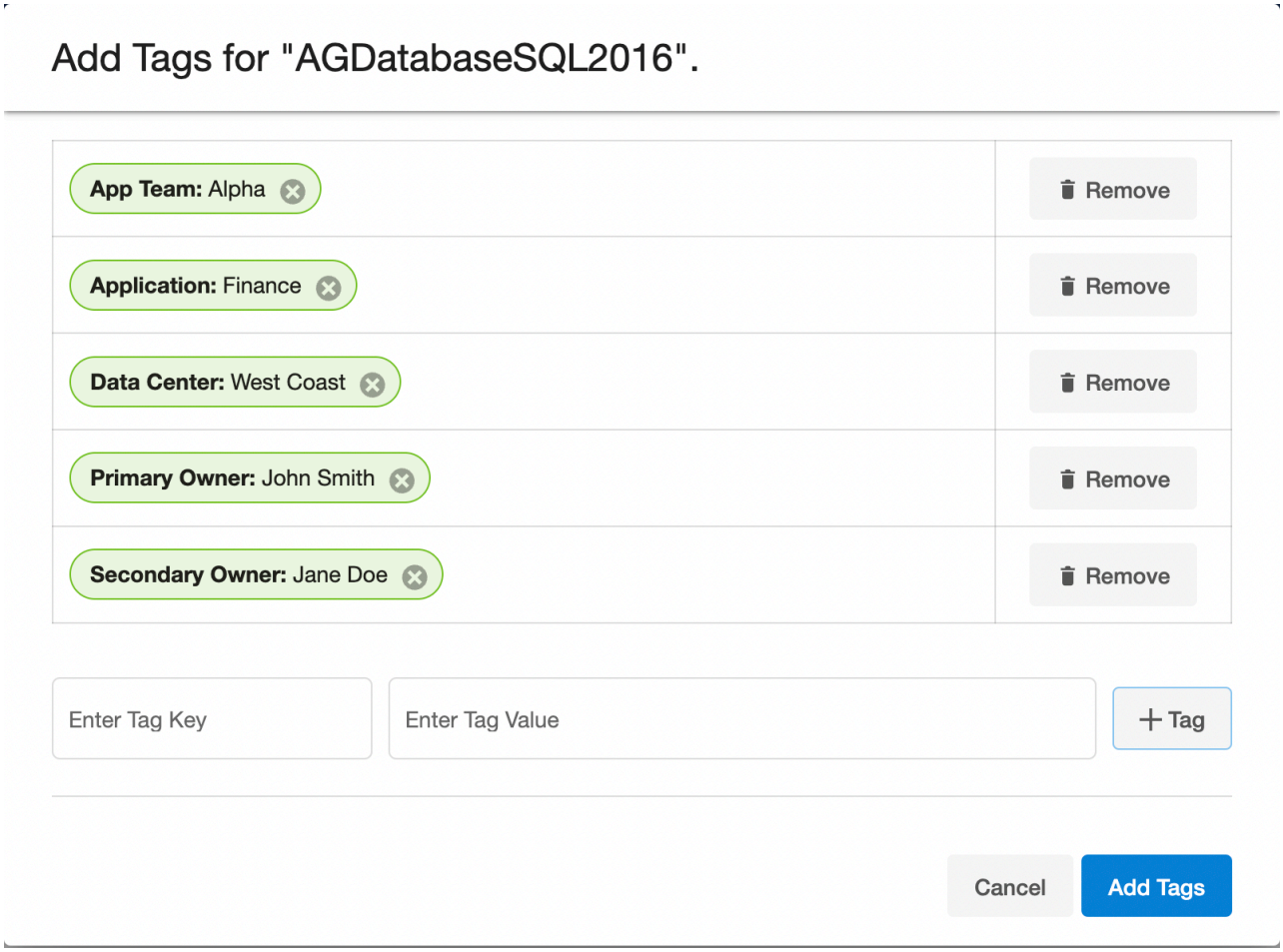
5.1.2.1 Tags management

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.

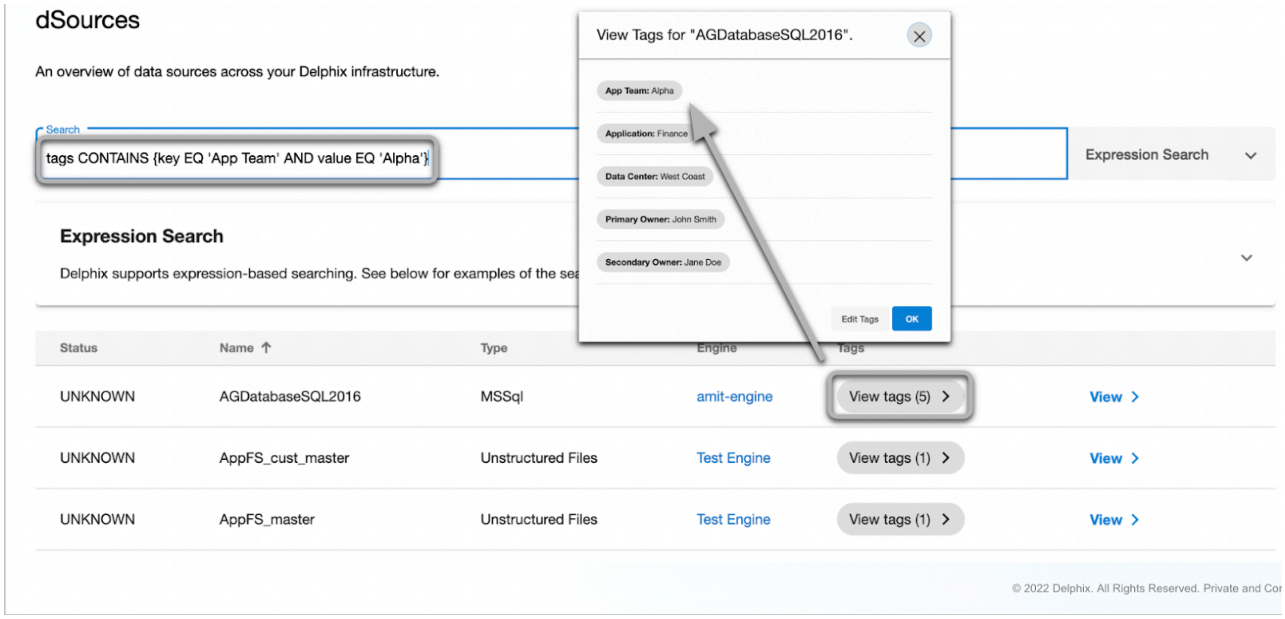
Tags are individual attributes on every object exposed in DCT; from VDBs, to compliance jobs, and even users. There are no limits on tag count per object and character limits are set for flexibility to enable robust grouping.

5.1.2.2 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:

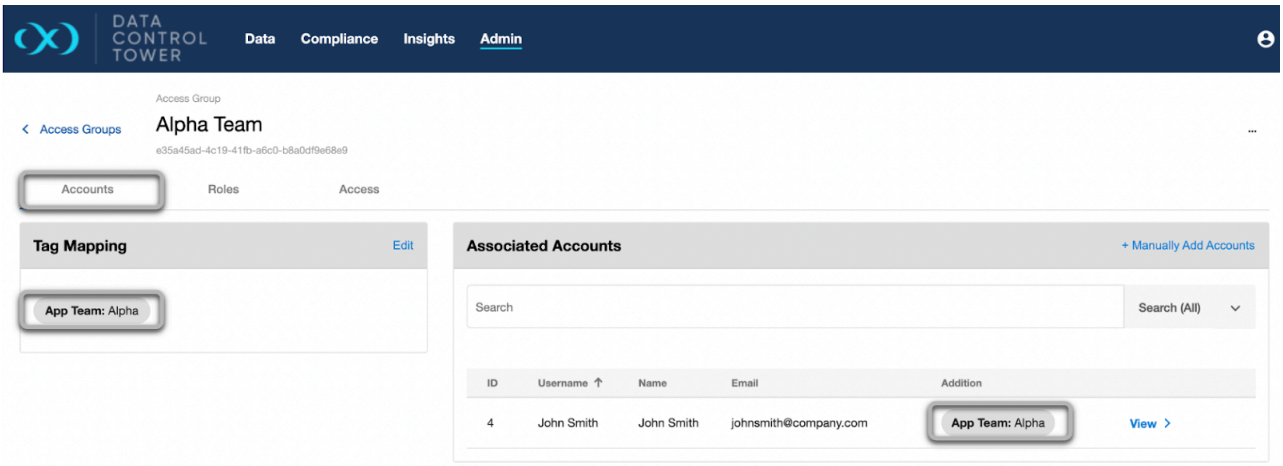


DCT tags enable complex searching to enable intelligent reports. A demonstration using the above example dSource and using expression-based search to filter dSources with the {App Team: Alpha} tag.

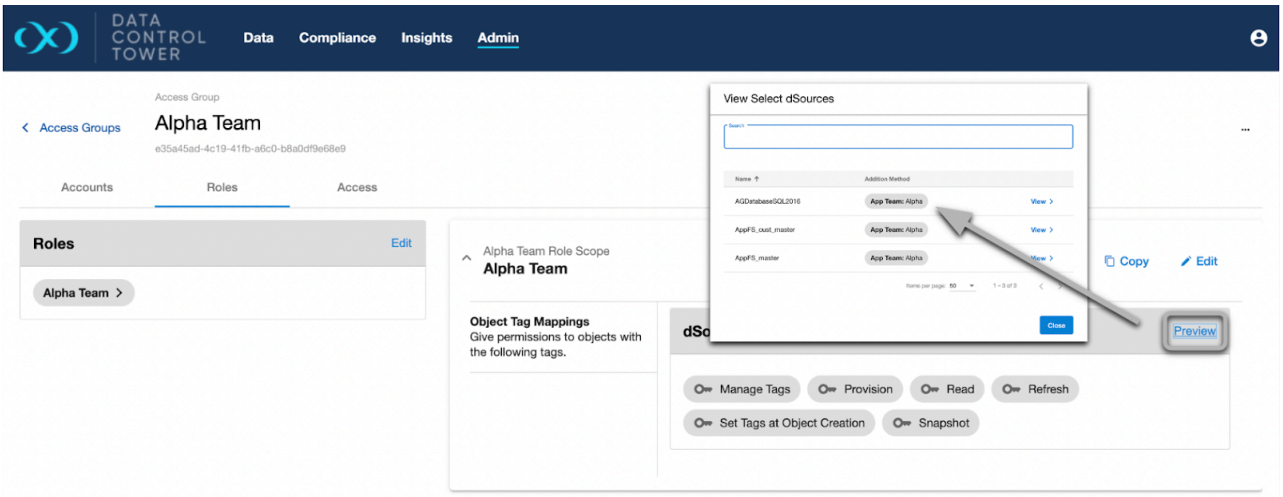


5.1.2.3 Tags powering attribute-based Access Control

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.



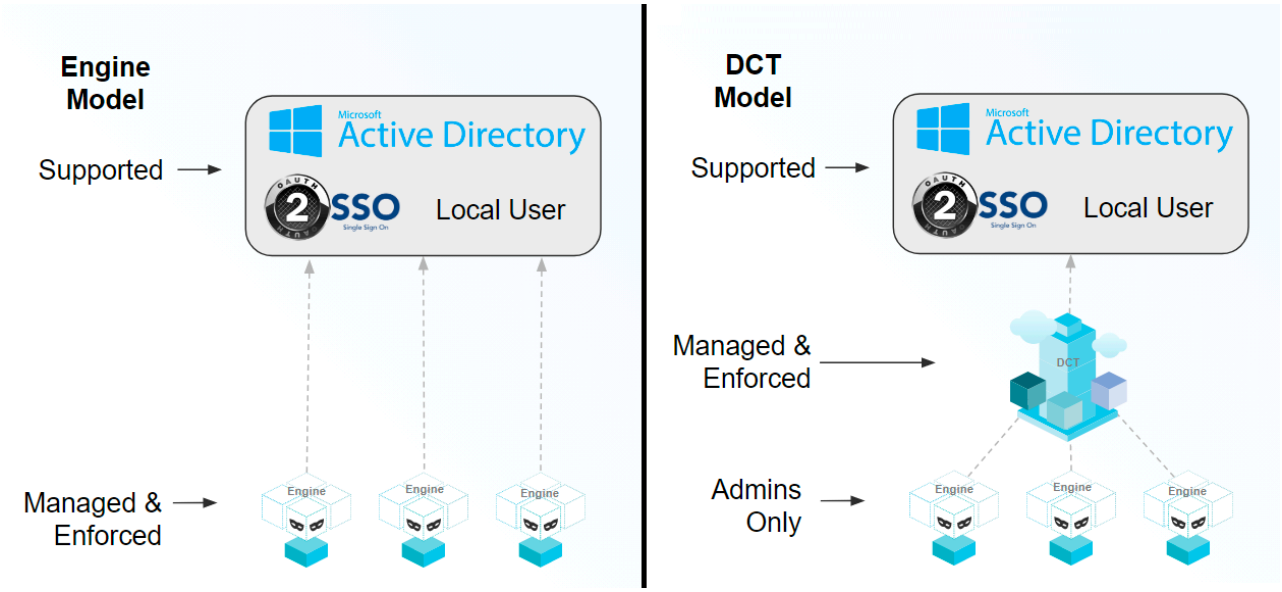
5.1.3 Access Control

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.



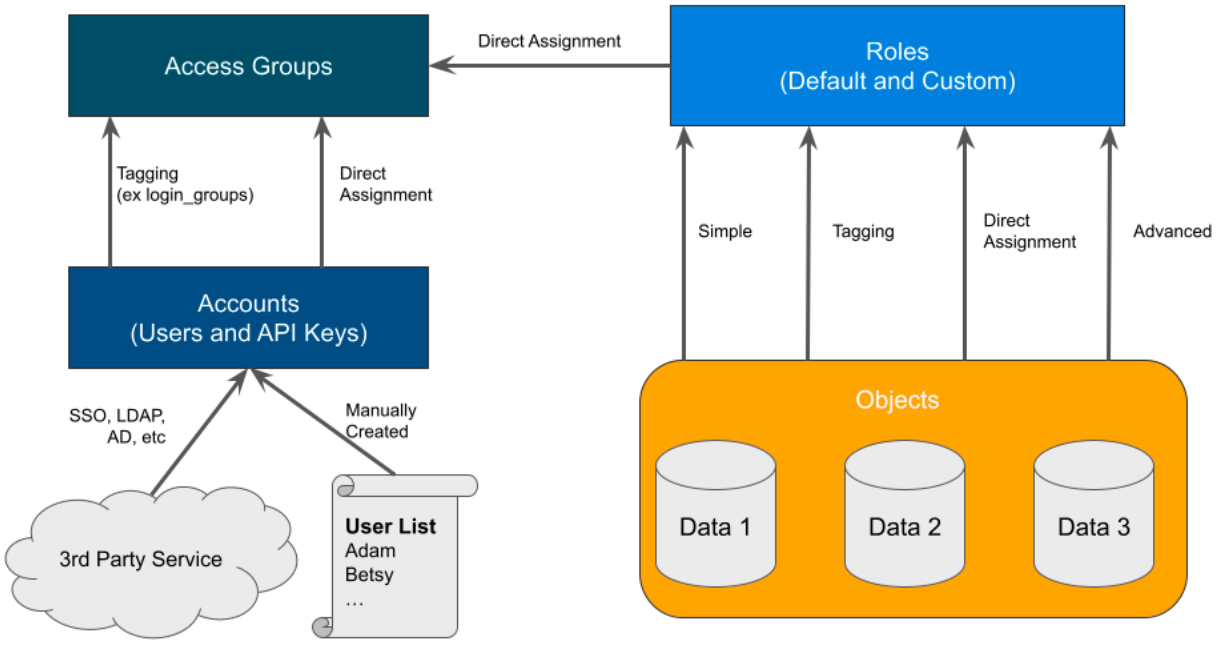
5.1.3.1 Access model overview

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.

Entity	Description	Managed By
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.



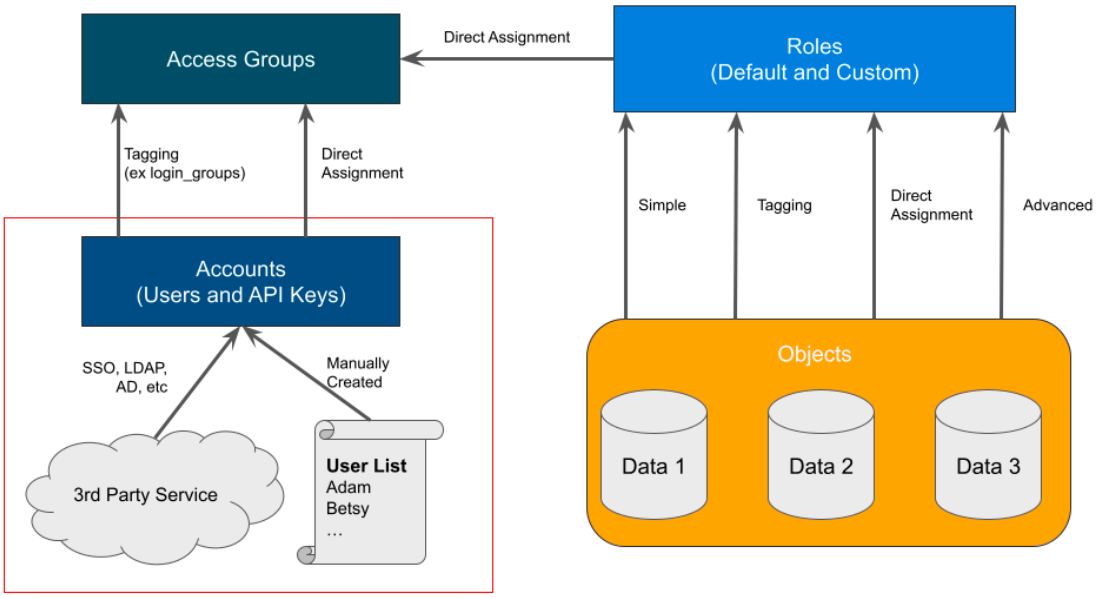
Understanding your team structure is imperative to identify the best access model. Usually, organizations have existing groupings defined in their Identify Provider (IdP). These groups are typically organized in one of two ways (a) a team dedicated towards a central goal (such as a product development 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[37], will produce a streamlined rollout.

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

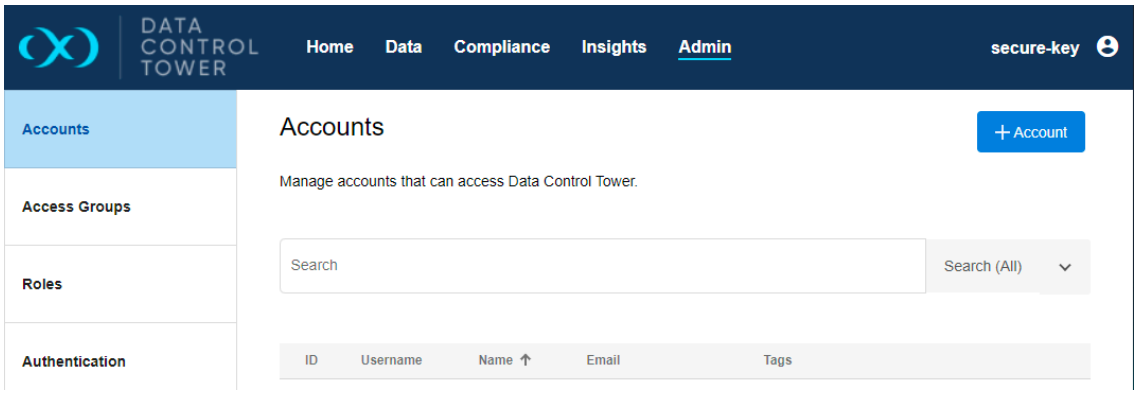
5.1.3.2 Accounts: Manual, LDAP/AD, or SSO/SAML

Accounts: Manual, LDAP/AD, or SSO/SAML

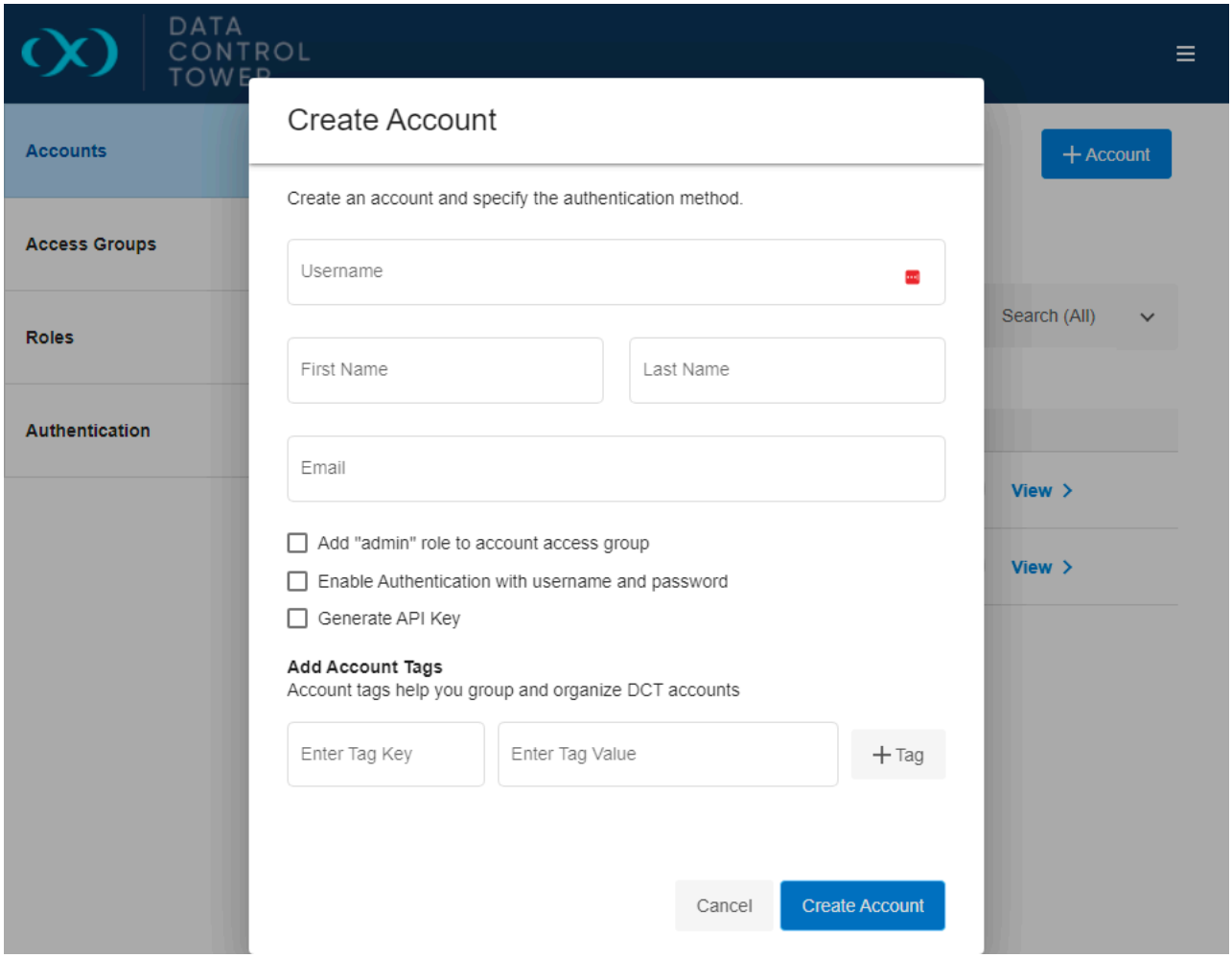
Goal: Import or create user accounts. Complete either the Manual or LDAP/SOO configuration.



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.

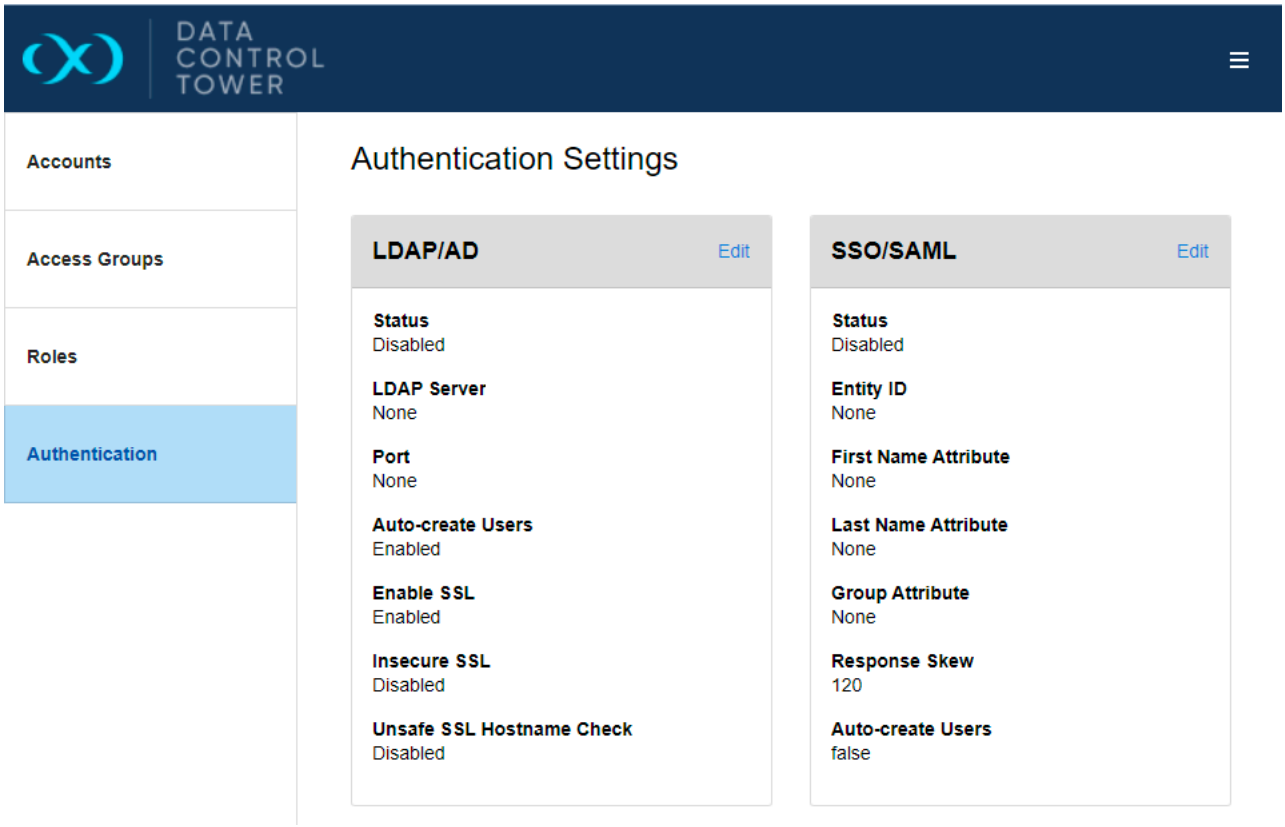
LDAP/Active Directory or SSO/SAML (3rd Party Service)

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[38]
- SSO/SAML Directions[39]

38 https://dct.delphix.com/docs/ldapactive-directory-3
39 https://dct.delphix.com/docs/samlsso-2



Once configured, Accounts will be automatically created when a user successfully logs in.

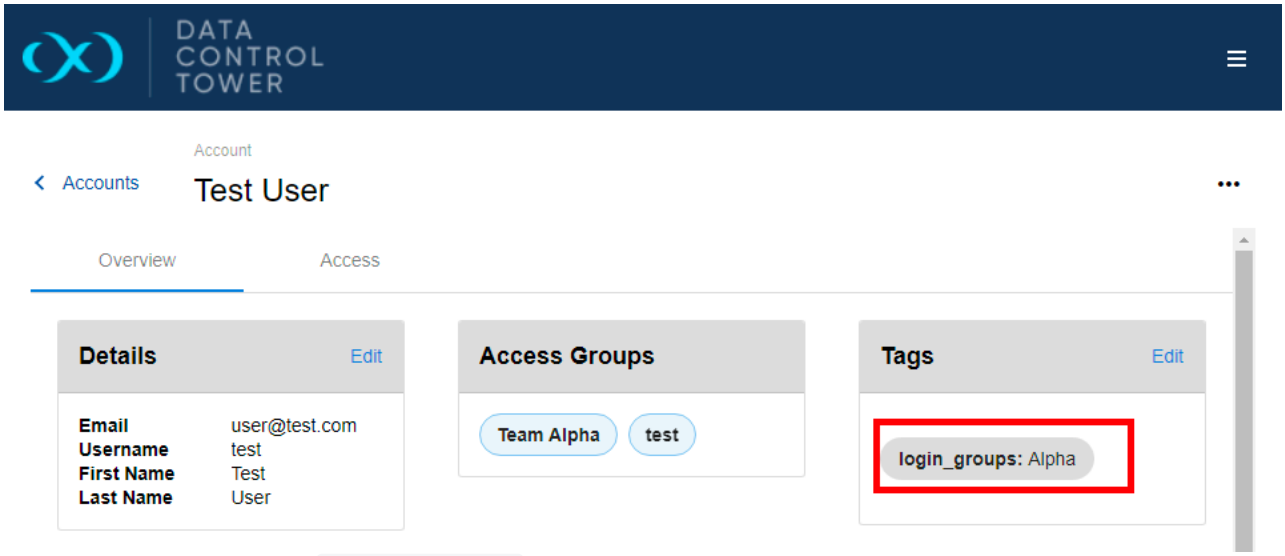
> This is functionally different from the old Engine model. Previously, the Account was created manually before they could log in.

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[40]

40 https://dct.delphix.com/docs/configure-ldap-ad-groups-2

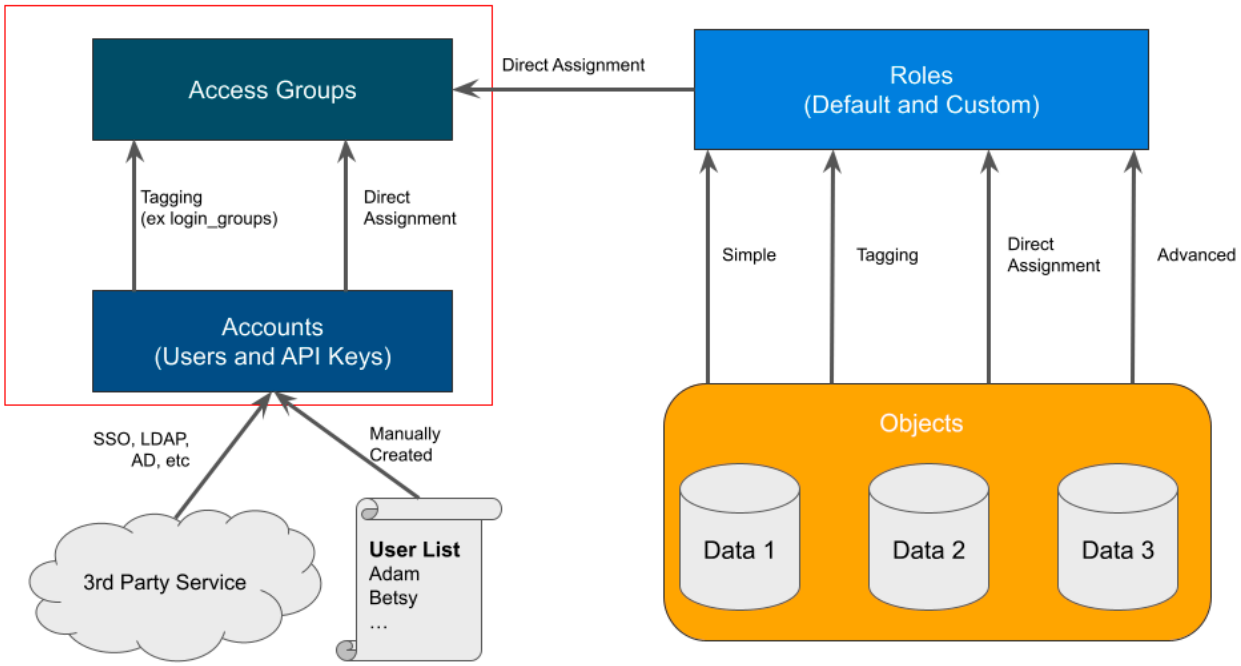


Example Account with the `login_groups` tag.

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.

5.1.3.3 Access Groups: Creation and account assignment

Goal: Create an Access Group and assign Accounts directly or through Tags.

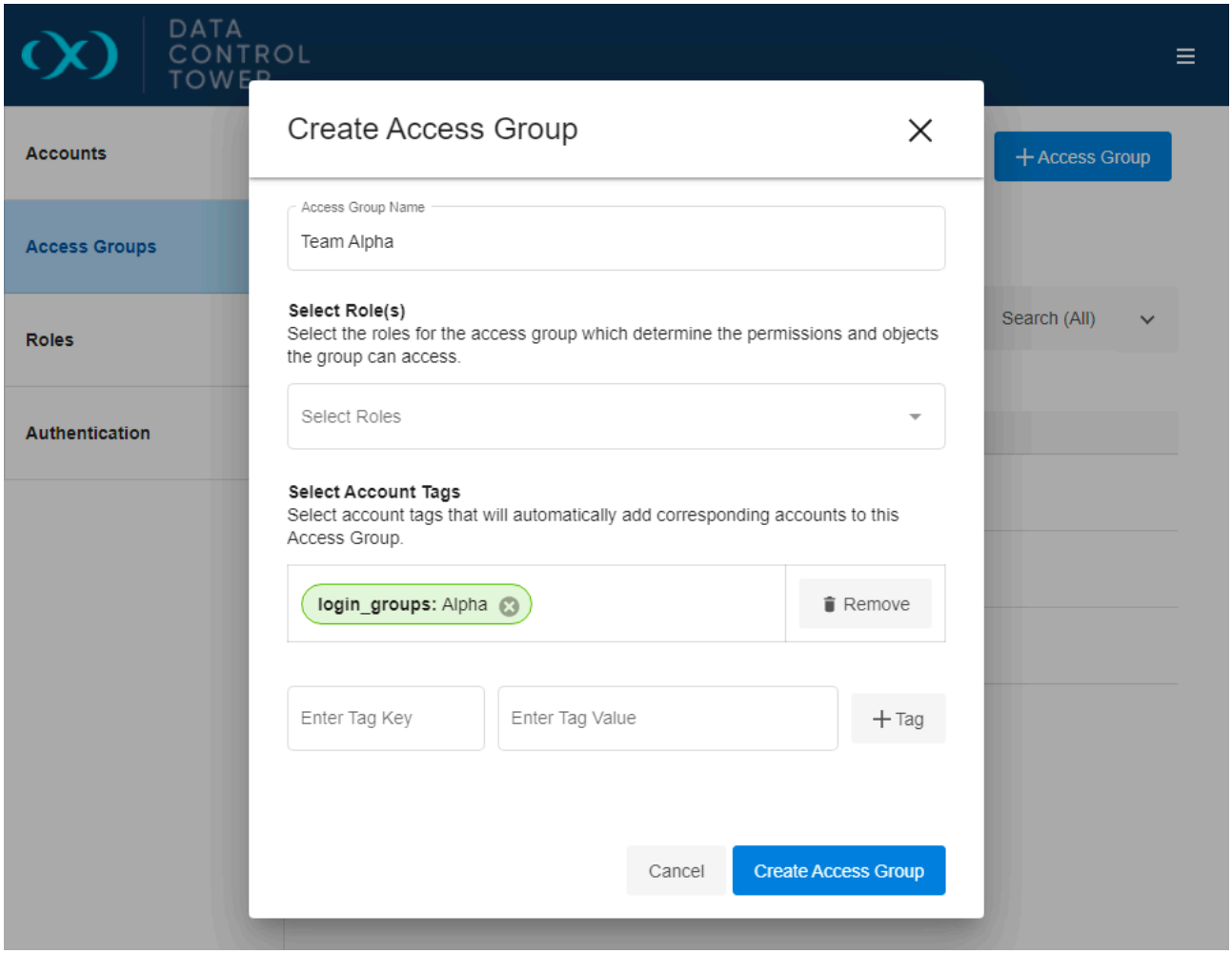


Access Group Creation

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".

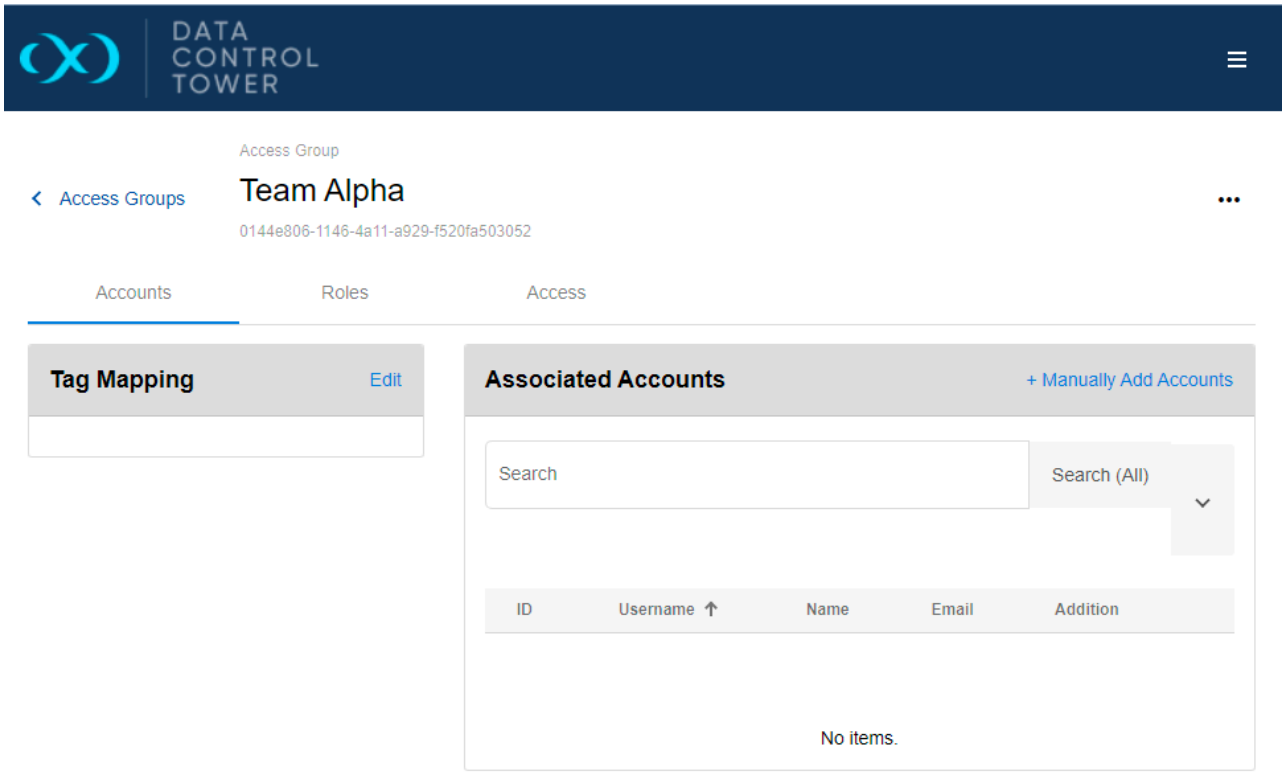
You can also select Roles if you already know which should be applied. Otherwise, ignore it.

Submit the form once you are happy with your new group.



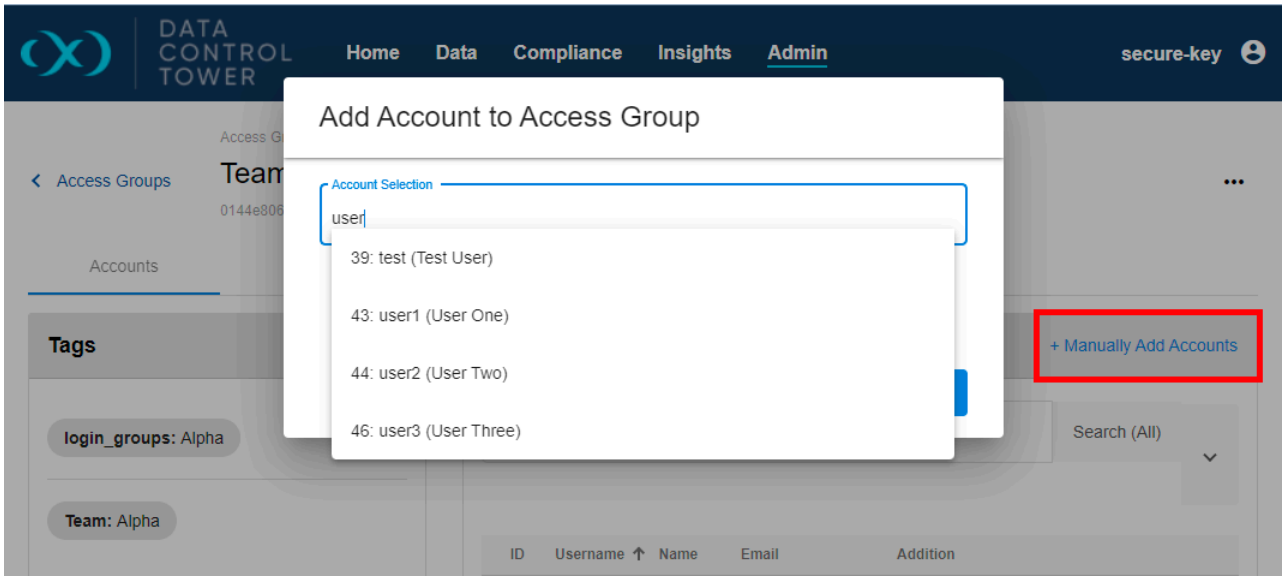
Unlike an Account, you can specify the `login_groups` tag on an Access Group shown in the picture above.

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

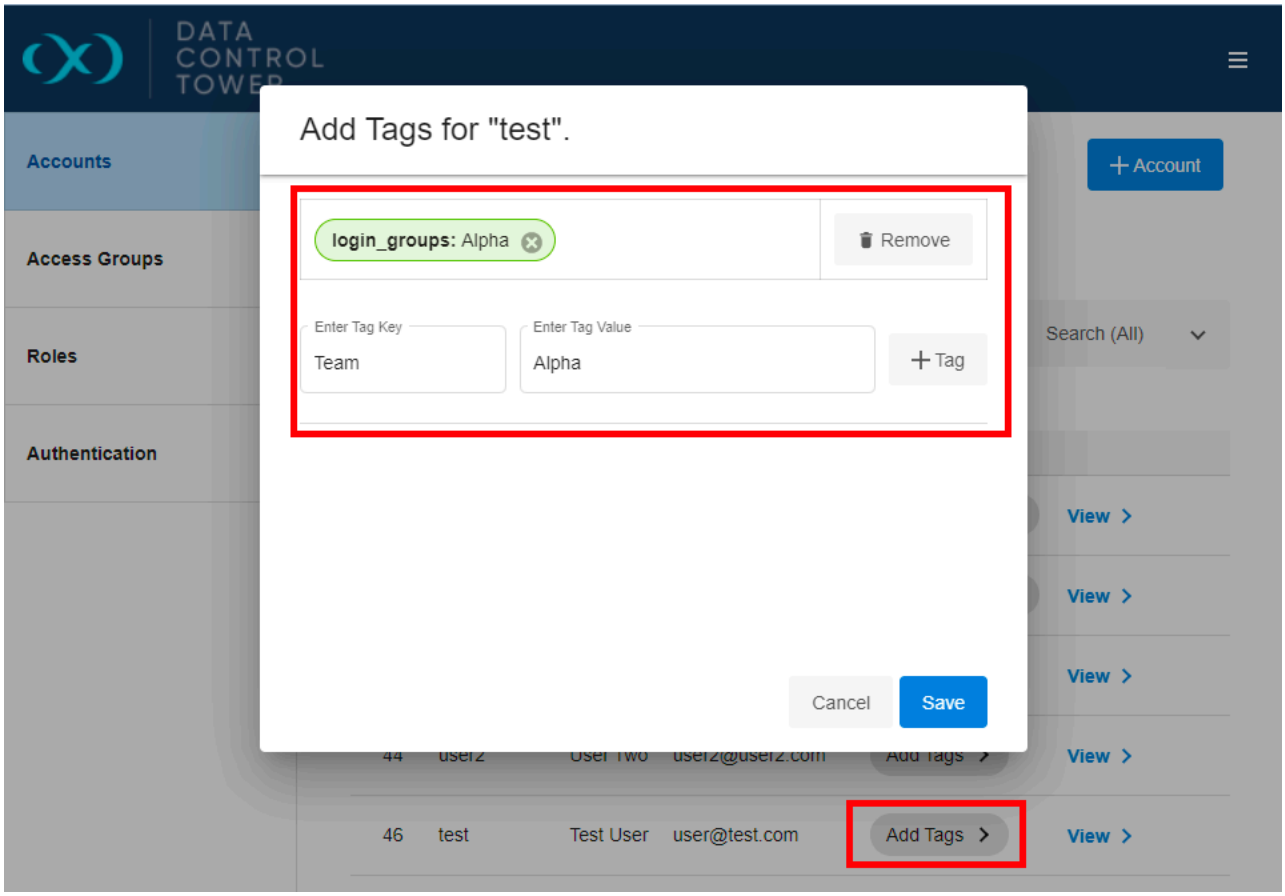
Select the "+ Manually Add Accounts" button, select the desired Account, and then "Add Account". Immediately, you'll see it presented in the Associated Accounts list.



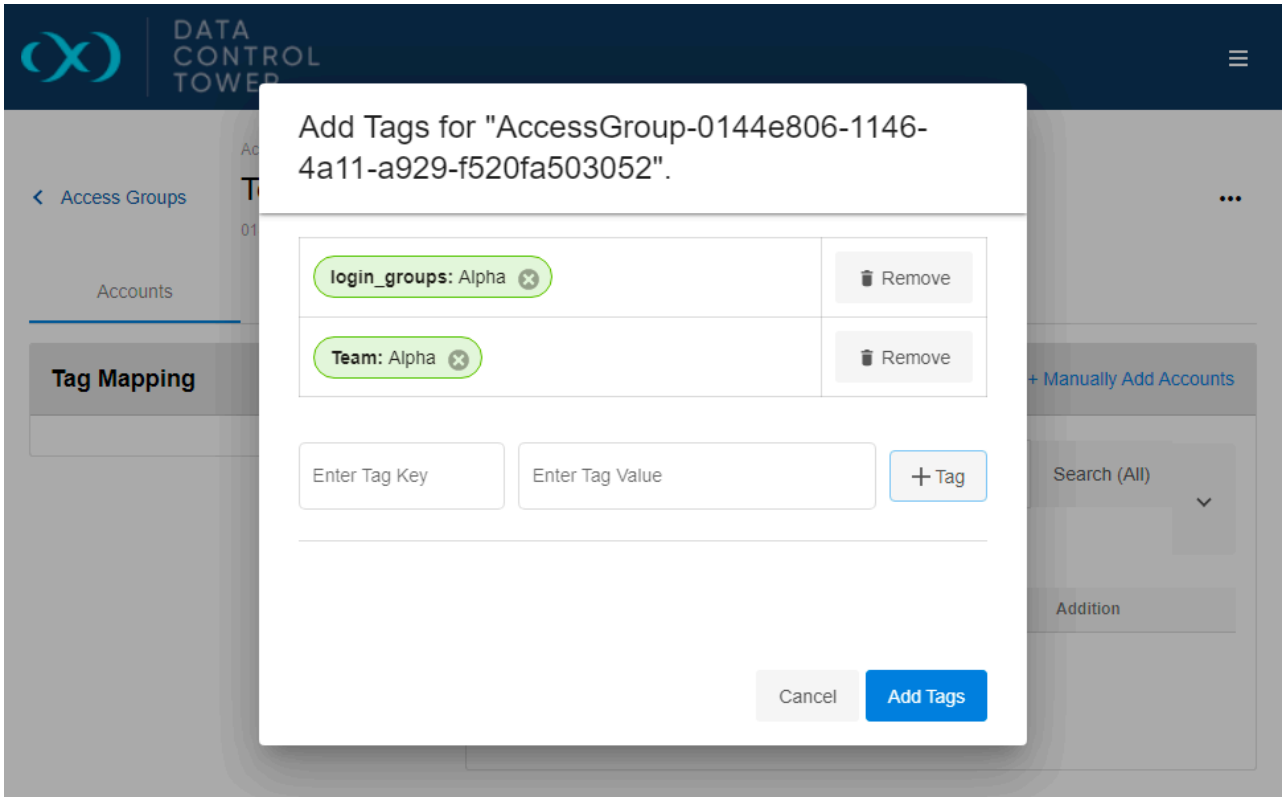
This is a good solution for quick management. However, it can be cumbersome as usage grows. Therefore, we recommend tags!

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.

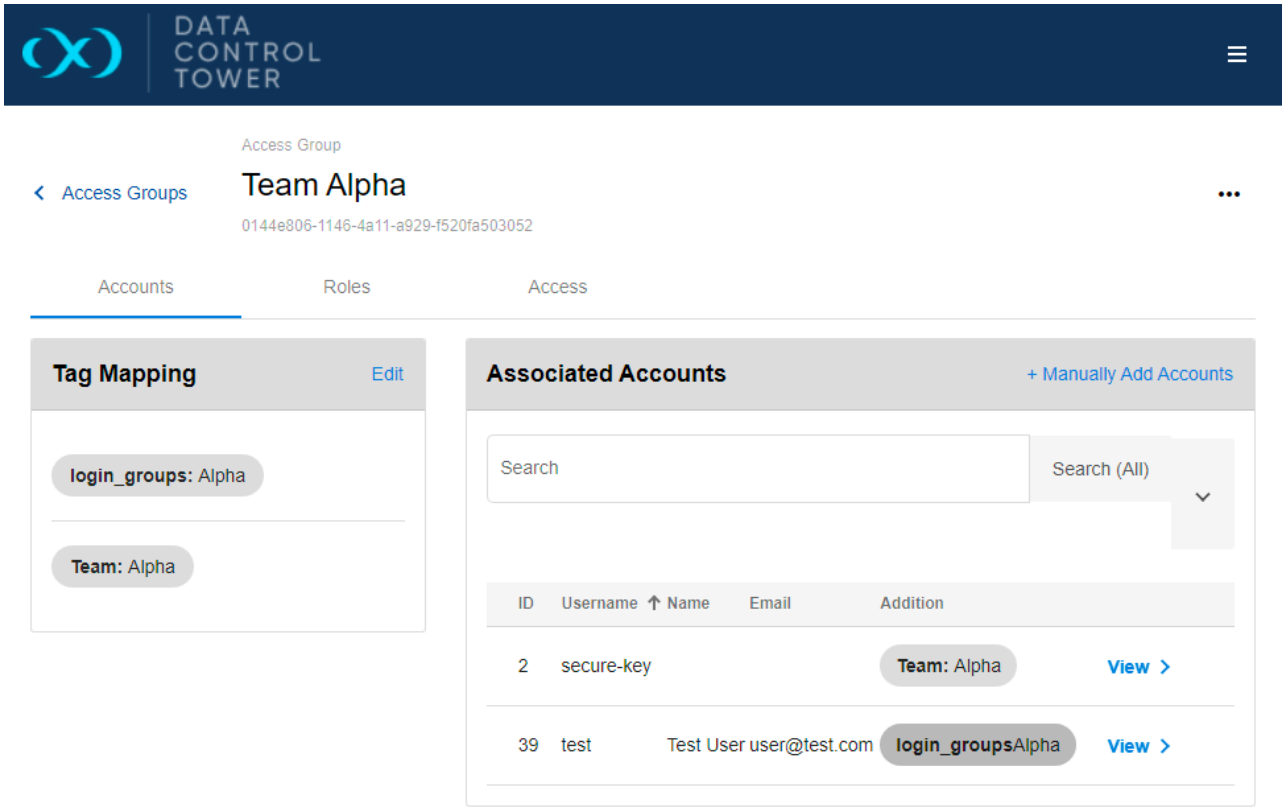


Next, navigate back to the Access Group, select the Tag Mapping's "Edit" button, and specify that same Key: Value pair. It might look similar to the below picture.



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.

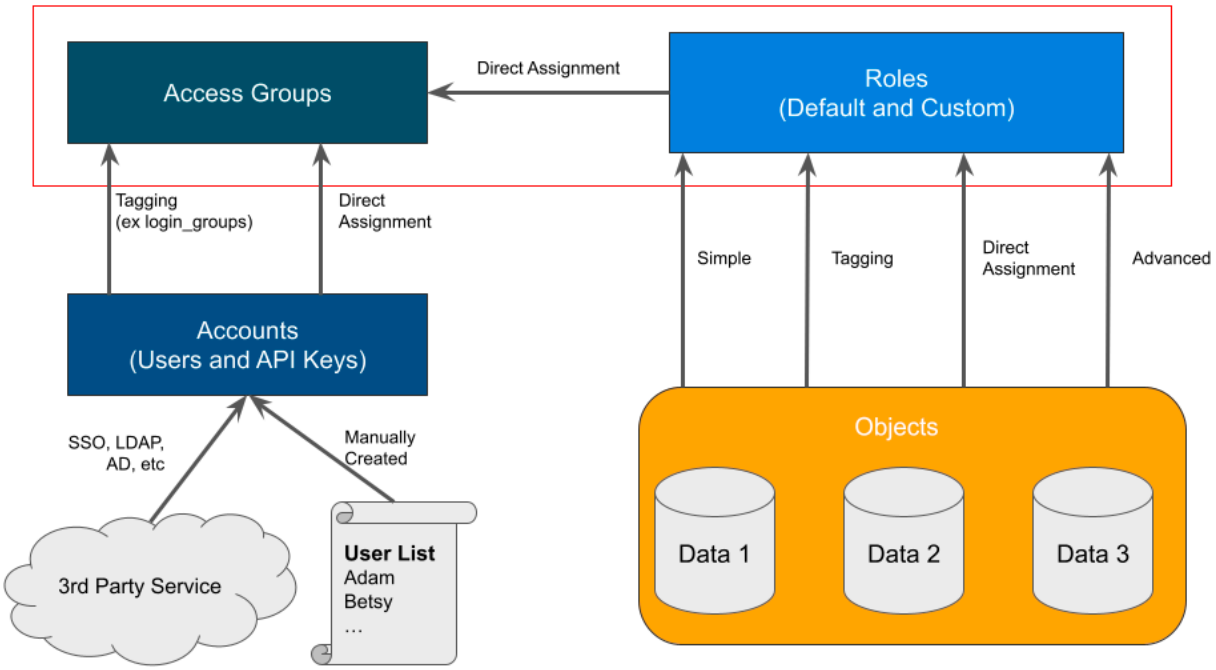
> The “login_groups” tag functions identically to a custom tag within the Access Group. Again, the only difference is that it's automatically assigned to the Account.



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[41].

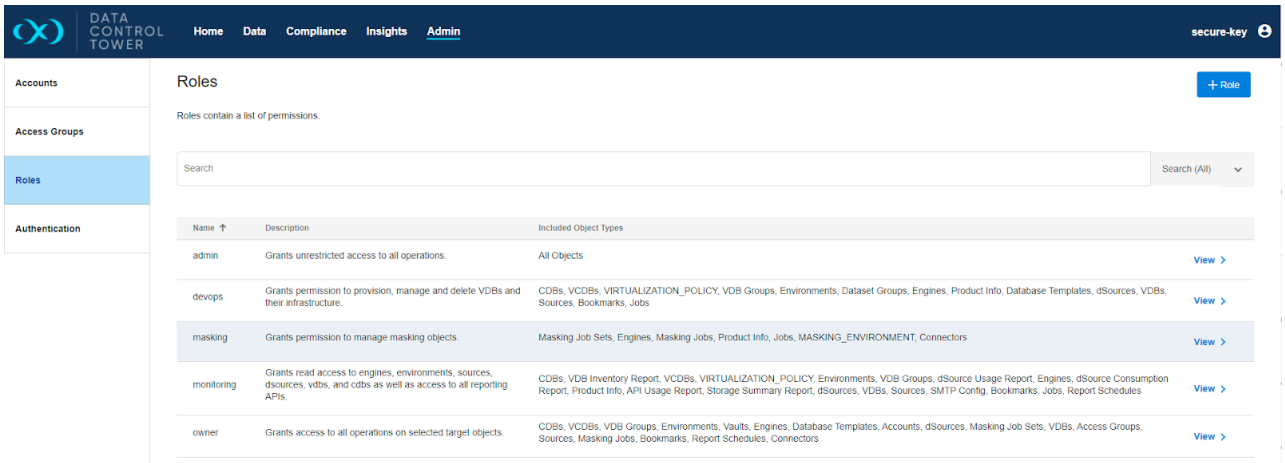
41 https://dct.delphix.com/docs/access-groups-2

5.1.3.4 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.

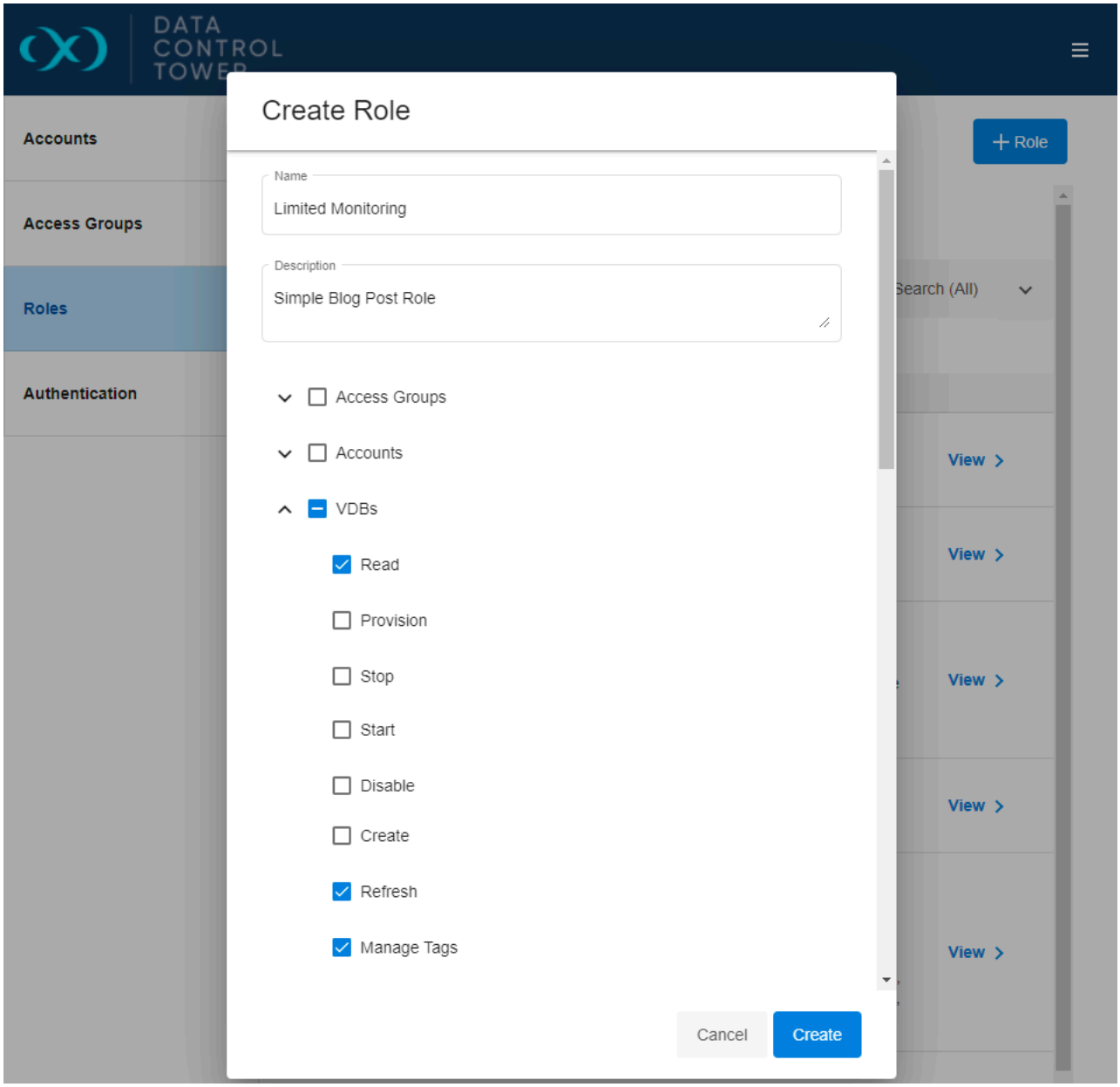


On the left-hand side, you can see a description, the Access Groups it's currently a part of, and any assigned Tags. On the right-hand side, is the complete list of permissions. For example, you can see here that the "devops" role has "Manage Tags" and "Read" permissions on the CDBs objects. These various permissions make up the Role's identity.

> 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.

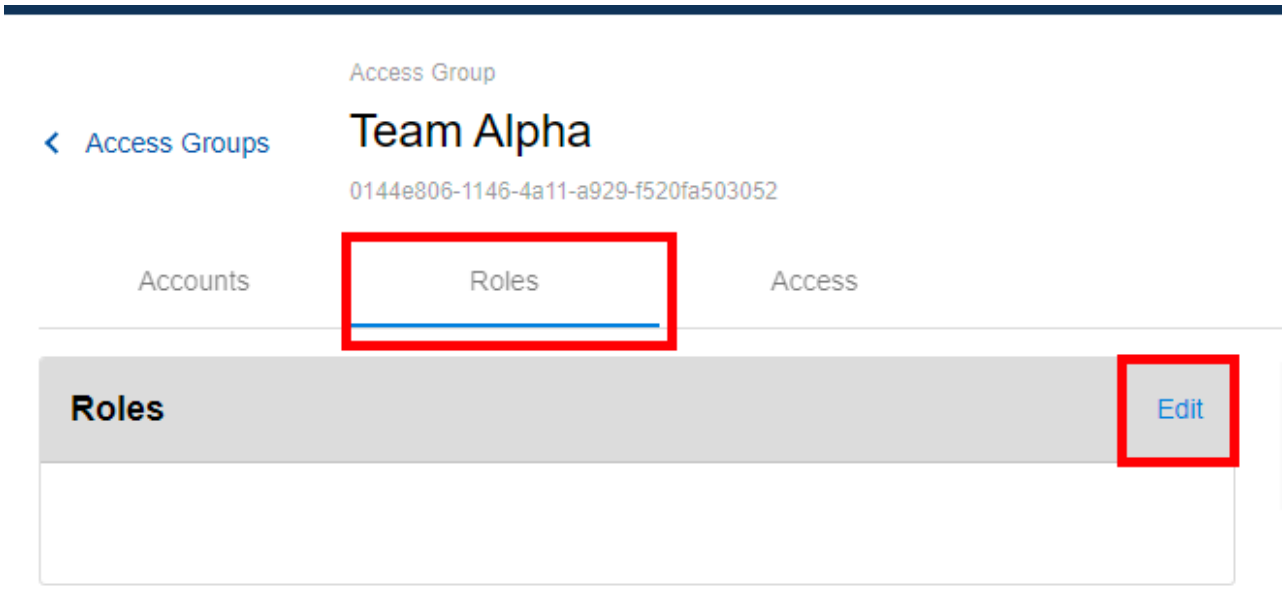


Once happy with your selection, click "Create". You can modify your Permissions further on the presented page.

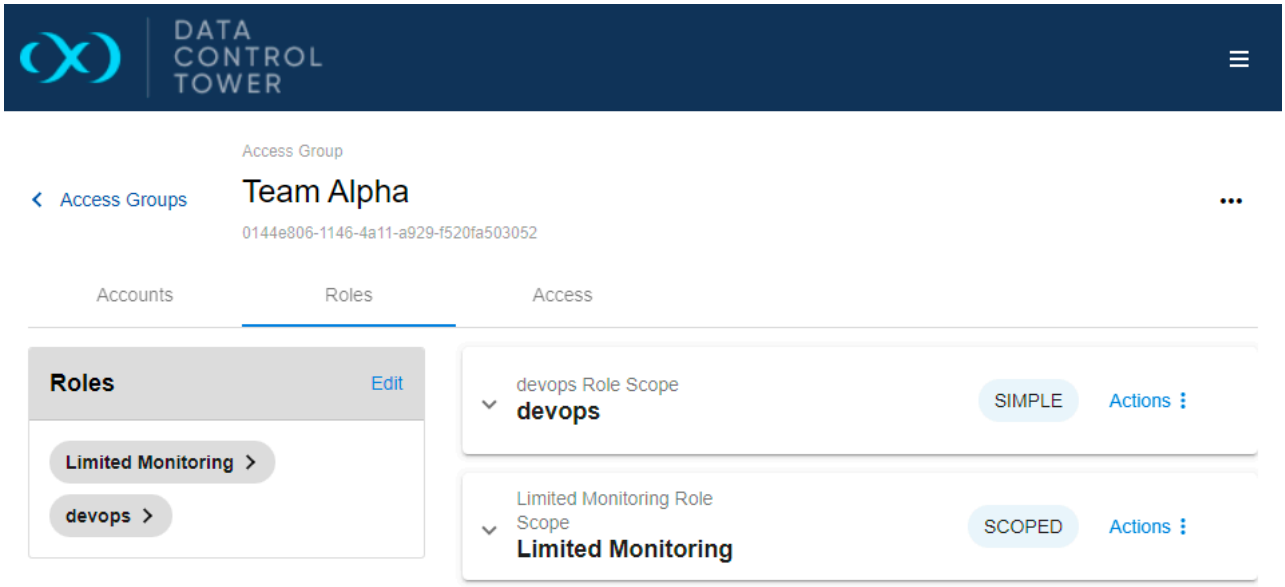
Role Assignment

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.

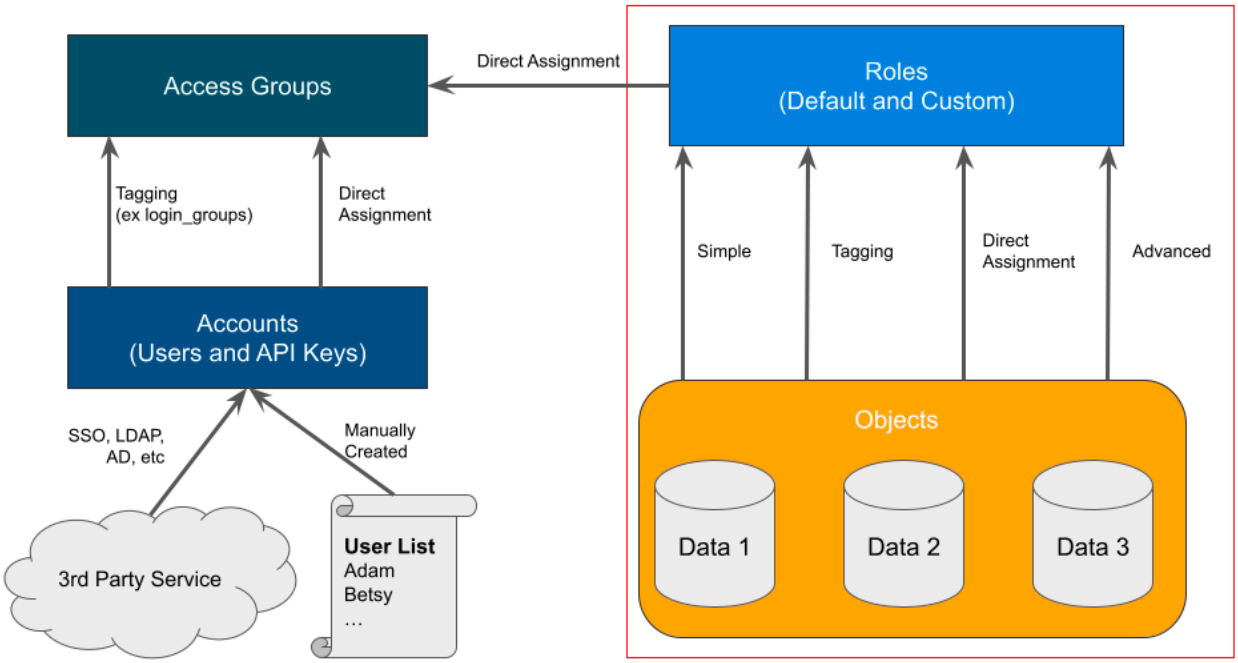


Now you can assign default Roles, such as “devops”, and your newly created Role. You might recall that Role assignment was also possible during Access Group creation. On Save, your Access Group might look like the following.



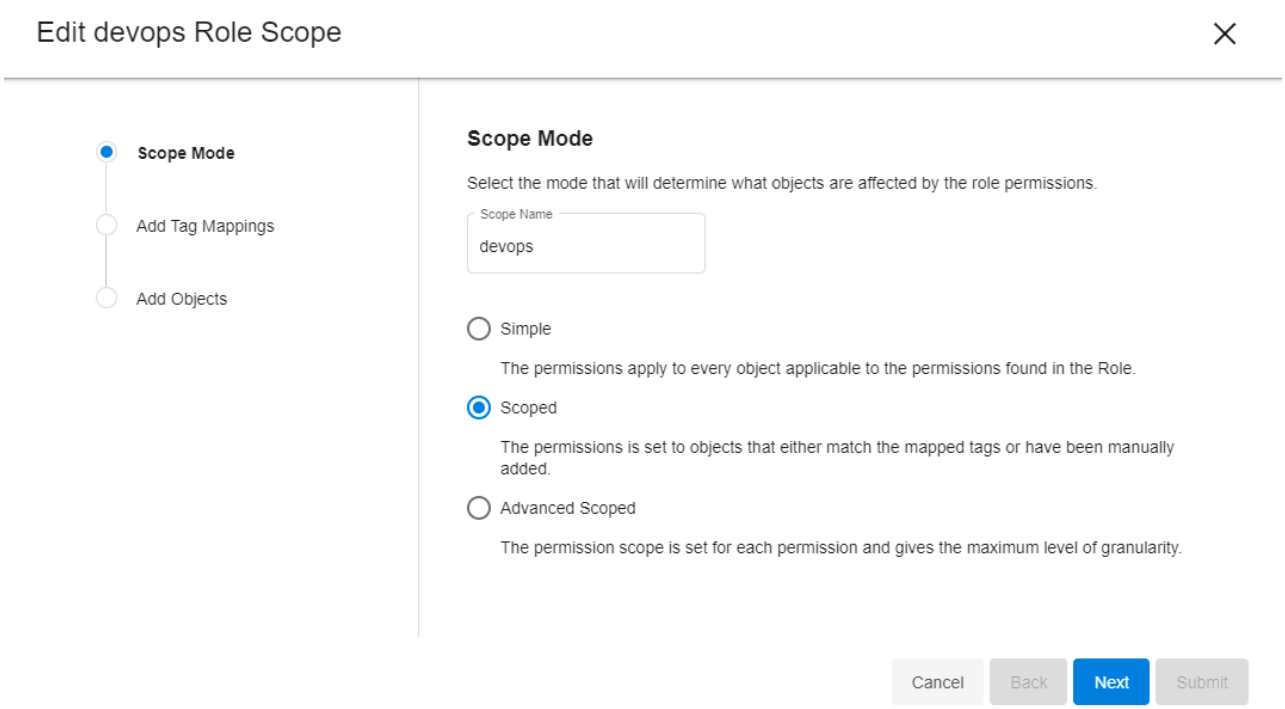
Immediately on assignment, all users within the Access Group will now have the permissions assigned to them through these roles. (Since you are currently an Admin user, you must log in as your test Account user.) However, you might notice that this user has full access to every object on DCT. The following section will define the Role scoping modes and refine the Account Object access.

5.1.3.5 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:

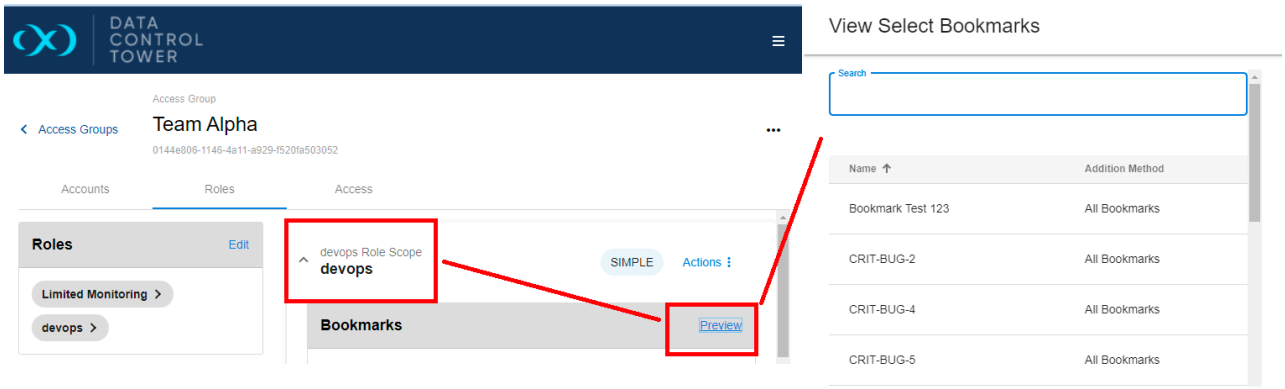
1. Simple - All Objects within DCT.
2. Scoped
    a. Tags - Objects with matching Tags.
    b. Direct - Objects manually assigned.
3. Advanced Scoped - Objects are assigned directly on the permission action (such as Read Bookmark, Edit Bookmark, or Delete Bookmark) using Tags or Direct Assignment.



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.

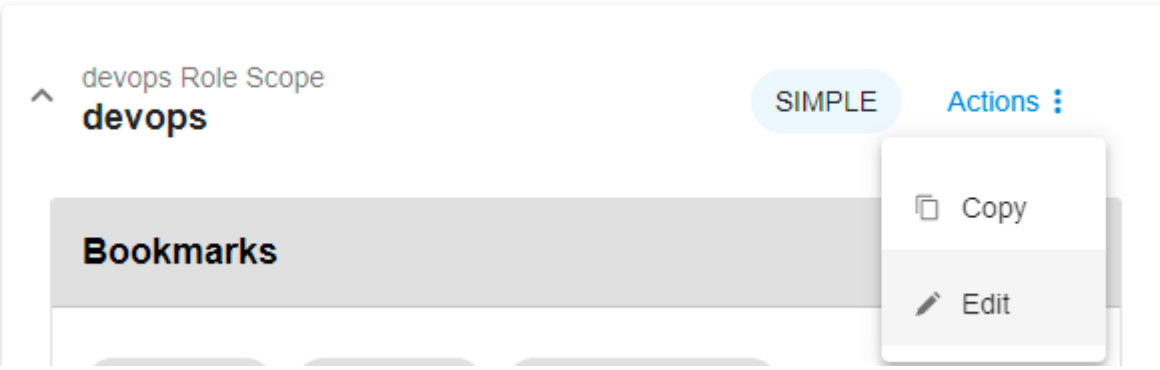


In this example, we select the “Preview” list for Bookmarks. It displays every Bookmark this role has access to and the method to which they are applied. If we wanted to validate, we could log in as a user on this Access Group and verify the permissions are applied. However, this is an easier way for 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.

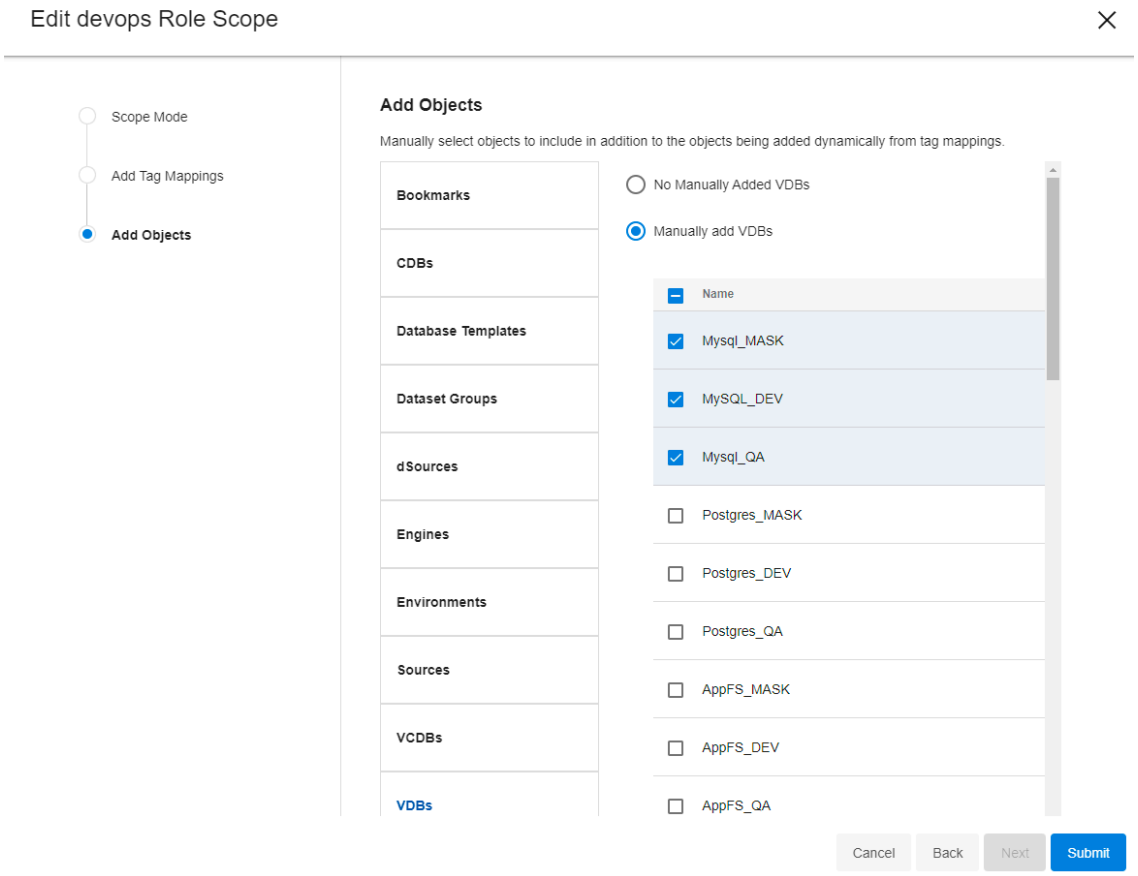
If you do not see any objects listed in the “Preview” widget, that object might not be available to DCT. This could be because (a) engines are not connected, (b) the DCT-only object (such as Bookmarks or VDB Groups) is not created, or (c) permissions are being enforced correctly.

Scoped - Direct

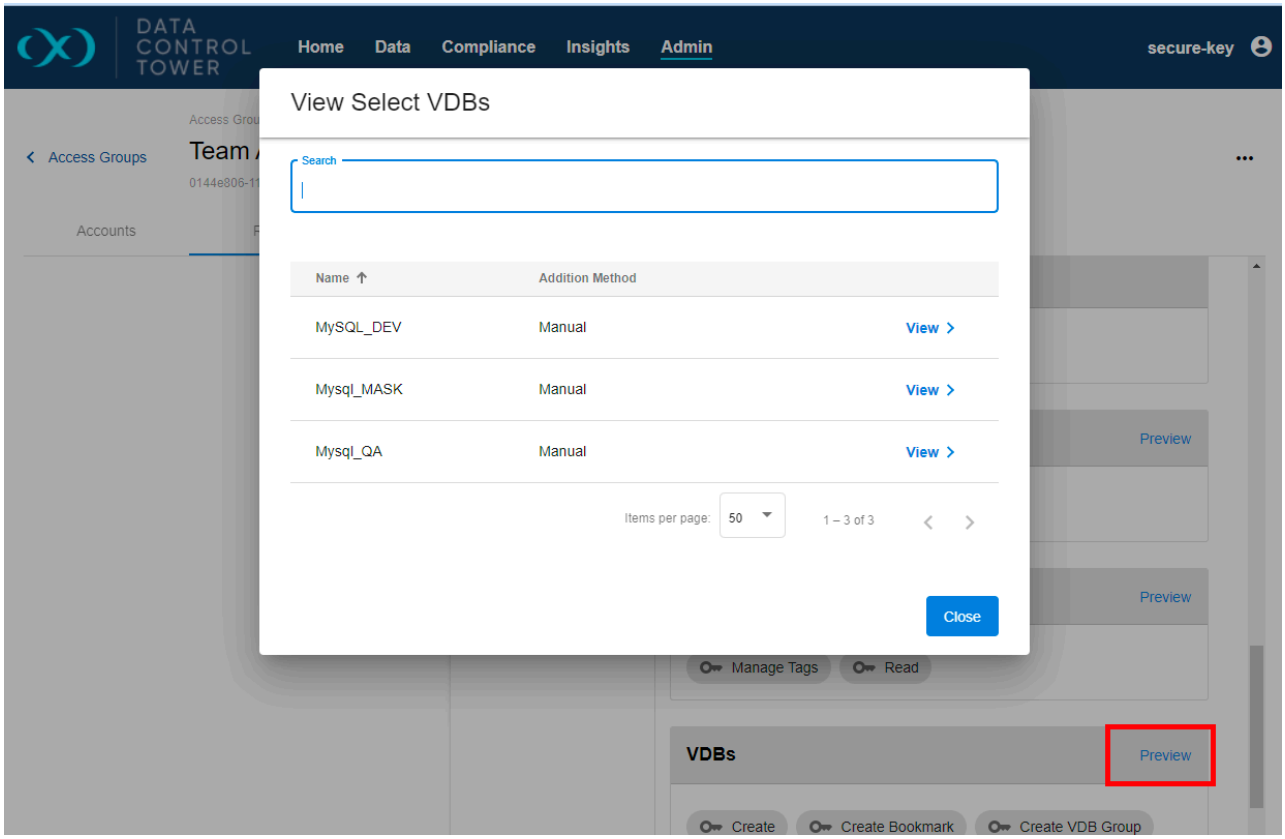
Let’s change the mode to “Scoped” and target a subset of VDBs. On the Access Group > Roles tab, select the **Action** > **Edit** button of your chosen Role.



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.



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.



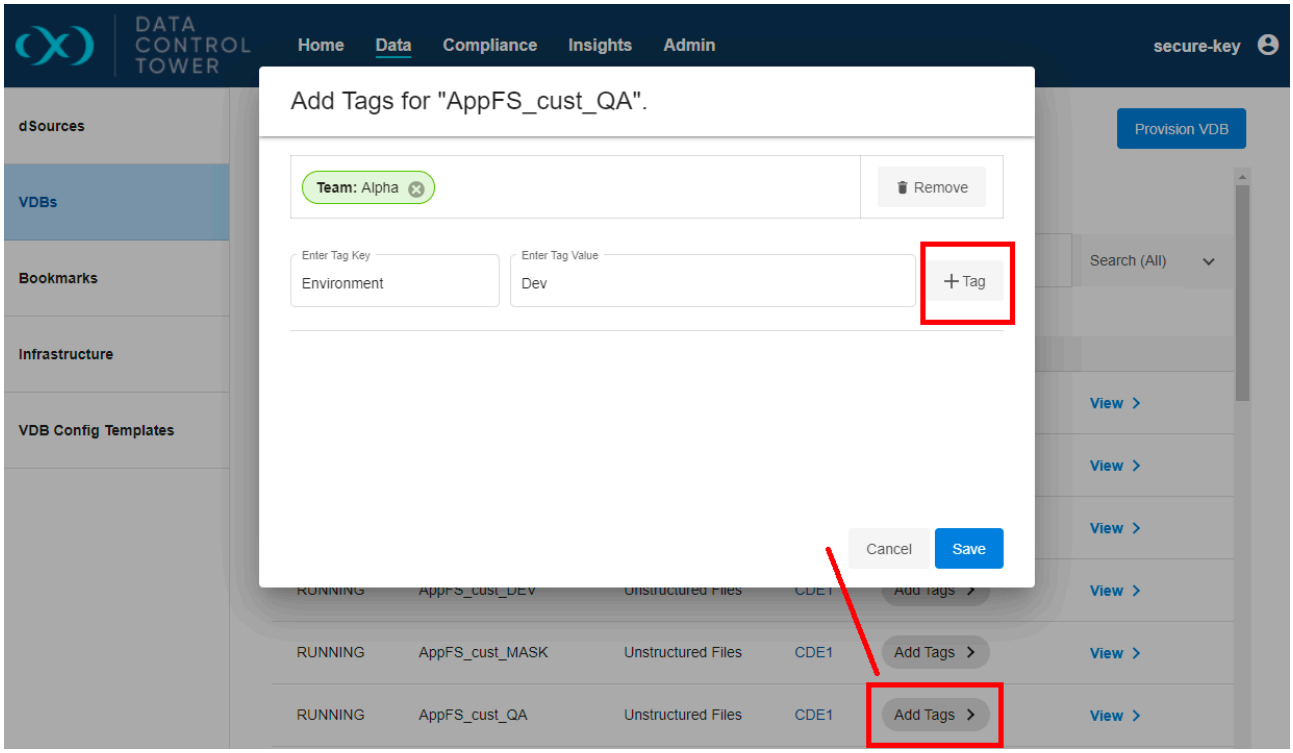
In my example, the same three VDBs I selected during permission configuration are shown here. If you want to verify manually, log in as your other test user and confirm.

> ⓘ 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

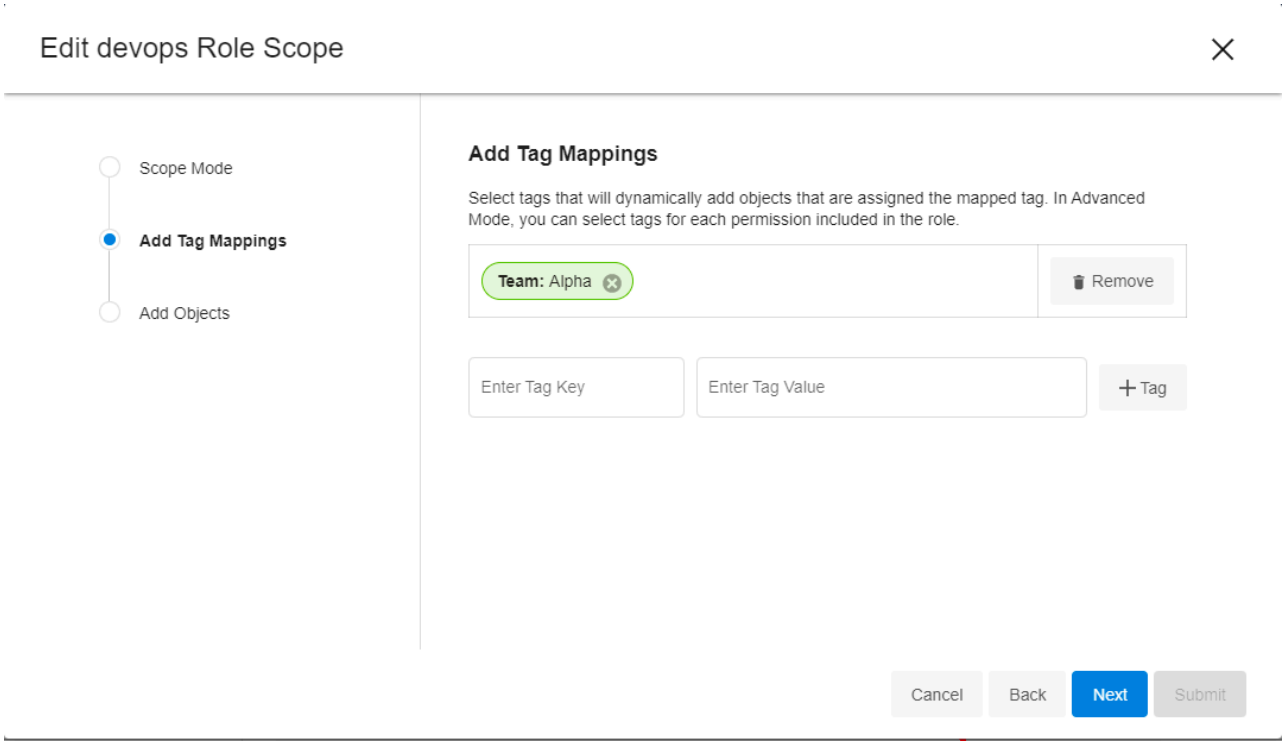
Direct Assignment is a solid strategy for early onboarding and one-off requirements. However, as we expand our consumption of Delphix, I suggest leveraging the Tagging mechanism to assign permissions quickly. Similar to the Account & Access Group’s “login_groups” tag, we can assign tags to Objects and 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.

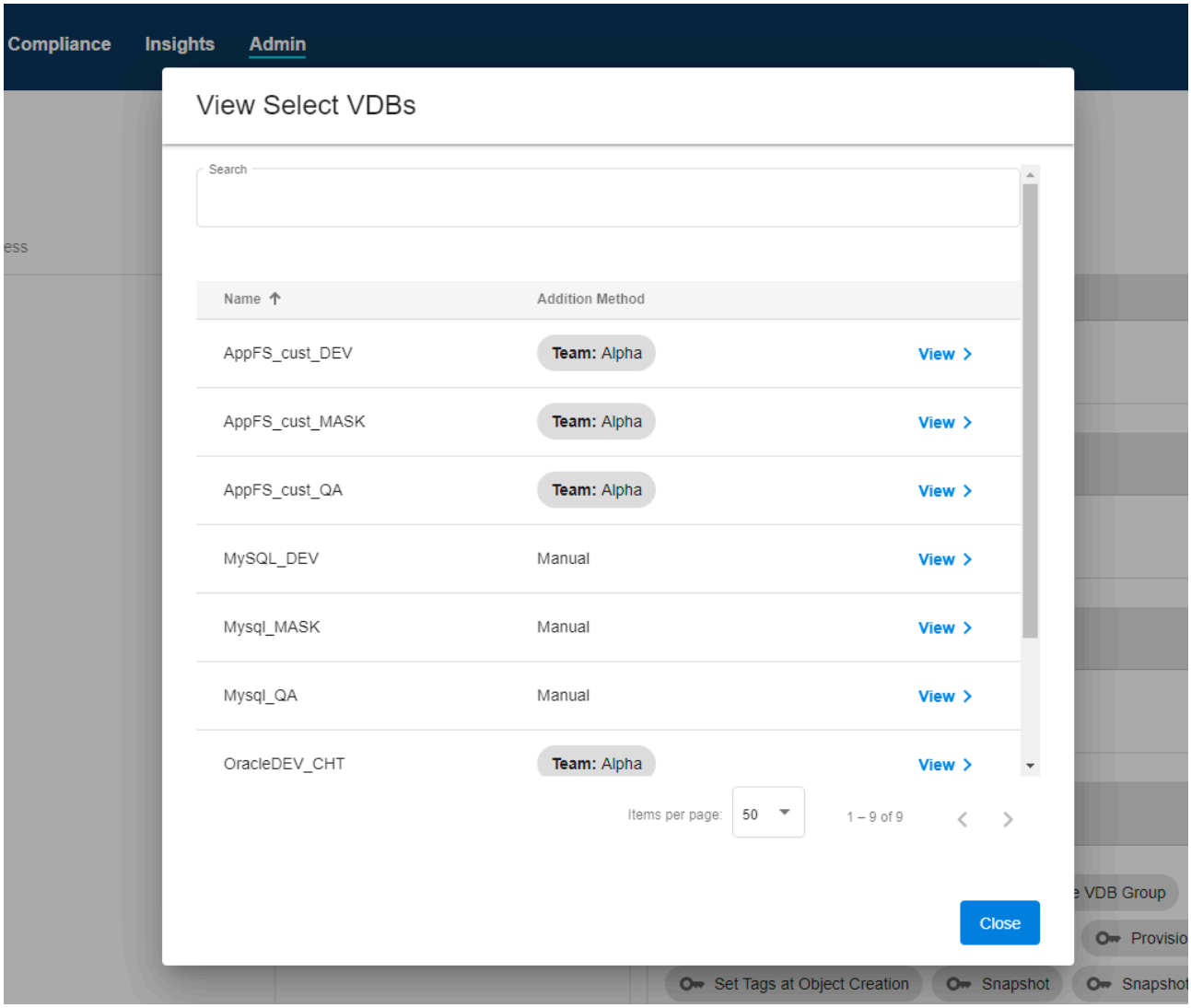


In this form, we assign a simple Key-Value pair. This pair helps govern access and maintain the organization of the Delphix Platform. I've selected the "Team: Alpha" and "Environment-Dev" pairs in my example. Repeat the process for a couple of other objects using similar or different Key-Value pairs. 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.



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.



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.[42]

5.1.4 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.

42 https://dct.delphix.com/docs/dct-tags

5.1.4.1 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.

5.1.4.2 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"
}'
```

5.1.4.3 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.

5.1.5 API metering

5.1.5.1 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”.

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
```

5.1.6 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.

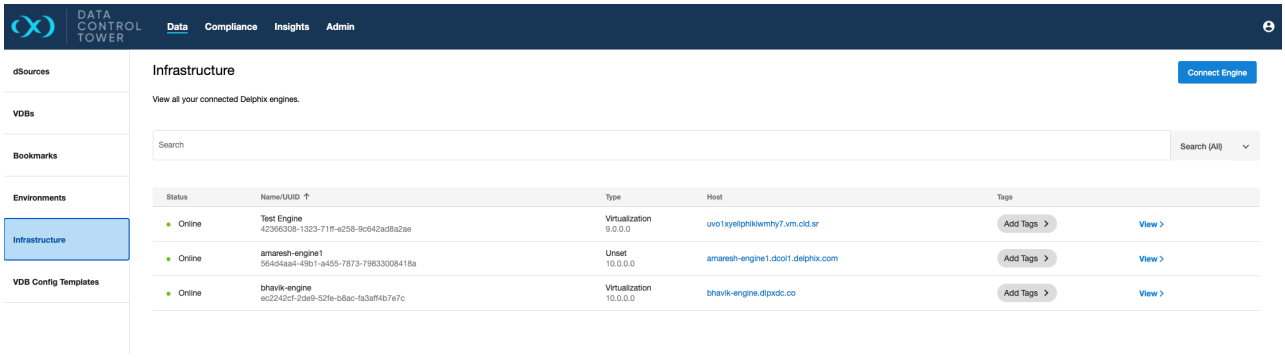
5.2 Central governance workflows

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.

5.2.1 Managing engines (Continuous Data)

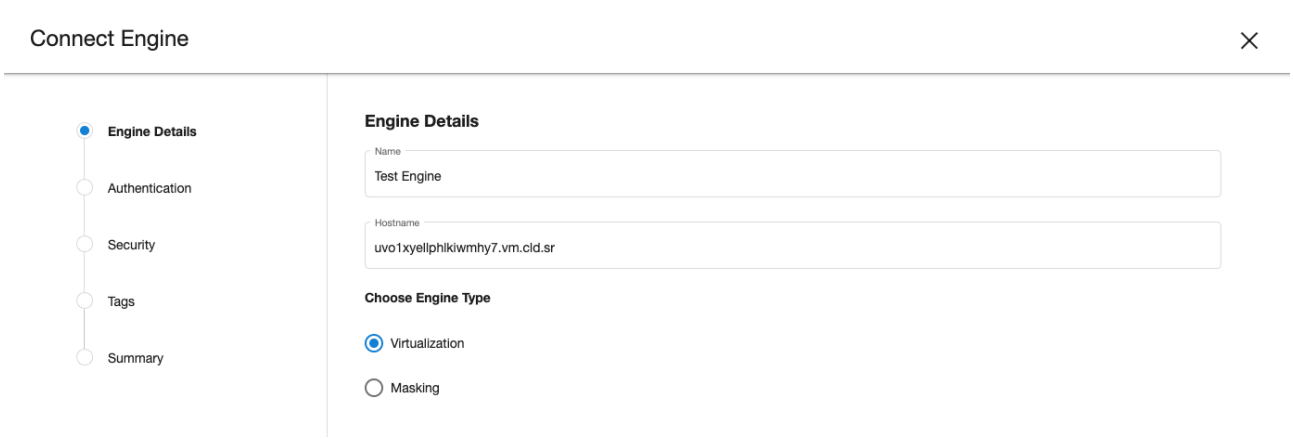
5.2.1.1 Infrastructure

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.



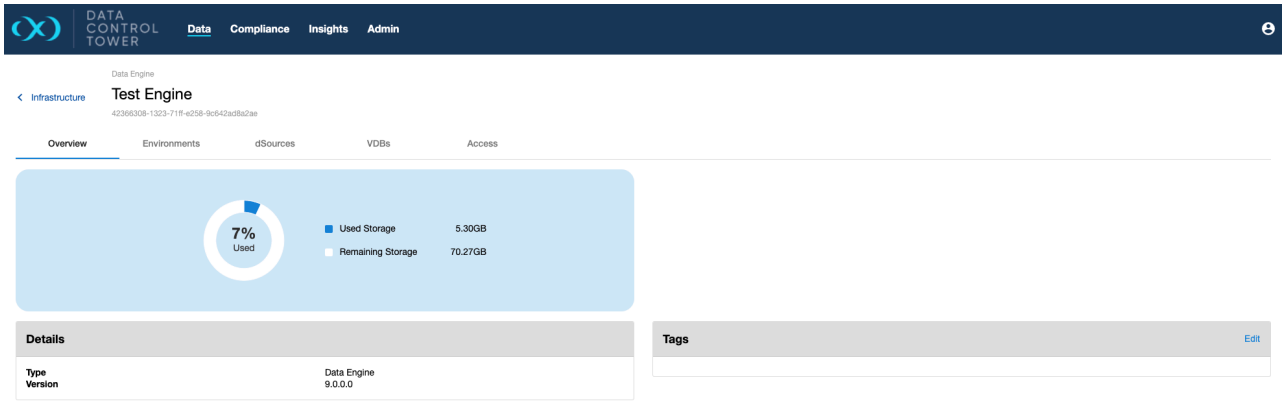
Administrators can manage engine connects via the “Connect Engine” button on the top right corner. By clicking this button, the below dialogue will appear, asking for connection details.

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.



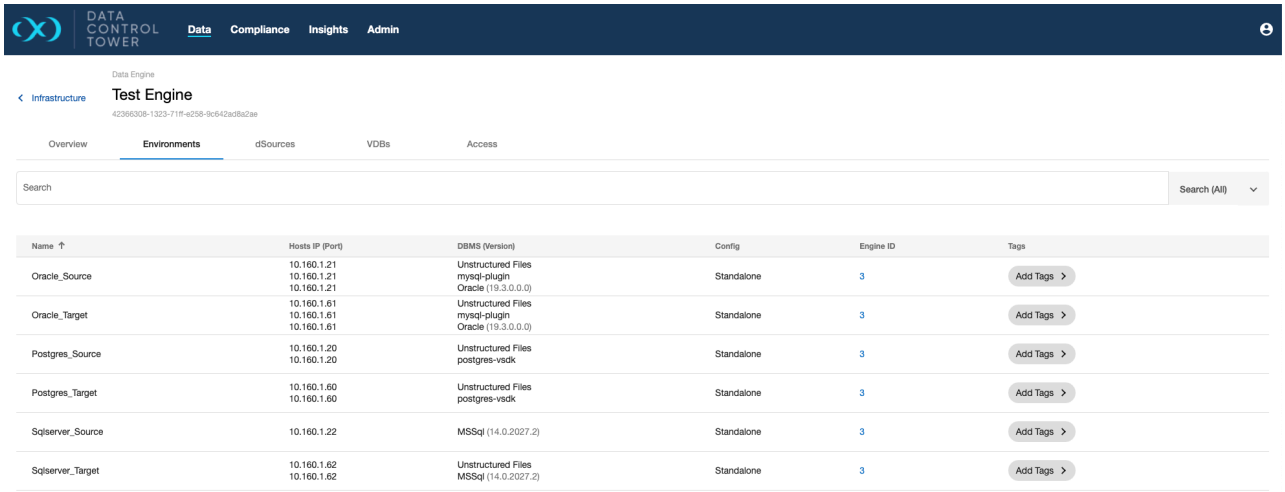
5.2.1.2 Engine overview

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.



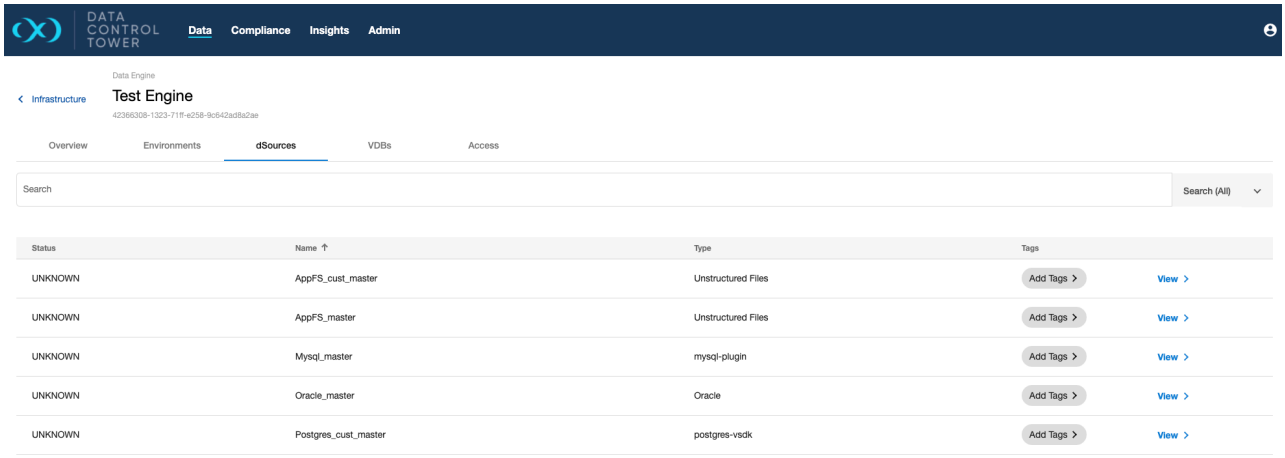
Engine-connected environments

The "Environments" tab presents all environment connections to that particular engine.



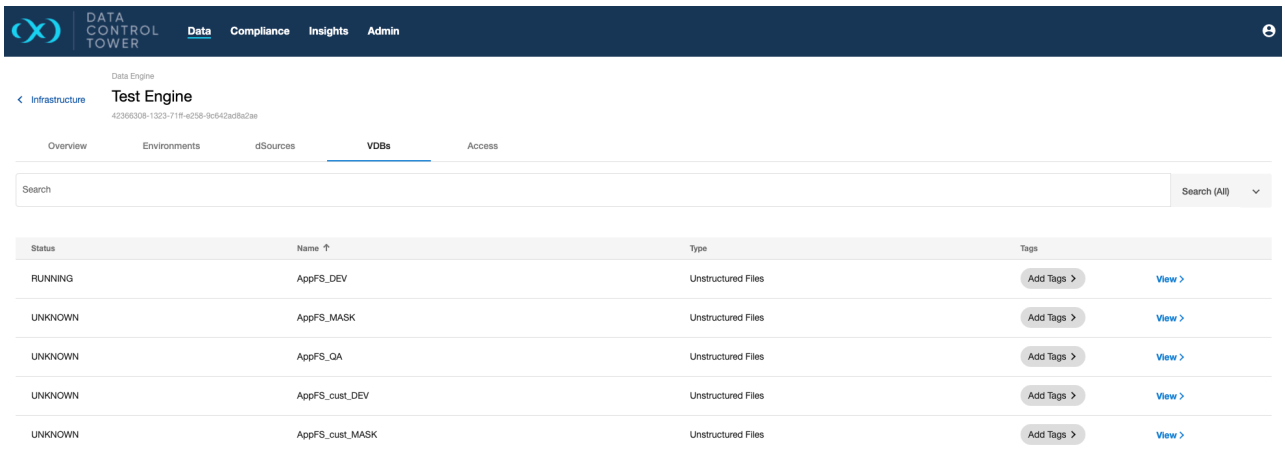
Local dSources

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.



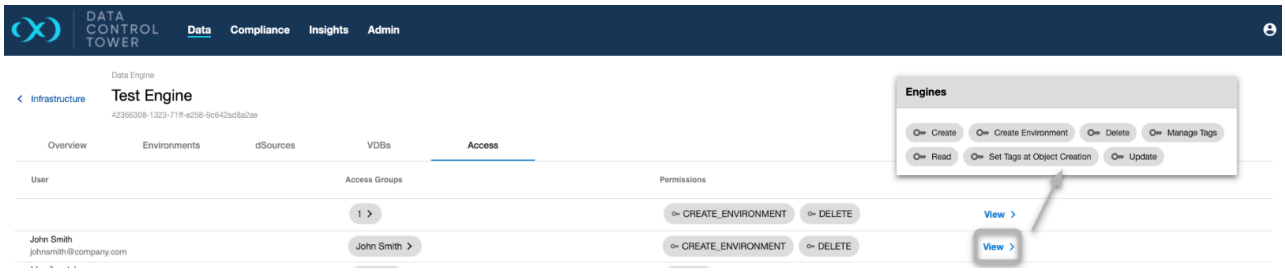
Local VDBs

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.



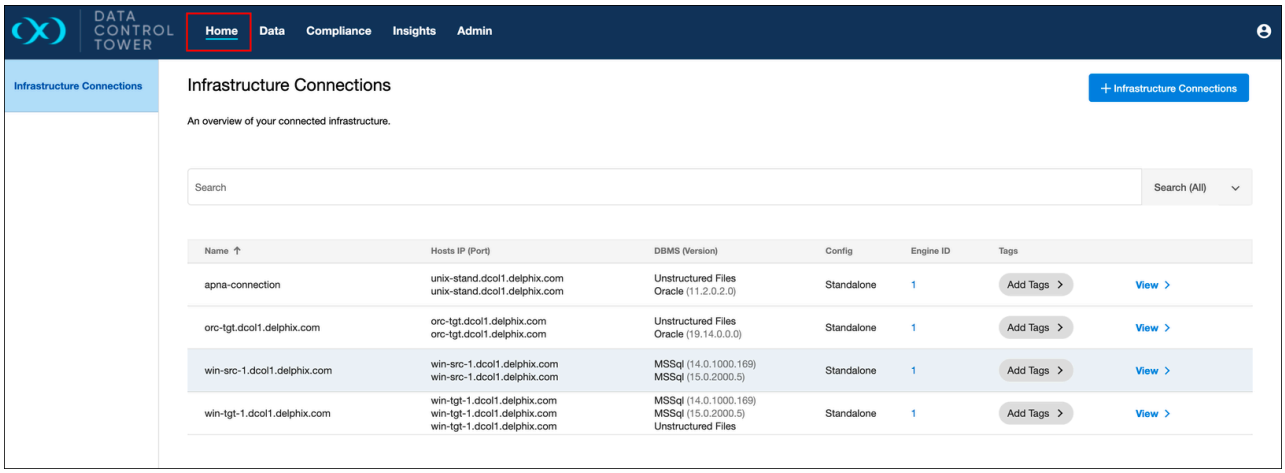
Engine-based Operations access

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.



5.2.1.3 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.

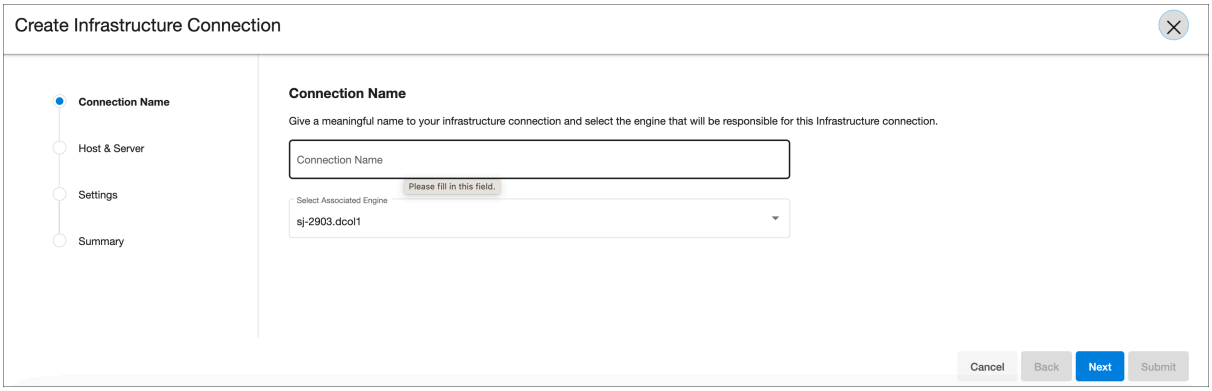


The following connections are available:

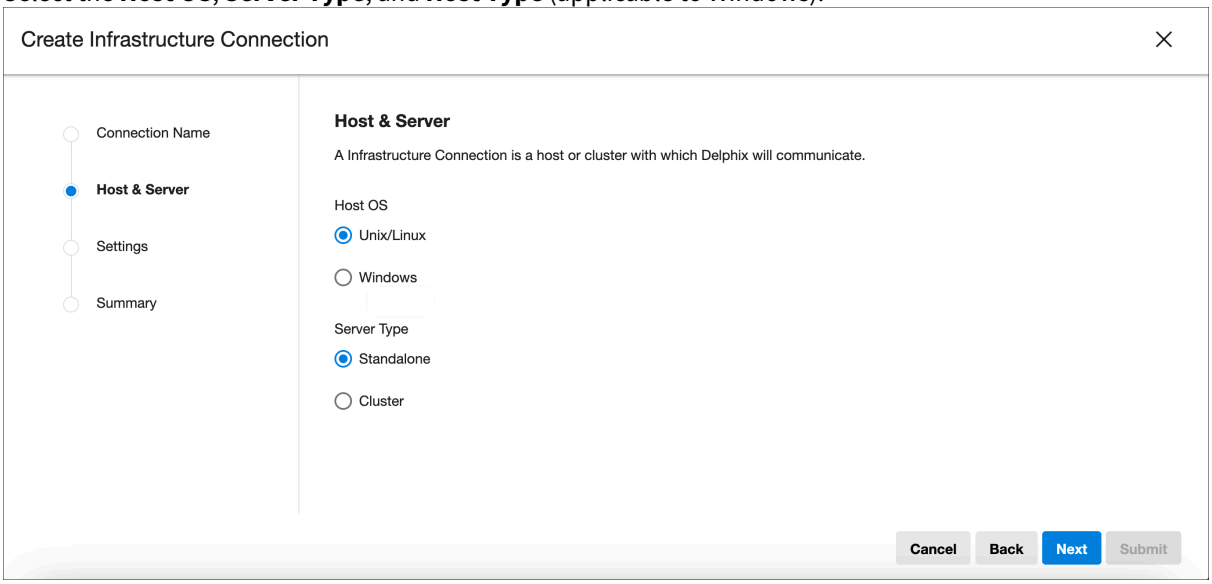
- UNIX environment
  - Standalone/Cluster
- Windows environment
  - Source/Target
  - Standalone/Cluster

Infrastructure connection wizard steps

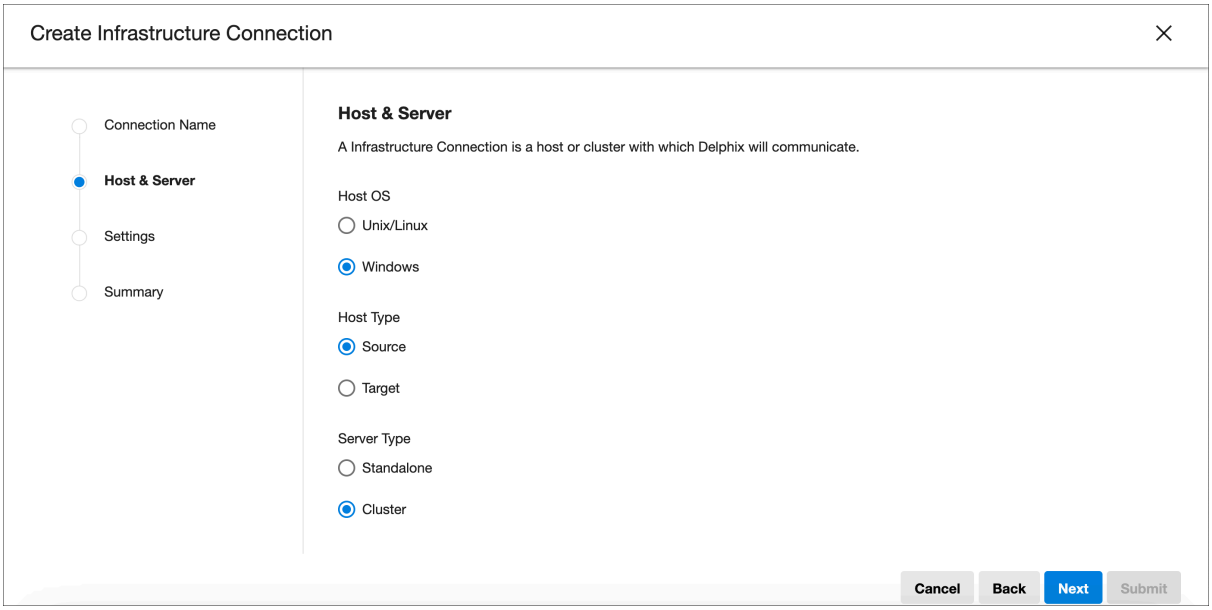
1. **Connection Name**
   Enter the **Connection Name** in the corresponding field and select the **Associated Engine** from the dropdown.



2. **Host & Server**
Select the **Host OS**, **Server Type**, and **Host Type** (applicable to Windows).



1 Unix

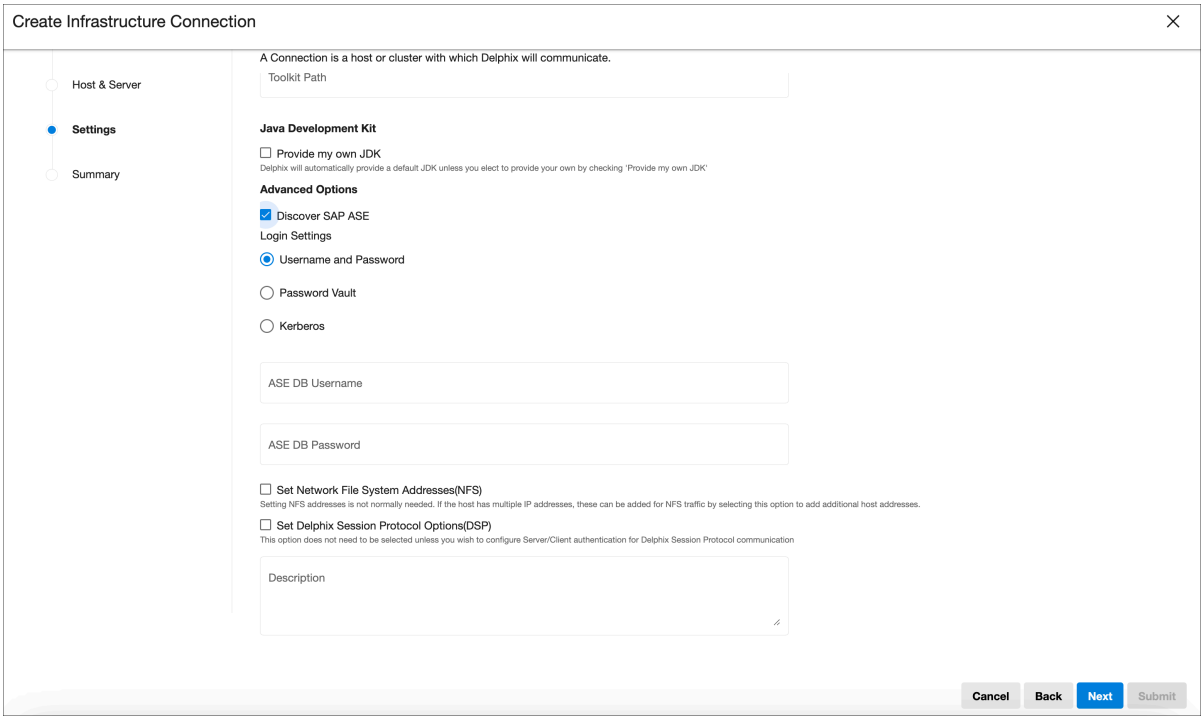


2 Windows

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.

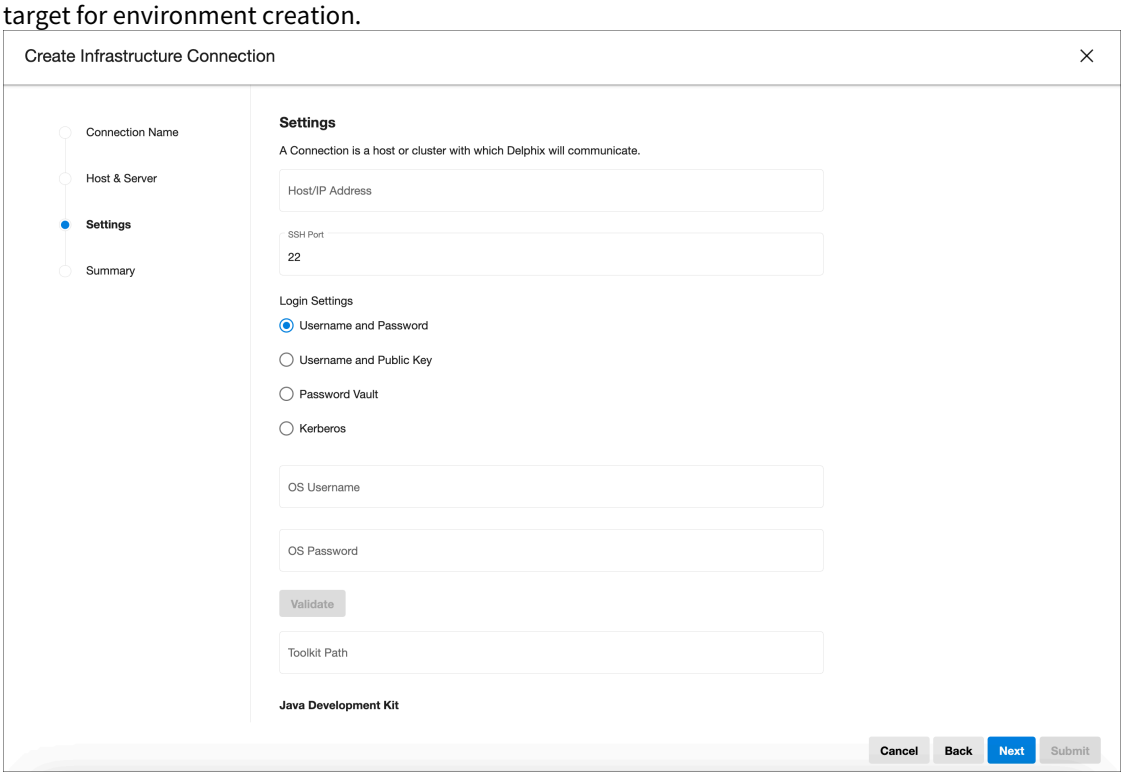
3. **Settings**

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.

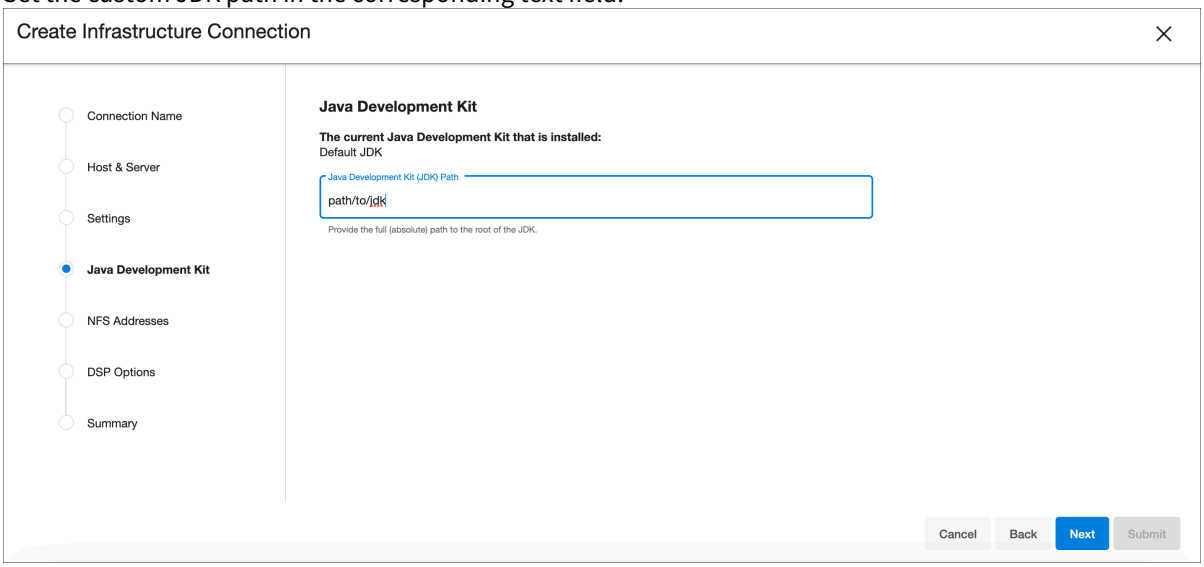


A wide range of **Login Settings** are available for the OS user and to **Discover SAP ASE** (if applicable), like username/password, username/public key, Password Vault, or Kerberos.

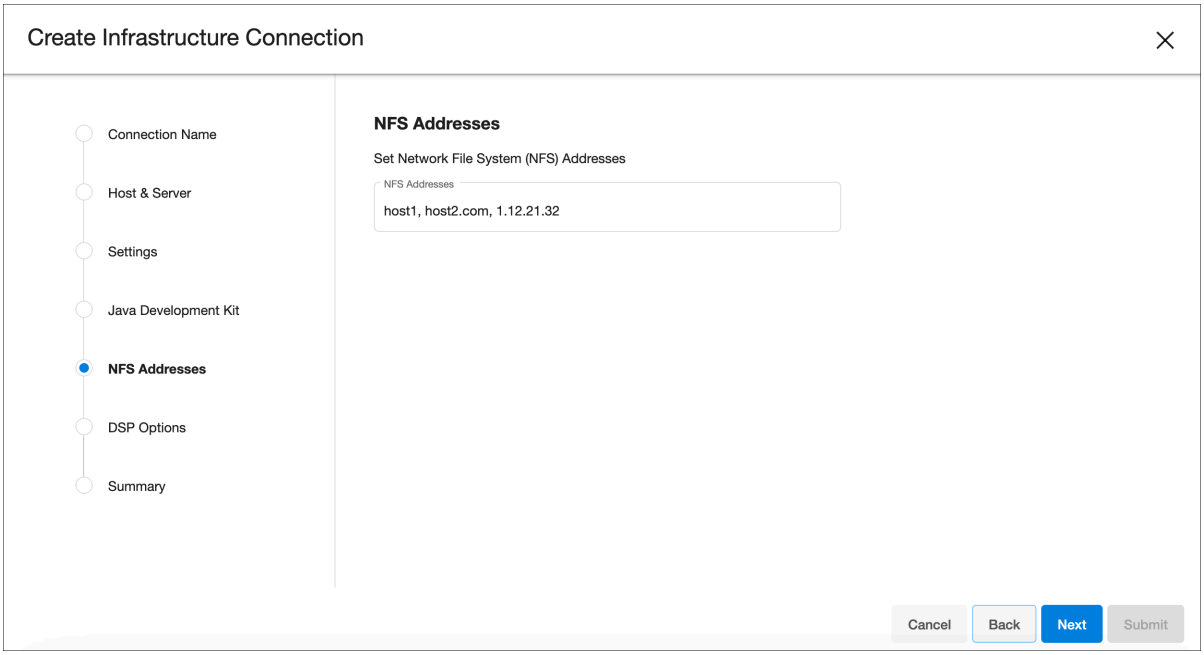
a. For vaults, HashiCorp and CyberArk vaults are supported.
b. The Kerberos login option only shows up when the user selects a Kerberos enabled engine as the target for environment creation.



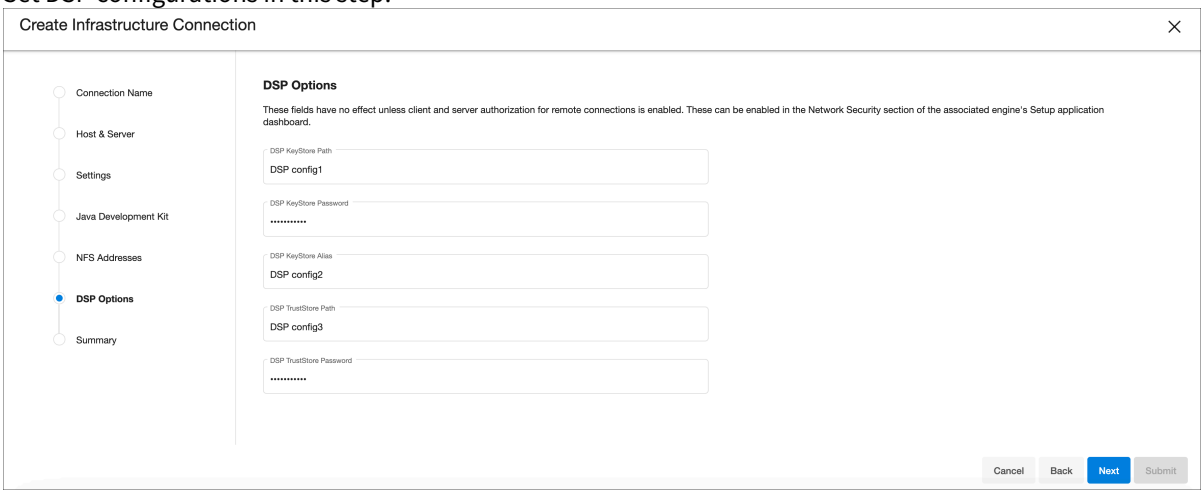
4. **Java Development Kit** (only applicable if selected in **Settings**)
Set the custom JDK path in the corresponding text field.



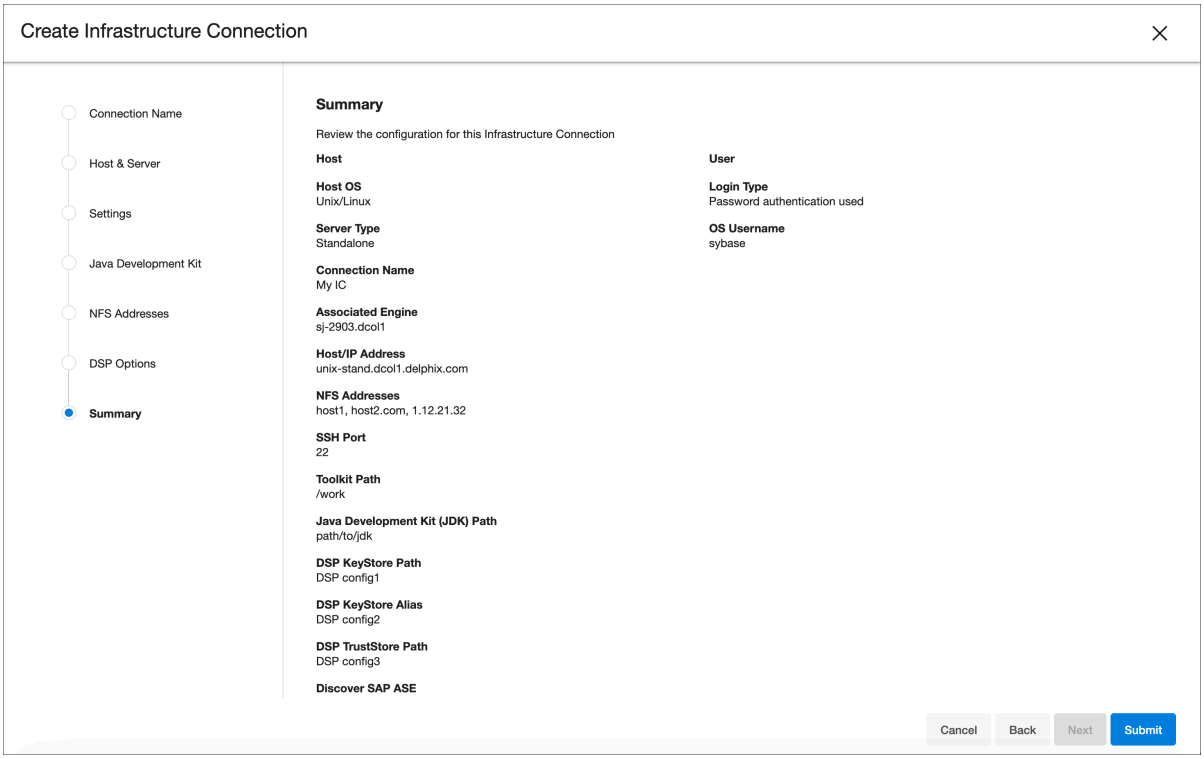
5. **NFS** (only applicable if selected in **Settings**)
Set NFS addresses in the corresponding text field (comma separated).



6. **DSP** (only applicable if selected in **Settings**)
   Set DSP configurations in this step.



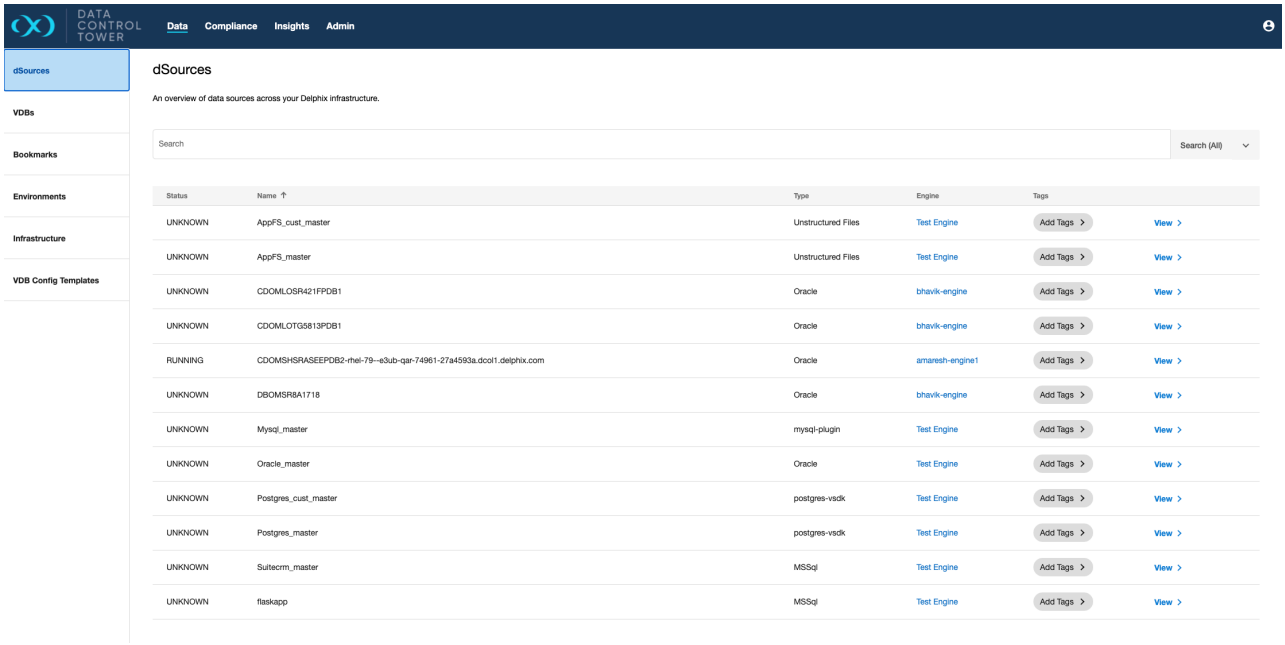
7. **Summary**
   Shows a comprehensive summary of the selected options in the previous configuration steps. Shows the type of login being used for both the OS user and SAP ASE (if applicable).



5.2.2 Managing dSources

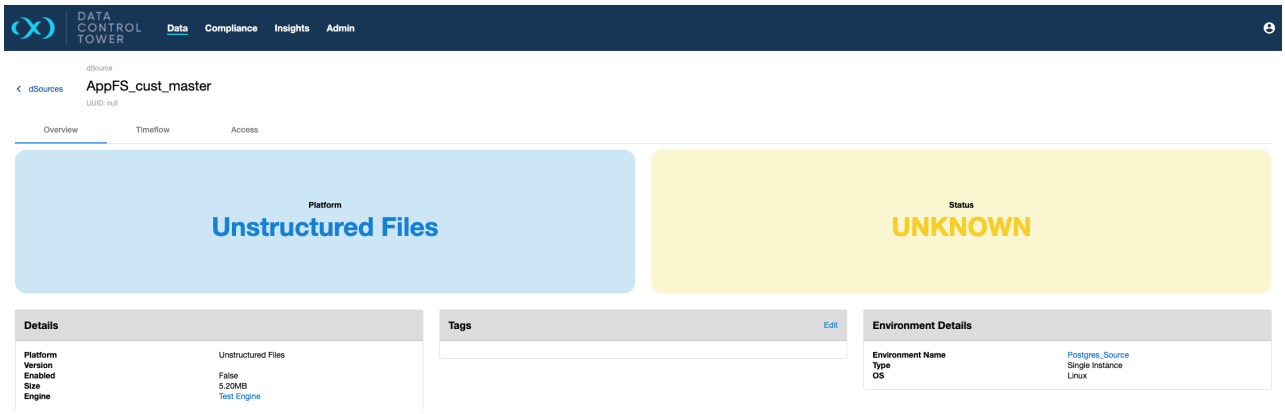
5.2.2.1 Managing dSources

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.



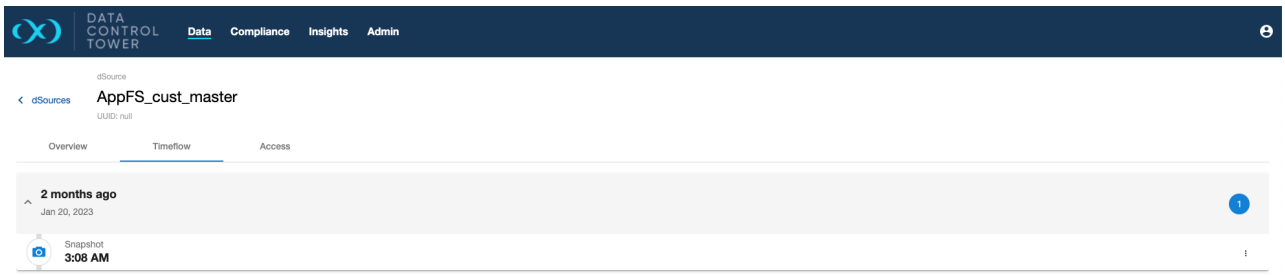
5.2.2.2 dSource overview

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.



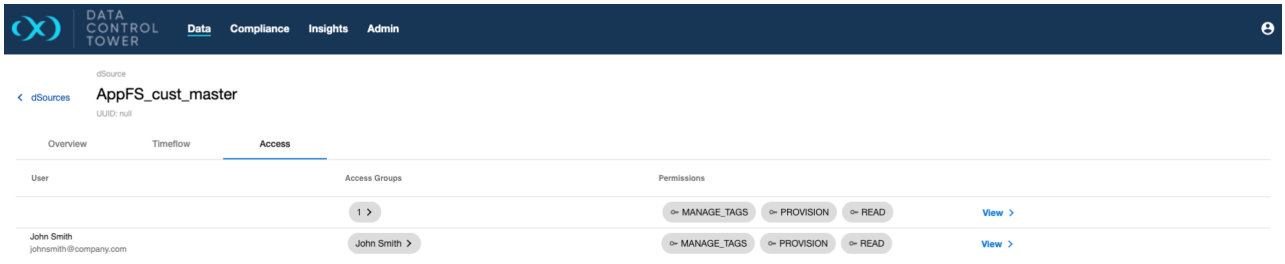
5.2.2.3 Timeflow visibility

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.



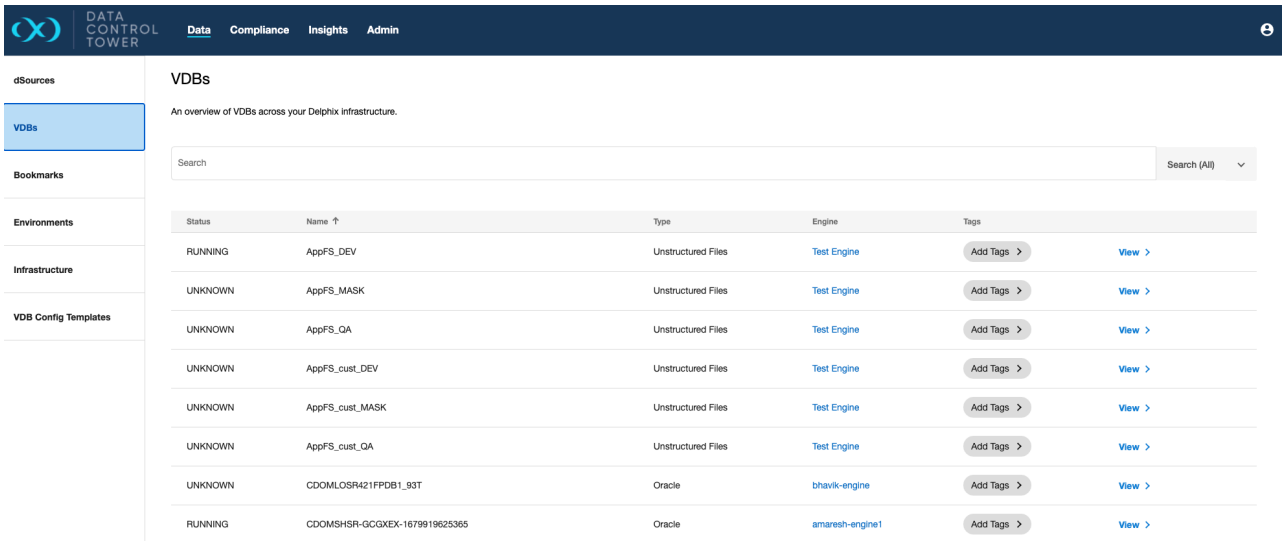
5.2.2.4 Access auditing

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.



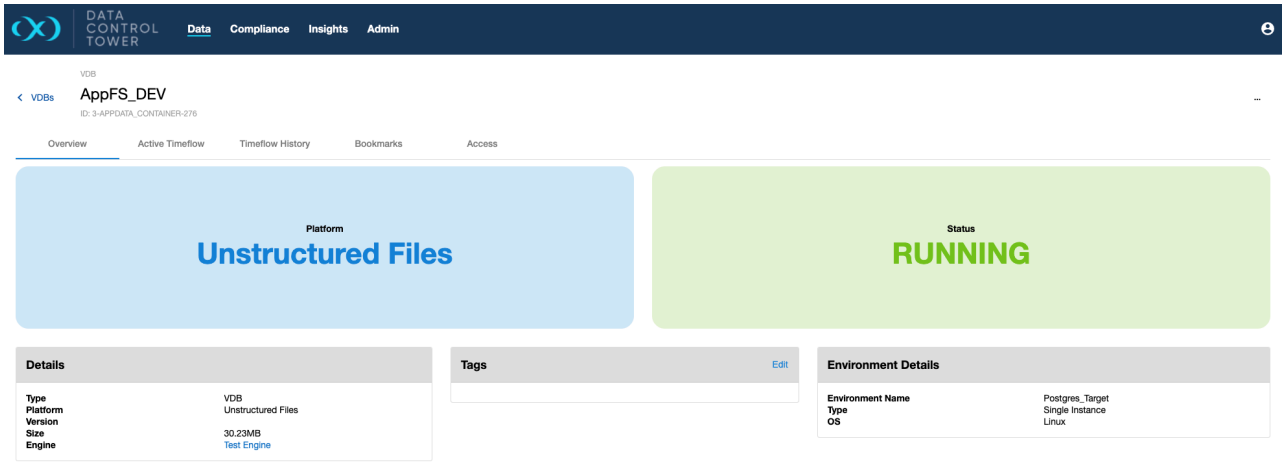
5.2.3 Managing VDBs

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.



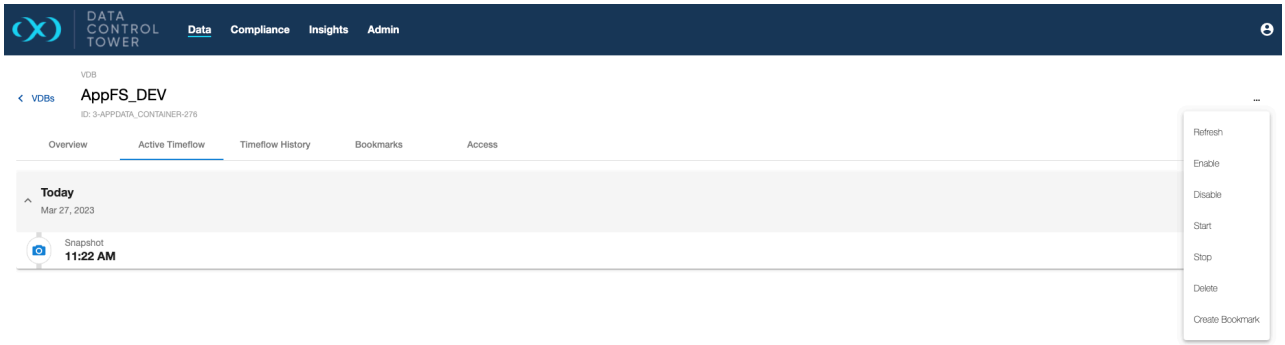
5.2.3.1 VDB overview

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.



5.2.3.2 VDB active timeline

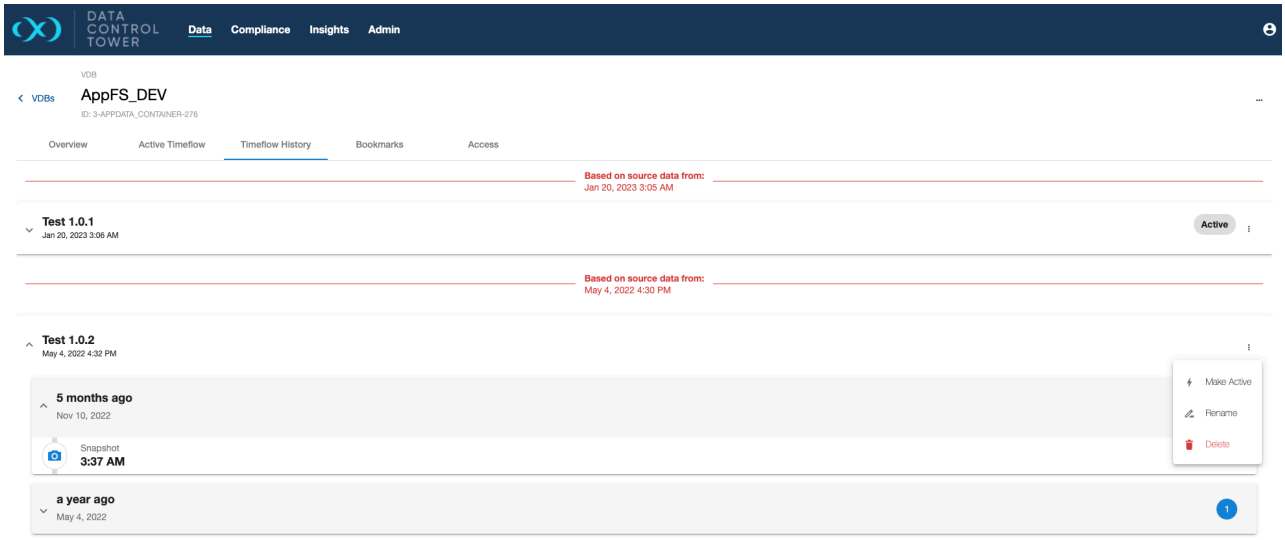
Actionable snapshots are listed on the "active timeflow" tab - from this page, users can refresh, enable, disable, start, stop, delete, and create bookmarks on the VDB. Navigate to the Continuous Data workflows section, then VDB operations in the UI, and see Active timeline UI for more details.



5.2.3.3 VDB timeline history

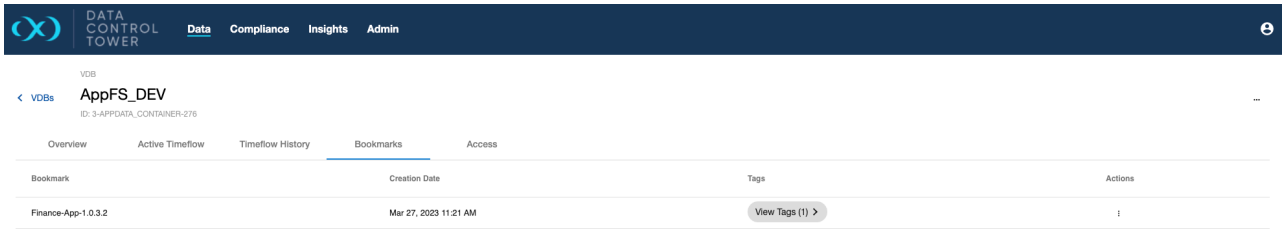
A chronological history of all non-active timelines (commonly referred to as timeflows) is shown under the "Timeflow History" tab. From this page, developers can curate their QA or Development work by renaming timeflows to match their testing history. Developers also have the ability to access old timeflow data by making a particular timeflow "active". Navigate to the **Continuous Data workflows** section, then **VDB operations in the UI**, and see Timeline history UI[43] for more details.

43 https://portal.document360.io/docs/timeline-history



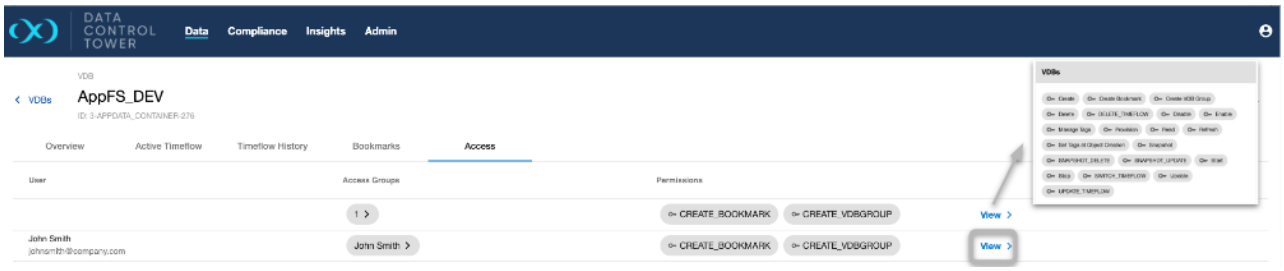
5.2.3.4 VDB bookmarks

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).



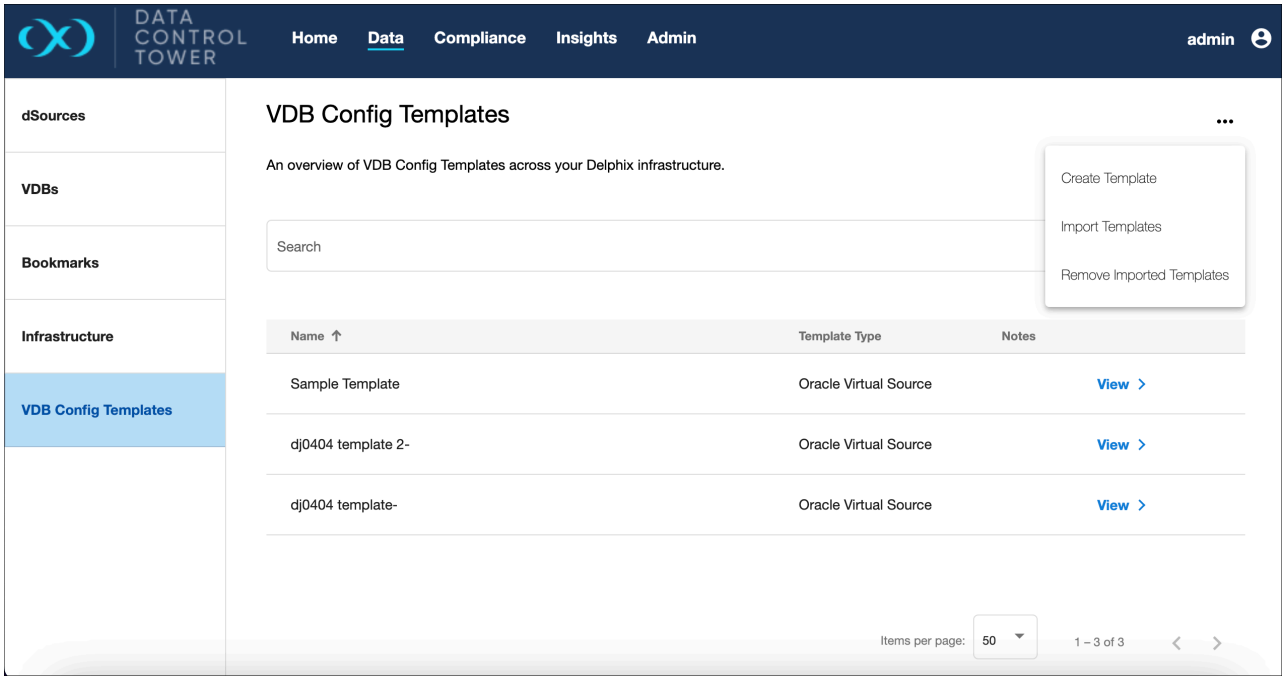
5.2.3.5 VDB access

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.



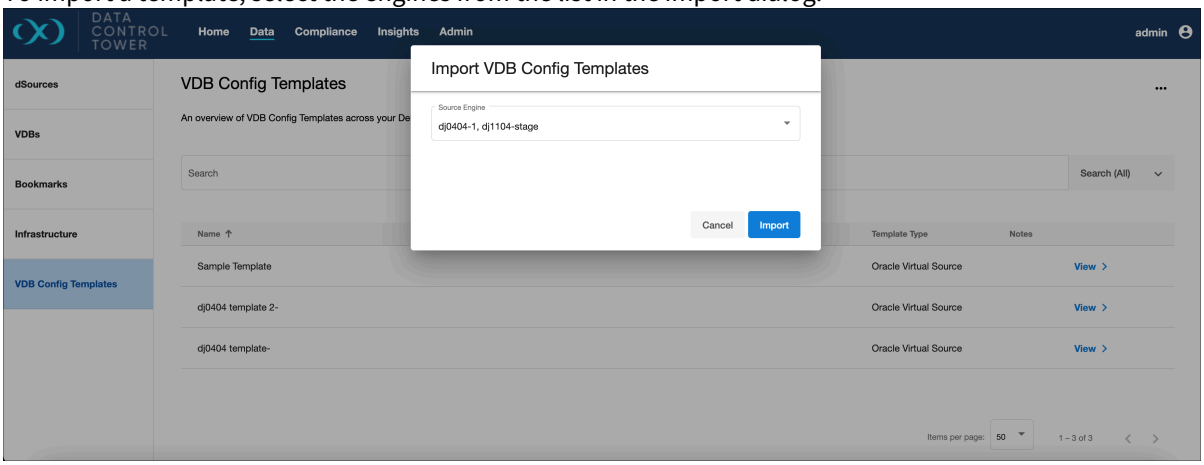
5.2.3.6 VDB templates

Importing and removing imported VDB templates from connected engines is an available action from the **VDB Config Templates** page.



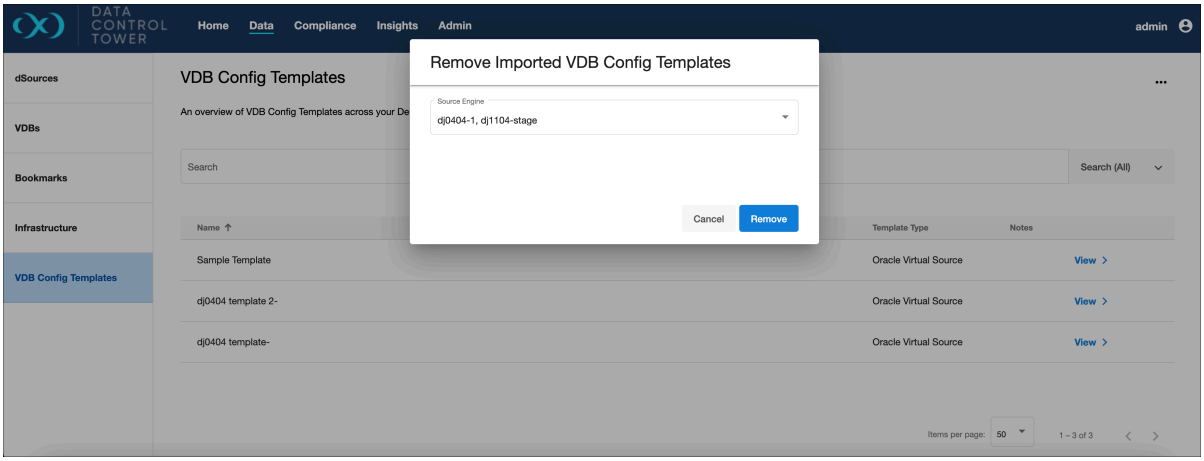
Import templates

- To import a template, select the engines from the list in the import dialog.



Remove imported templates

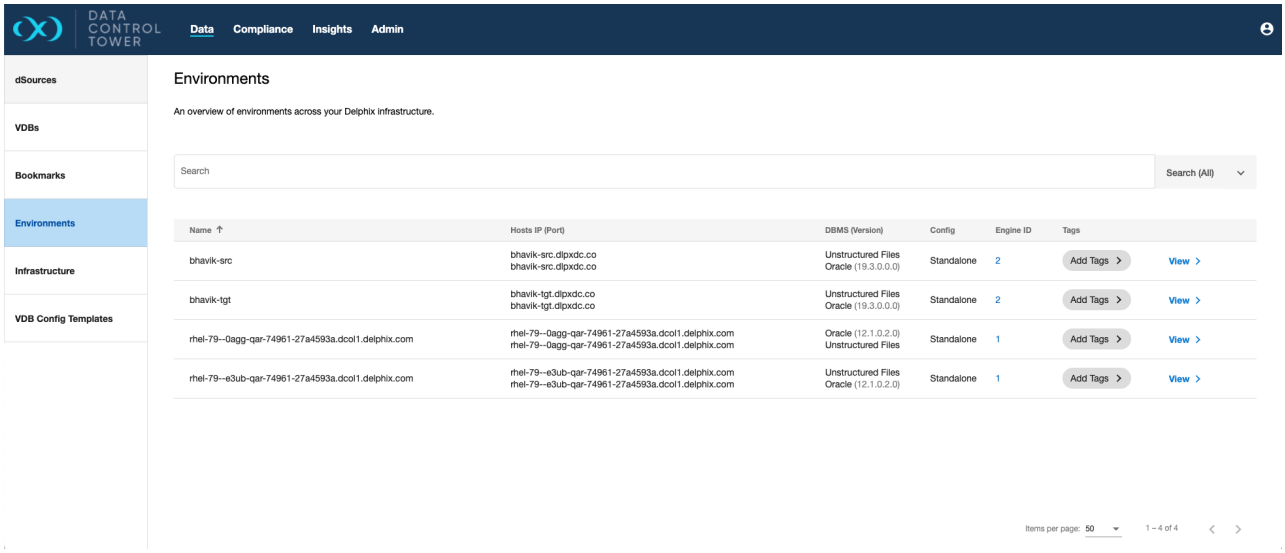
- To remove a template, select the engines from the list in the remove dialog.



5.2.4 Managing environments (Continuous Data)

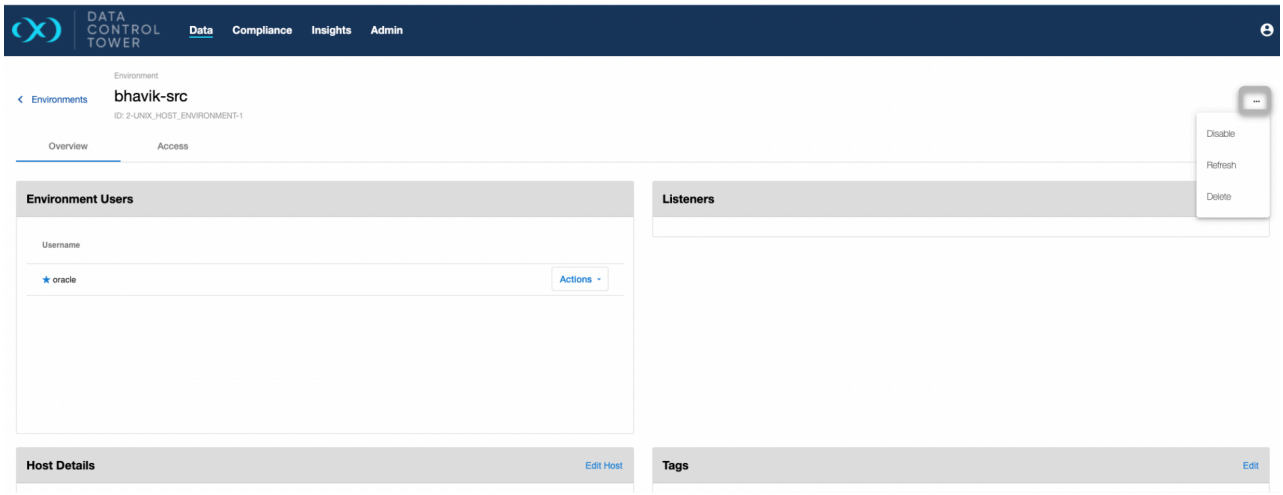
5.2.4.1 Global environments list

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.



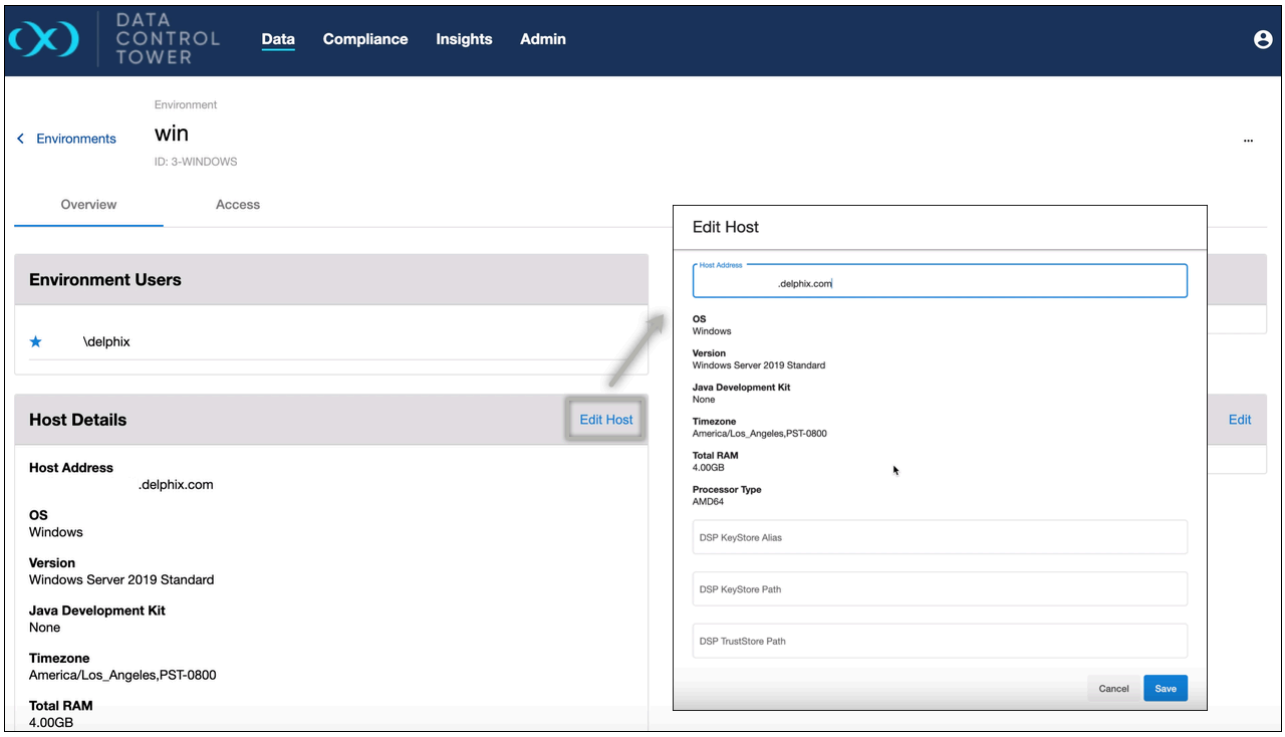
5.2.4.2 Manage environments

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.



5.2.4.3 Edit host details

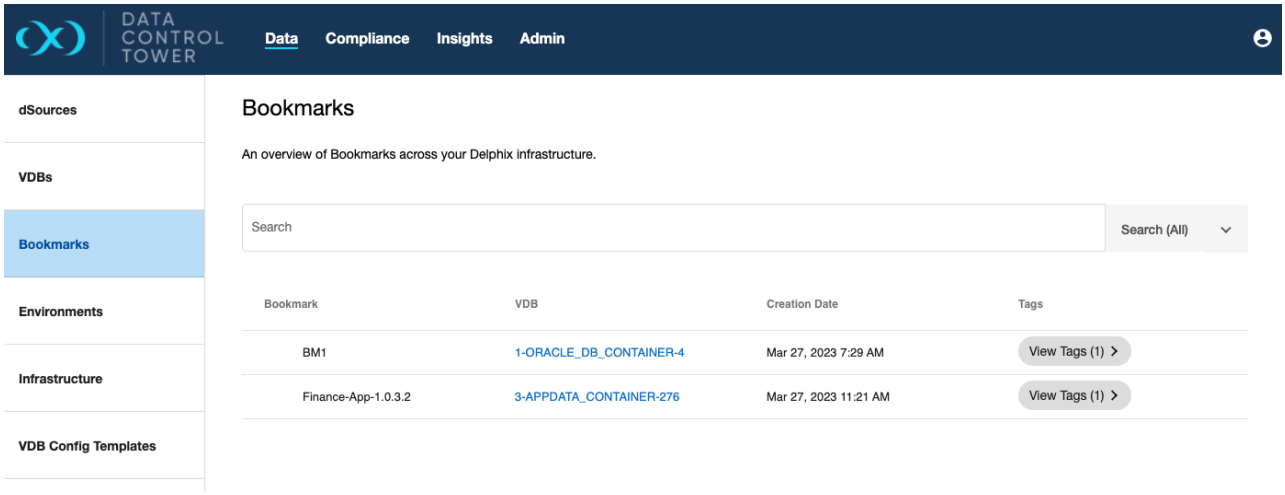
Selecting a standalone environment in the **Data** page shows an **Edit Host** option; it is not yet available for cluster environments. When the button is selected, the host details window will open, showing the input fields that can be edited. Select 'Save' to confirm the changes and close the window.



5.2.5 Managing bookmarks

5.2.5.1 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.



5.2.6 Replication management

5.2.6.1 Replication mappings

5.2.6.2 **Introduction**

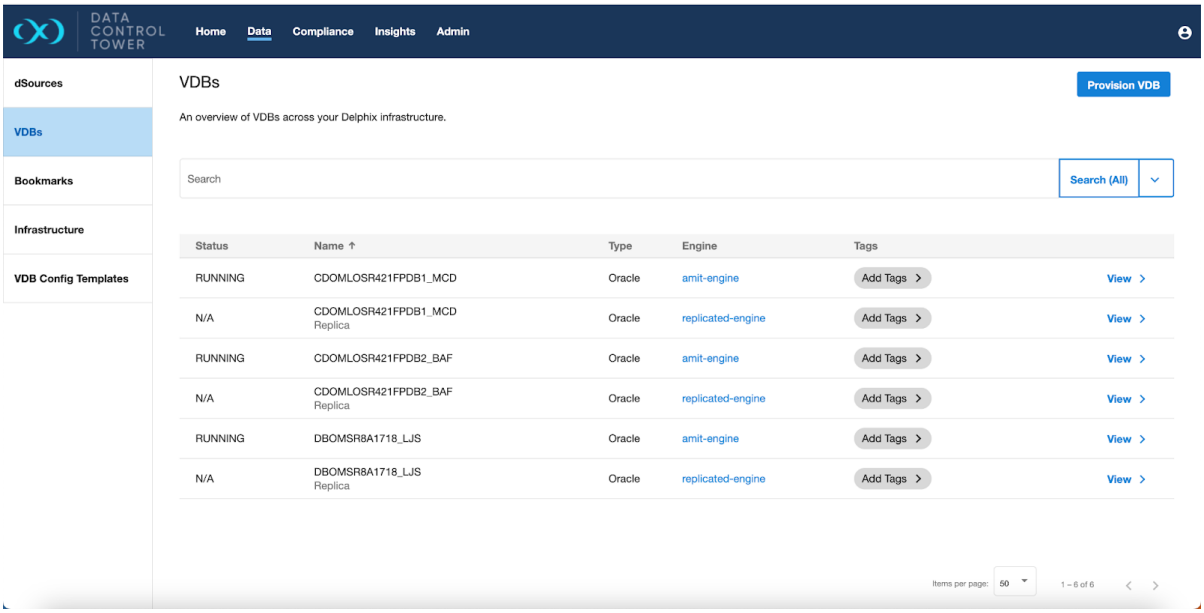
DCT 9.0.0 introduces the feature to differentiate between the replicated objects and original objects in case of a parent and replicated engines, both are registered with DCT.

5.2.6.3 **Prerequisites**

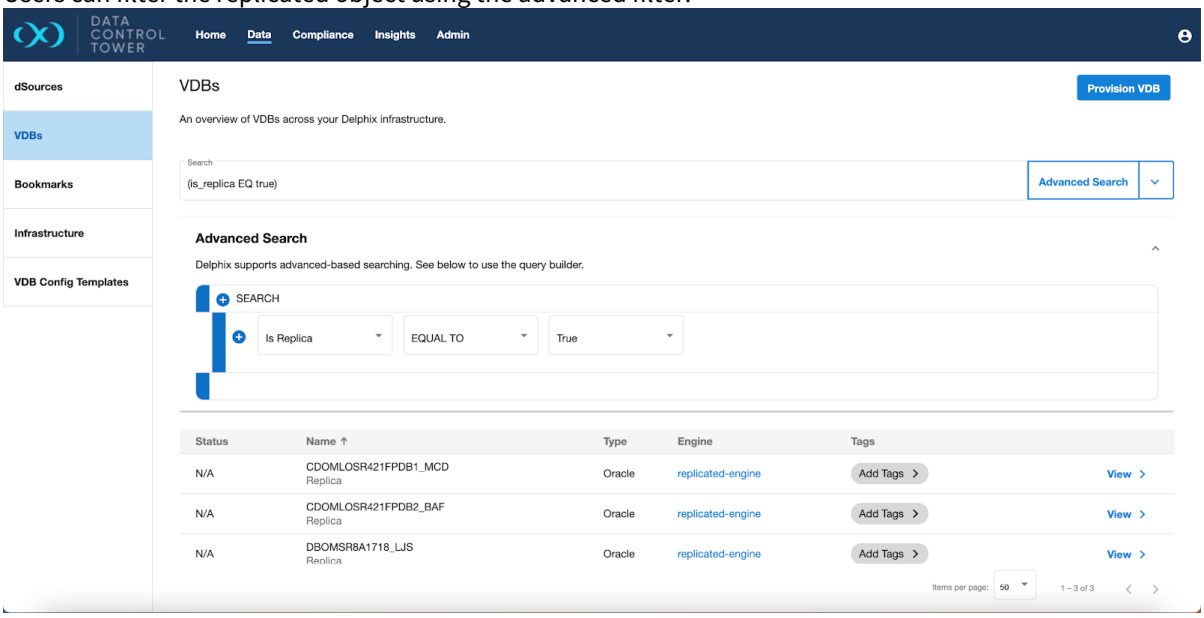
One parent and one replicated engine is required with few replicated dSources and VDBs.

5.2.6.4 User interface

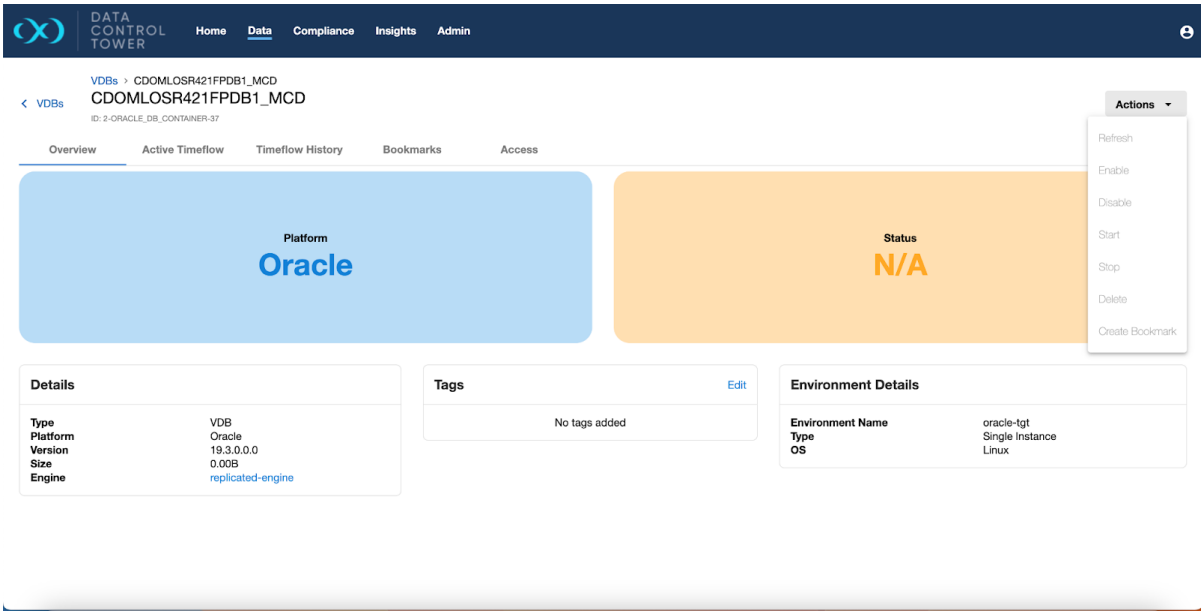
- Convenient separation between the replicated objects(VDB/dSource/environments) and original objects.



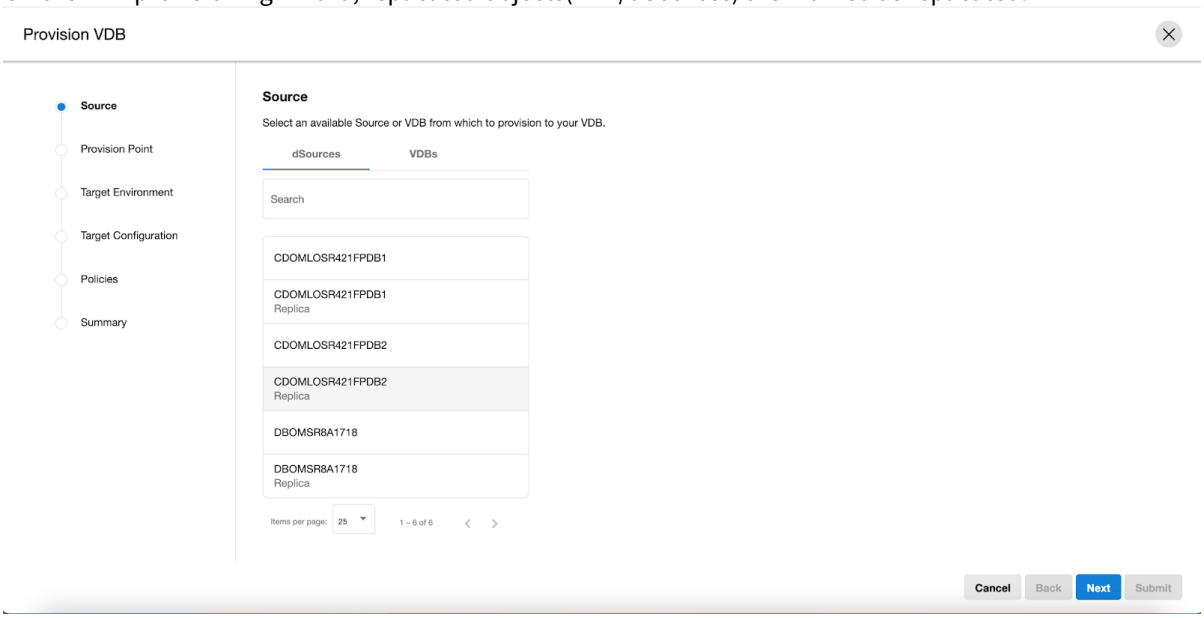
- Users can filter the replicated object using the advanced filter.



- 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.



5.2.6.5 API

Below are the list and search APIs, updated to return three additional fields (`is_replica` , `namespace_id` , `namespace_name`):

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`

Below are the three fields added in response of these APIs:

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.

5.3 Insight reports

5.3.1 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.

5.3.2 VDB Inventory

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.

5.3.3 dSource Inventory

The dSource Inventory report provides users with a comprehensive list of all the dSources created in the DCT platform and their identification metadata.

5.3.4 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.

5.3.5 Compliance Job Metrics

The Compliance Job Metrics 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.

5.3.6 Block Storage

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.

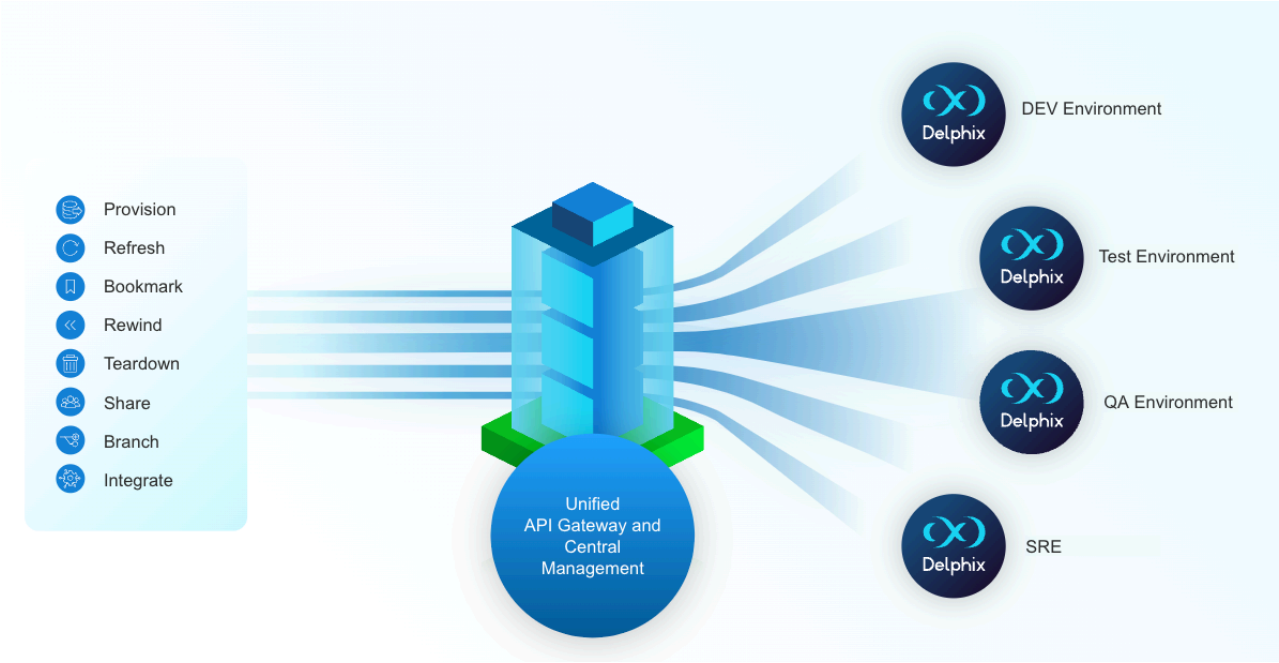
5.3.7 Activity Audit Log Summary

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.

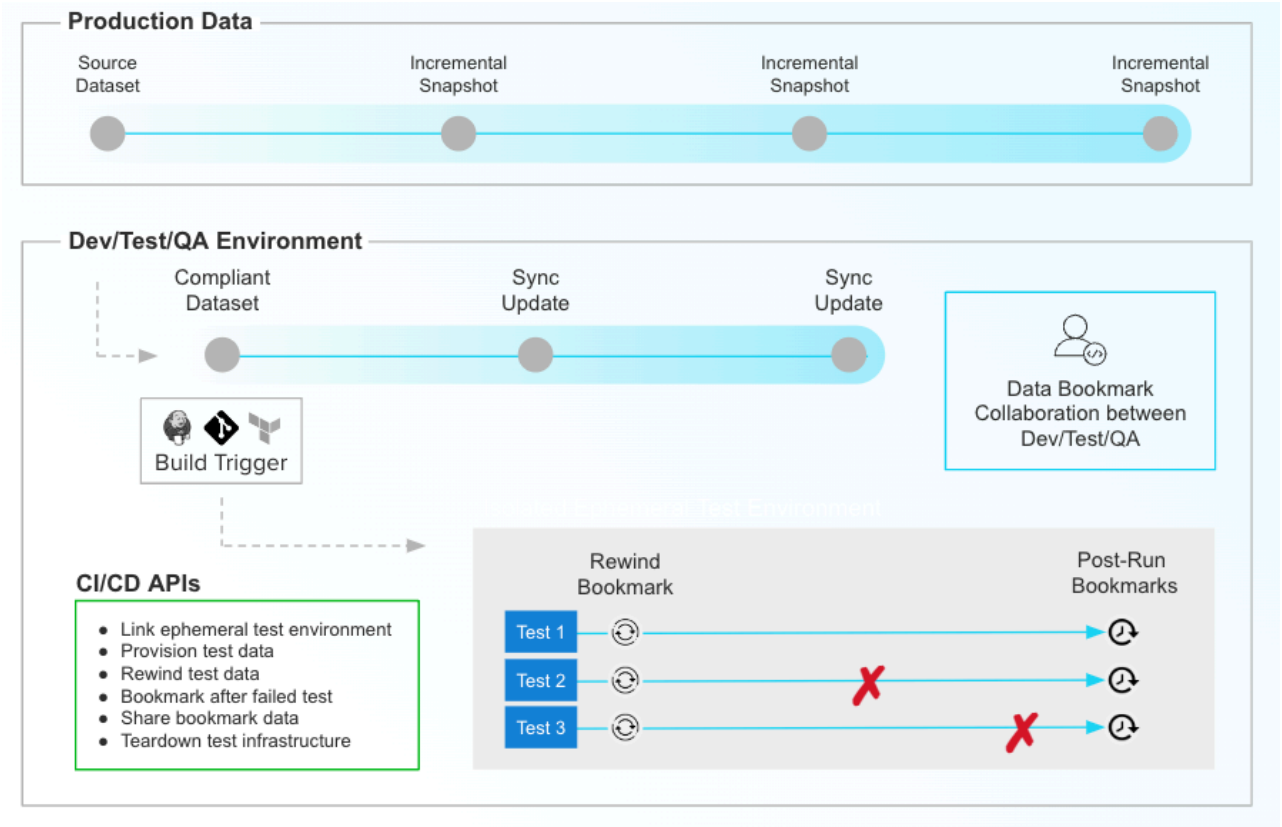
6 Continuous Data workflows

6.1 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.



6.2 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.

6.3 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.

6.3.1 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 155) 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.

6.3.2 Key differences

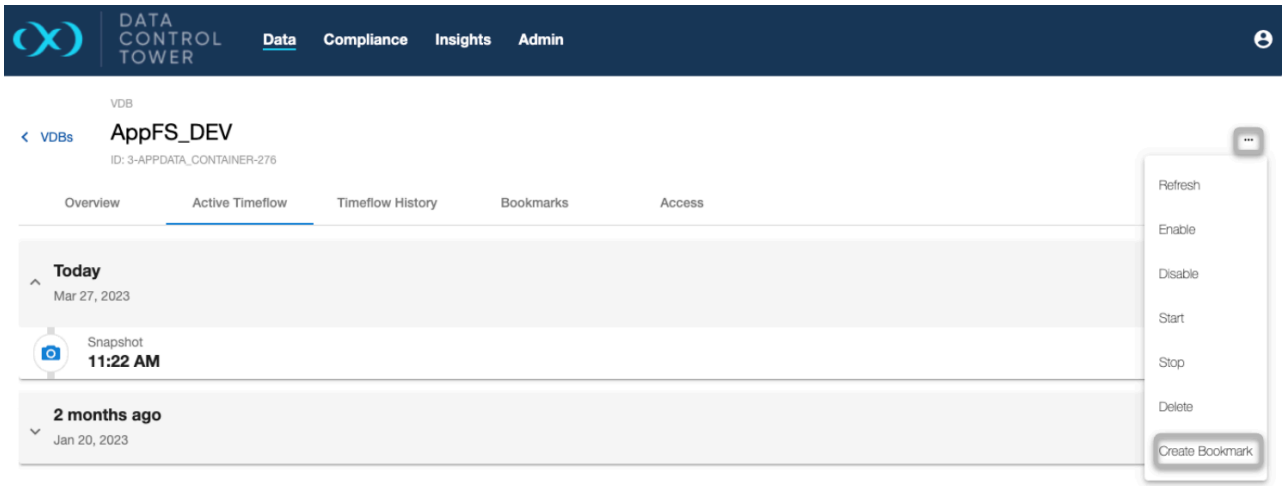
1. **DCT delivers a central interface powered by its converged architecture**
   This means that developers have a single location to log into in order to access and manipulate their virtual data sets.
2. **User experience**
   The DCT developer experience UI has completely been reworked to make developer 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.

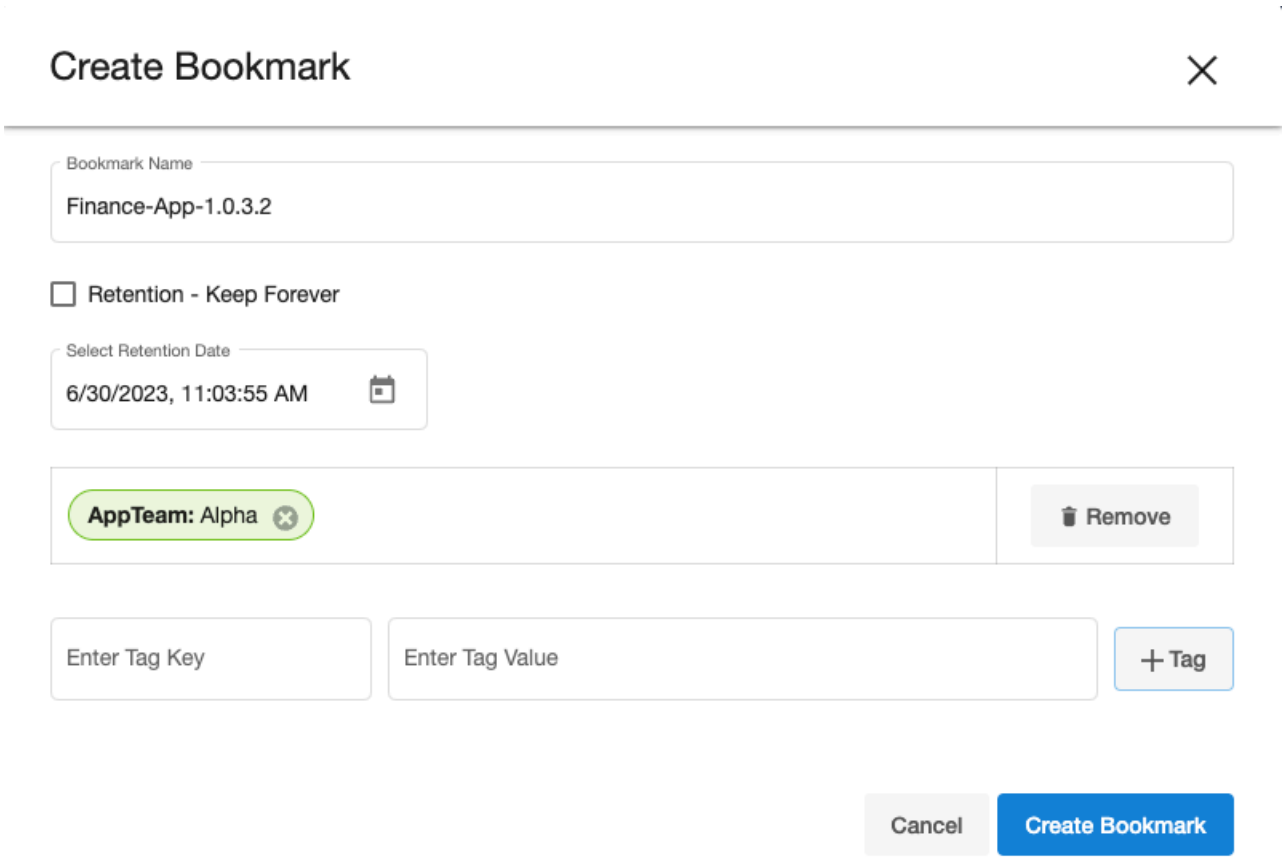
6.4 Creating and managing bookmarks

6.4.1 Create new Bookmark

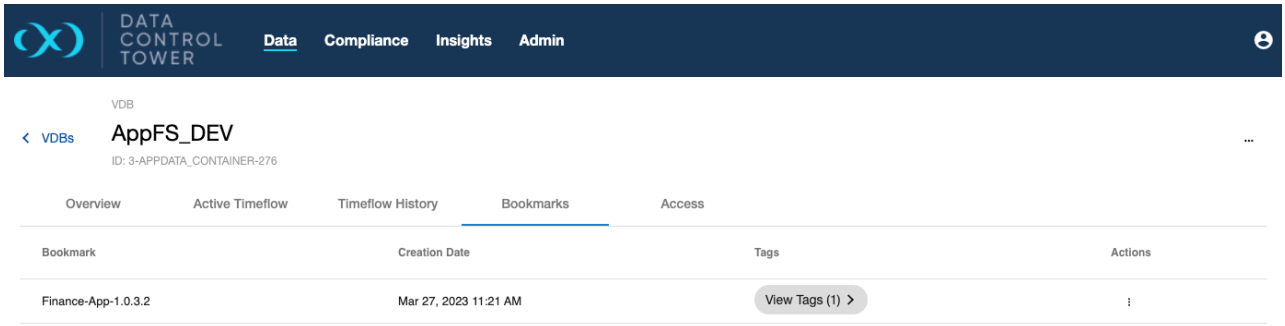
Bookmarks are a critical developer tool that enables the creation of a namable time reference to a snapshot of a VDB or VDB group. Bookmarks for single VDBs can be created from the DCT UI by selecting a VDB and expanding into its detailed view. From the **Active Timeline** view, users can select the 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.



6.4.2 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.

6.4.3 Create a Bookmark at the current time for multiple VDBs

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"
  }
}
```

6.4.4 Create a Bookmark for a VDB from an existing Snapshot

DCT Bookmarks (as of the 6.0 release) 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.

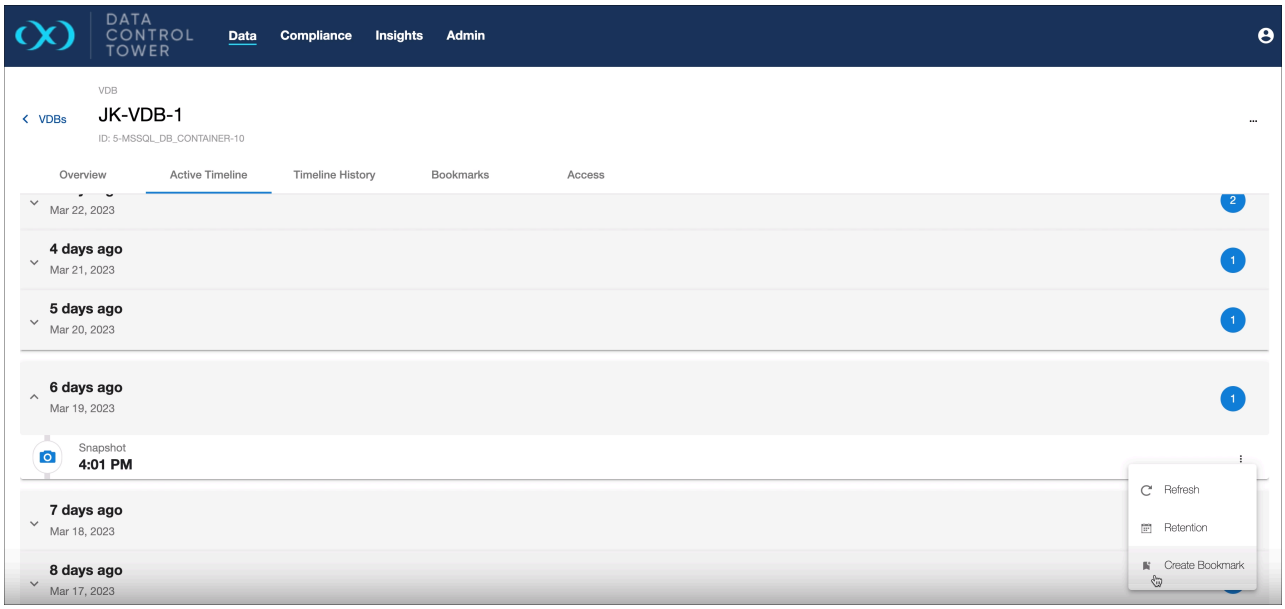
```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/bookmarks' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "MyBookmark2",
  "snapshot_ids": [
    "1-ORACLE_SNAPSHOT-11"
  ]
}'
```

6.4.5 Create a Bookmark for a VDB from an existing Snapshot

Starting in version 7.0, the UI has an option to create bookmarks from the 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.

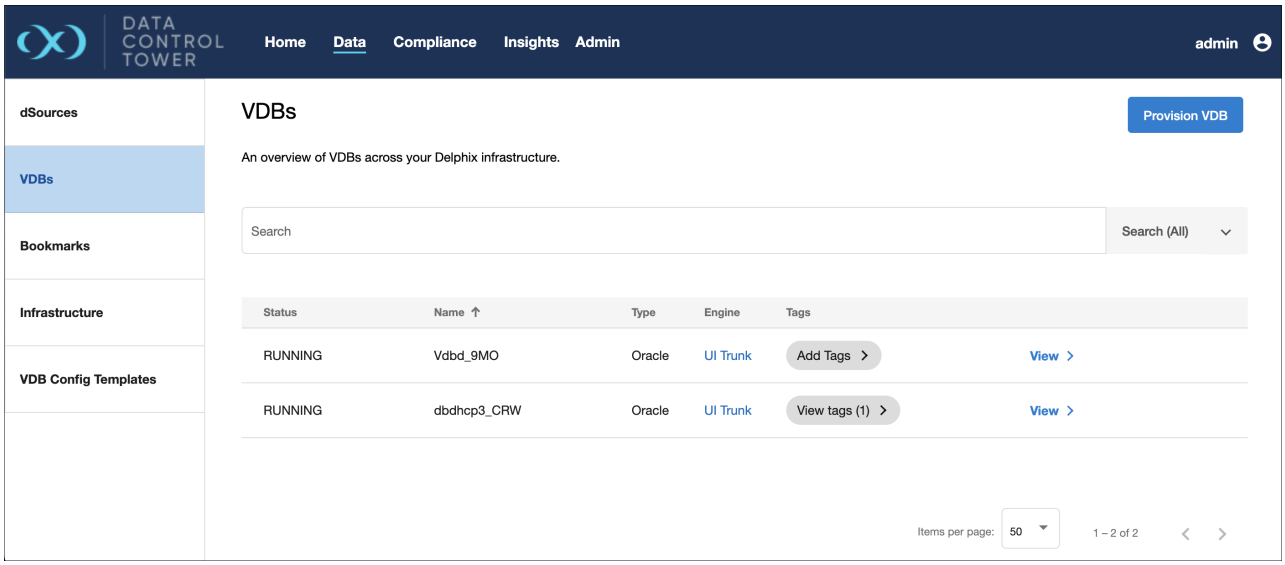
Once the user clicks **Create Bookmark** in the dialogue, the bookmark will be created for that particular snapshot (if all the mandatory fields are completed), else errors will be shown.



6.5 VDB operations

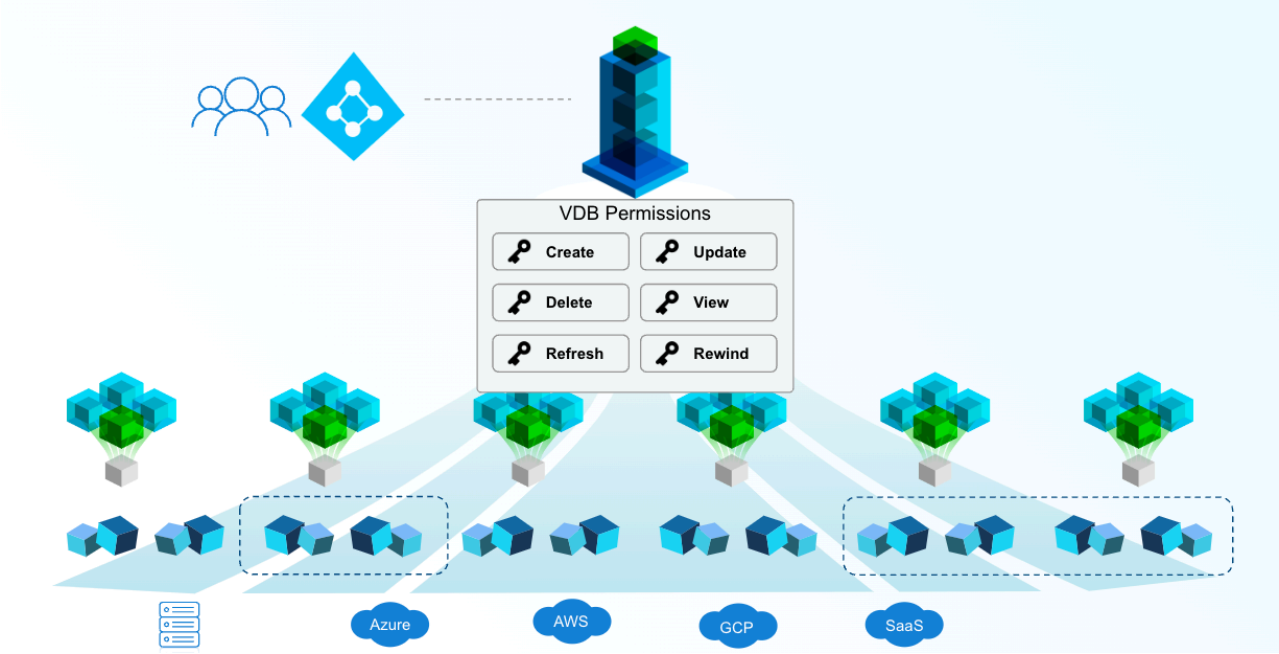
The developer experience will continue to see investment and additional capabilities over the next few releases.

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).

6.5.1 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.

6.5.1.1 Supported database platforms

For Oracle Multi Instance (Single Tenant)

- Editing the Oracle instance number and instance name is not supported, they are set to default values.

For 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.

For MSSql Multi Instance

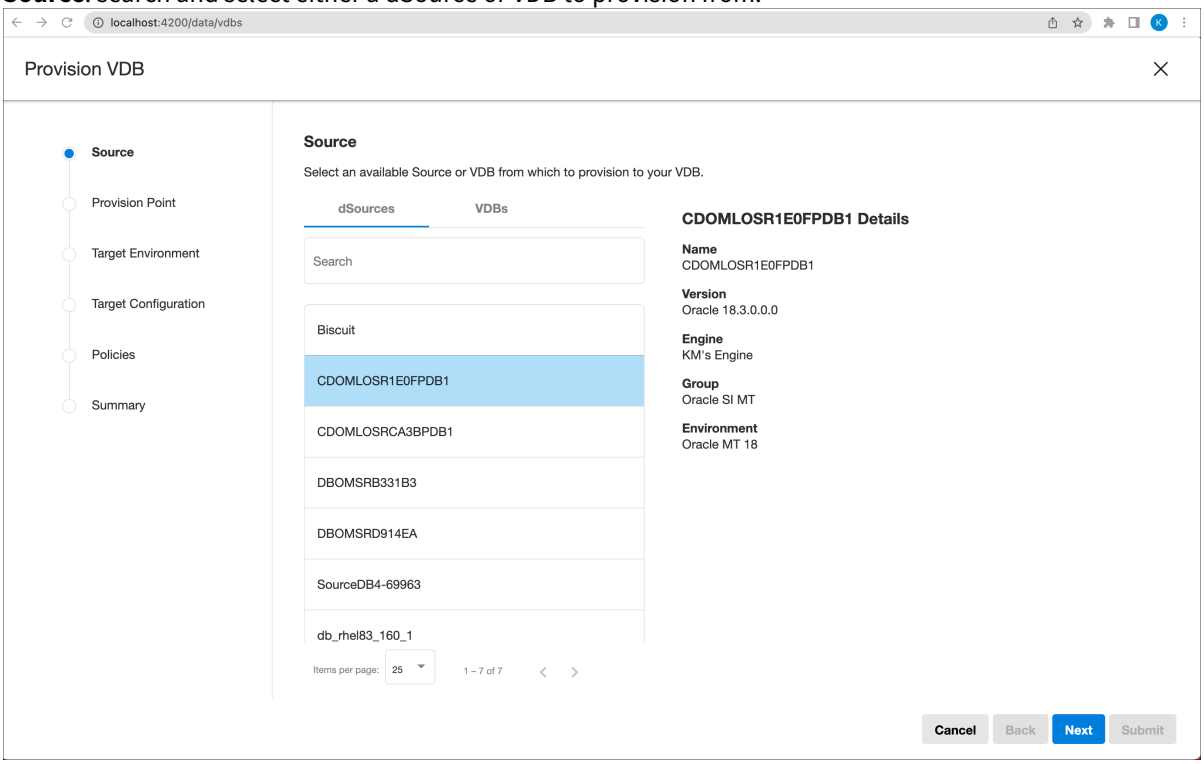
- Drive letters are not supported and are set to default values.

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).

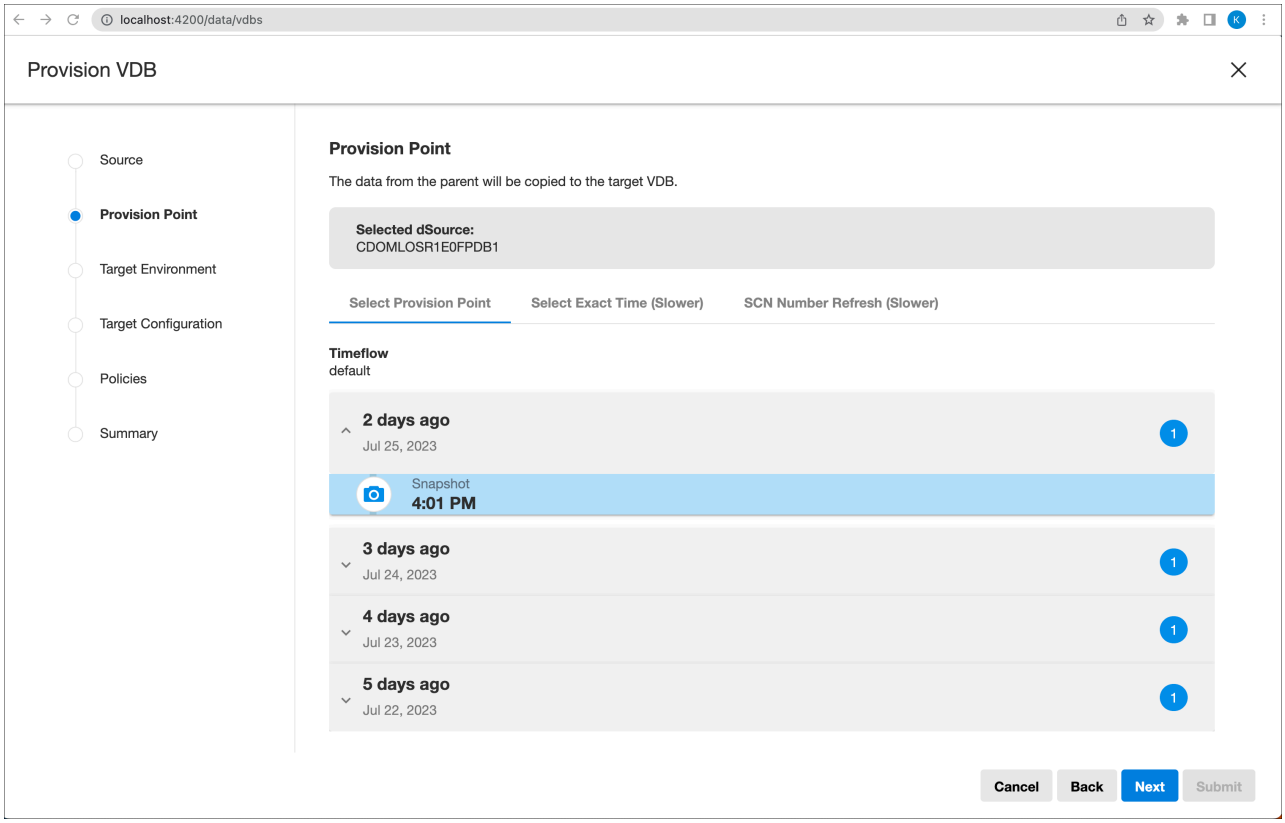
6.5.1.2 Step examples

The provisioning wizard will walk through the following steps:

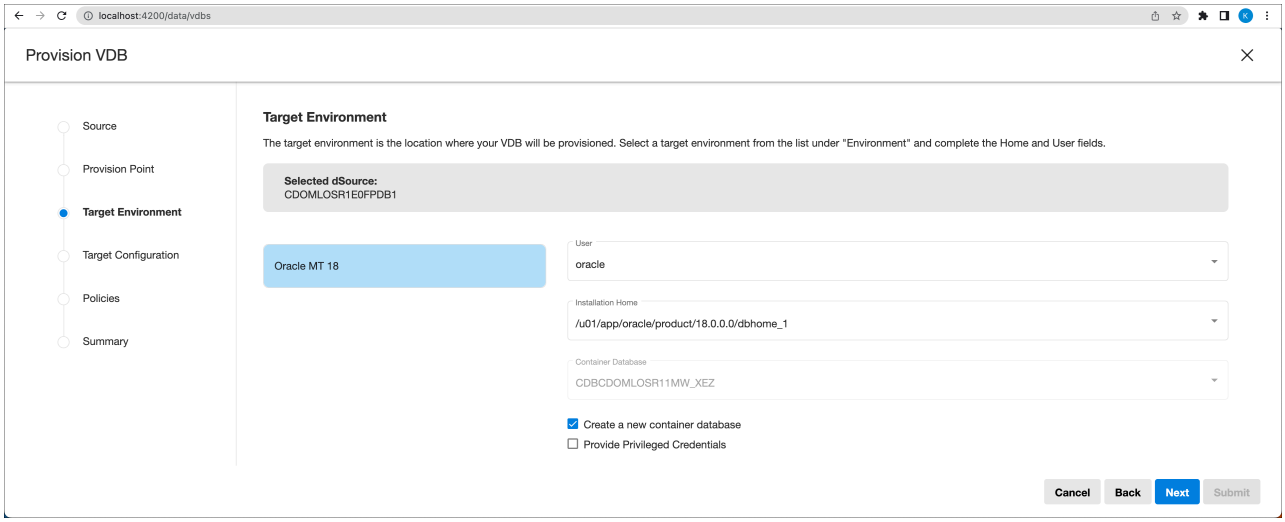
- **Source**: search and select either a dSource or VDB to provision from.



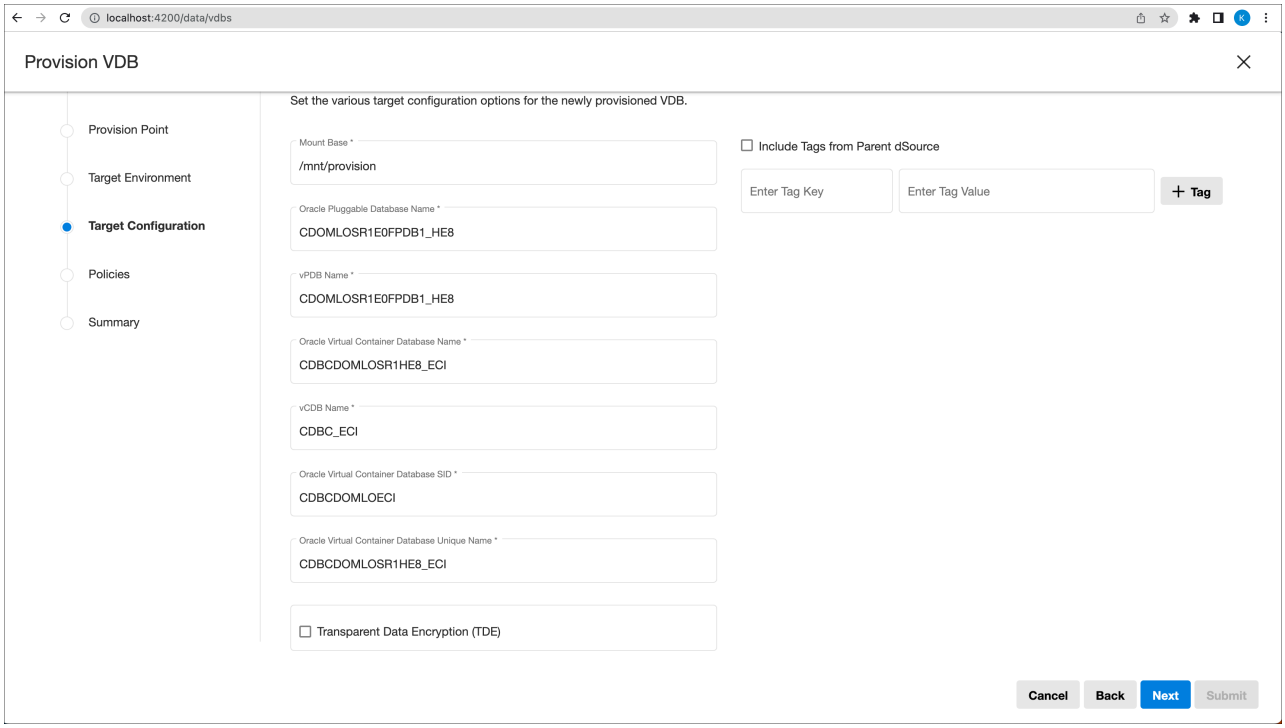
- **Provision Point**: three options for a provision point, similar to a refresh point.
  - A selected snapshot
  - A specific timestamp (closest snapshot to the timestamp)
  - A location ID/number (closest snapshot to the location number)



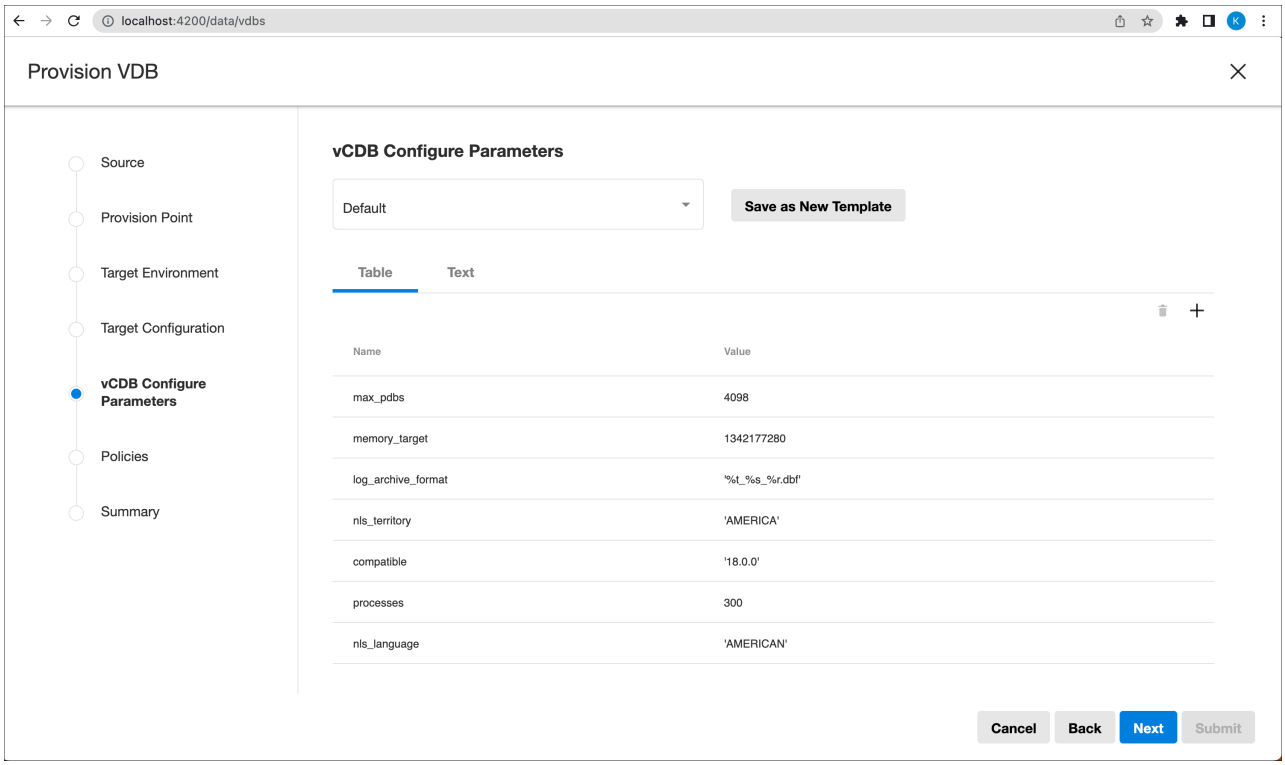
- **Target Environment**: shows compatible environments with compatible repositories, and can optionally provide privileged credentials.



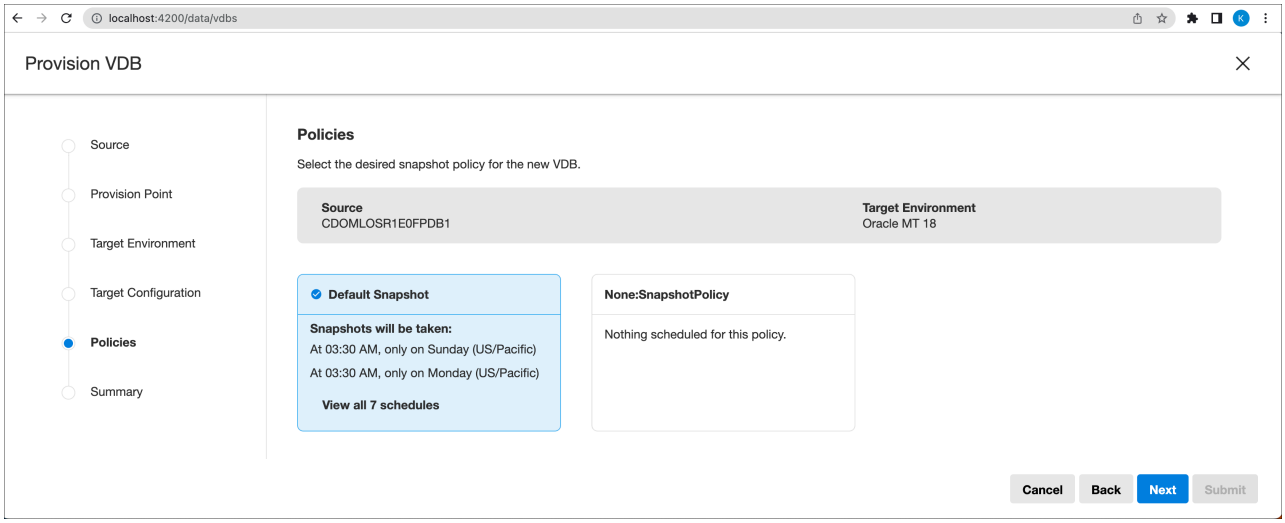
- **Target Configuration**: should be prefilled with default configurations. One thing to note here are the tags which are additive when “Include Tags from Parent” is checked, and you wouldn’t immediately see the tags from the parent in the editor. Tags are added when the **Include Tags from Parent** 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.



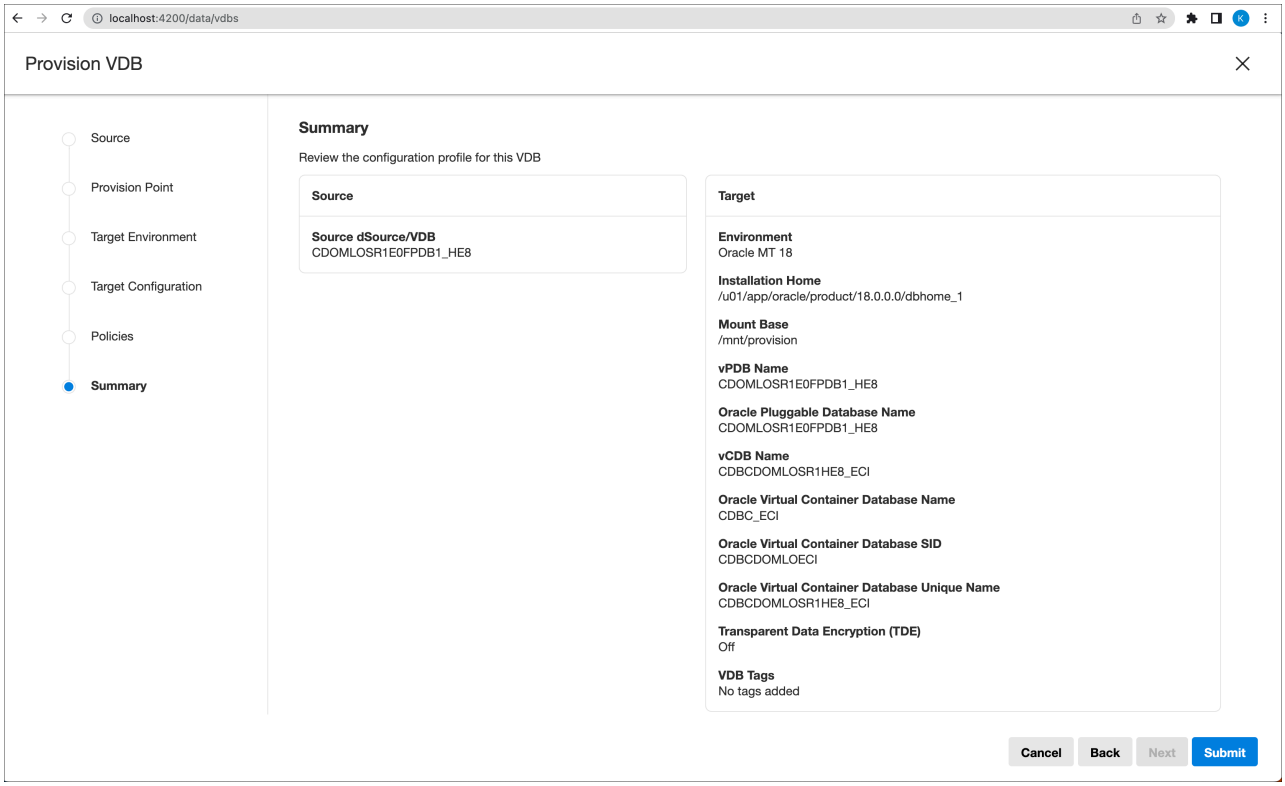
- **vCDB Configure Parameters**: Applicable to Oracle Single Instance (Multi Tenant) with linked vCDB.



- **Policies**: choose a snapshot policy.



- **Summary**: review the selections that have been made.



6.5.1.3 Limitations

TDE and Auto VDB restart are only supported for Oracle version 12.2 or higher.

6.5.2 VDB refresh UI

6.5.2.1 Overview

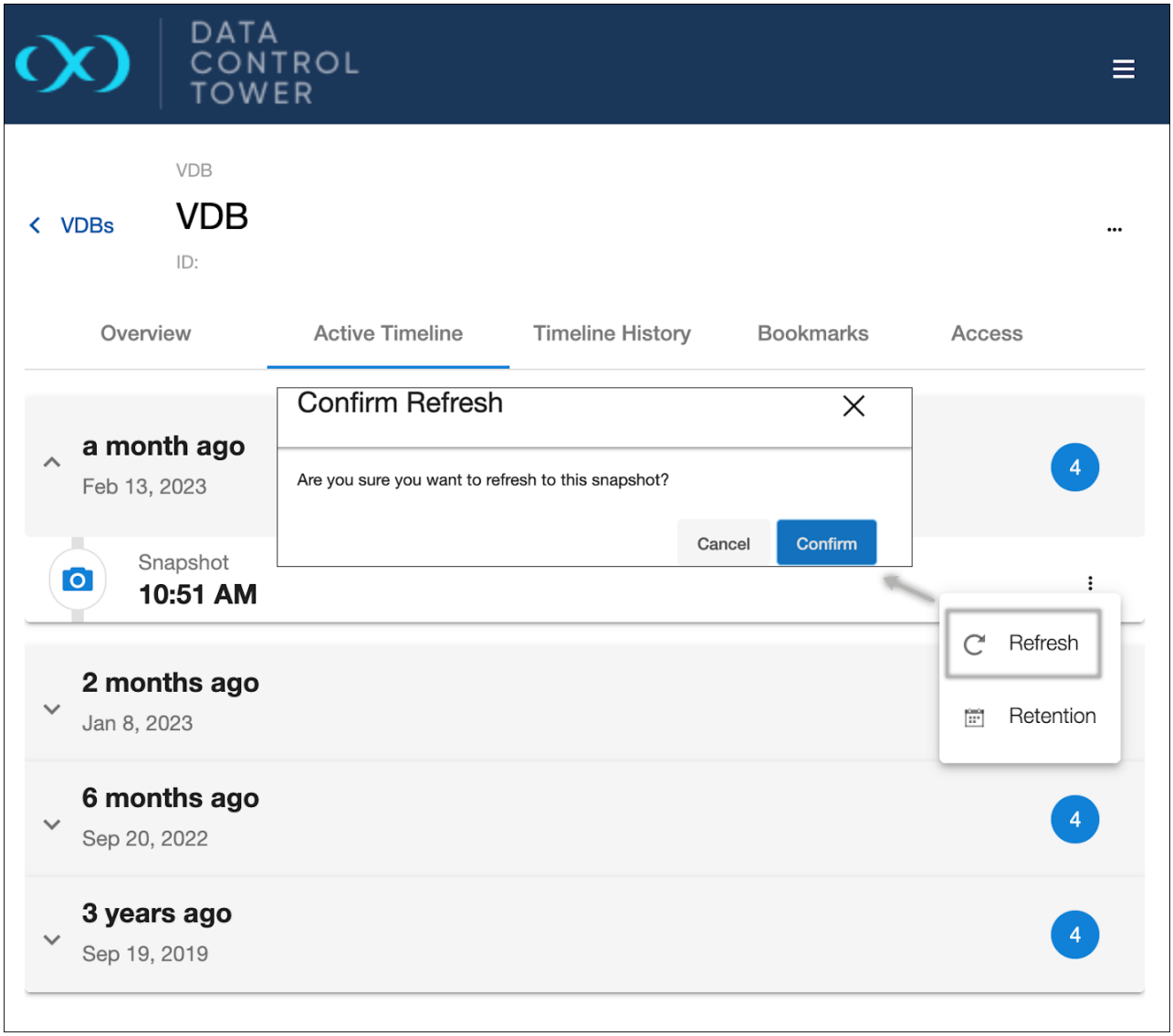
The VDB refresh wizard in the Data Control Tower UI offers important engine refresh operations like:

- **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 (which could be the immediate parent, or parent of parent, etc.) or any VDB sharing the same origin dSource (which could be parent, child, sibling, or similar relationship).
- **Refresh to bookmark** snapshot
  - Refreshes a VDB to a compatible bookmarked snapshot.

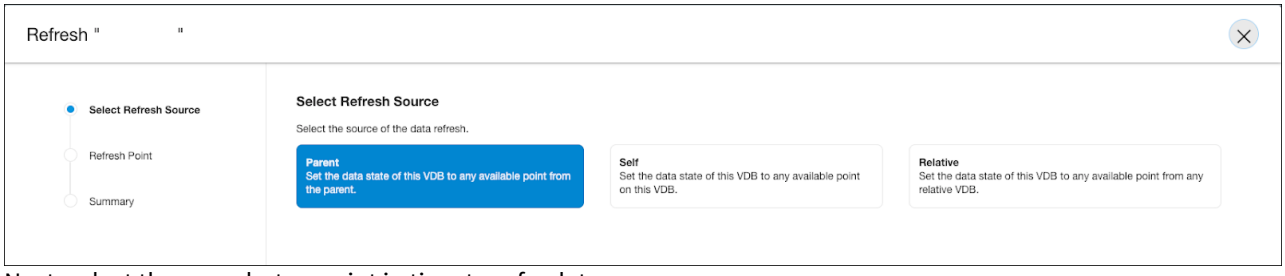
In order to refresh from a dSource or VDB, the account performing the action must have the REFRESH permission on both the VDB being refreshed and the dSource (or VDB) from which the data is being refreshed.

6.5.2.2 User interface

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.



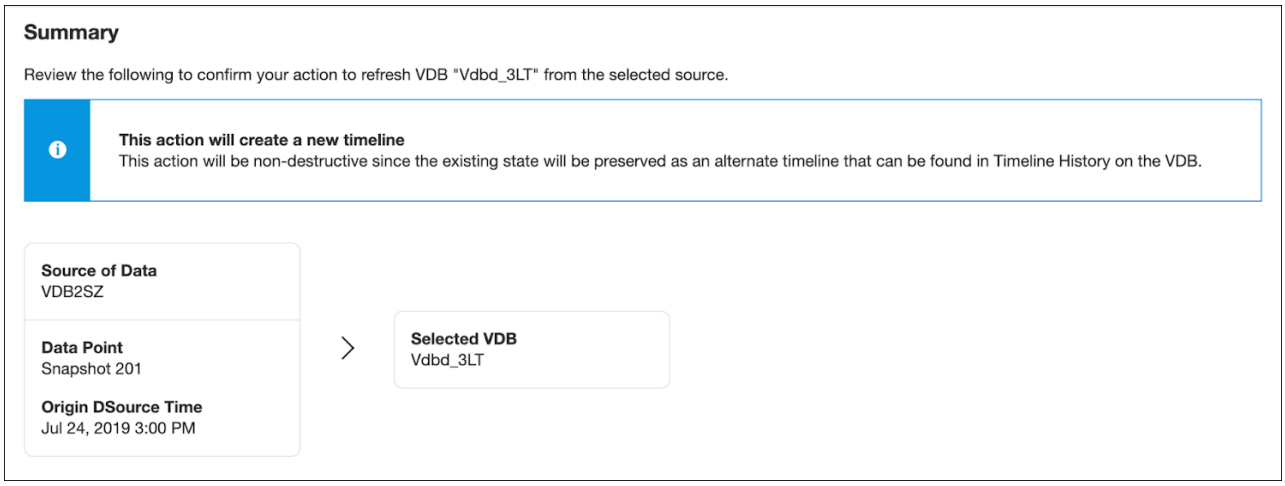
All three refresh types can be performed by opening the refresh wizard on the VDB details page. In the first step, select the refresh source:



Next, select the snapshot or point in time to refresh to:

- For self-refresh, a timeflow from the VDB's own history must be selected, and then a snapshot within the selected timeflow.
- For parent refresh, a timeflow from the VDB's provision parent's history must be selected, and then a snapshot within the selected timeflow.
- For relative refresh, first a relative dSource or VDB must be selected, then a timeflow, then a snapshot.

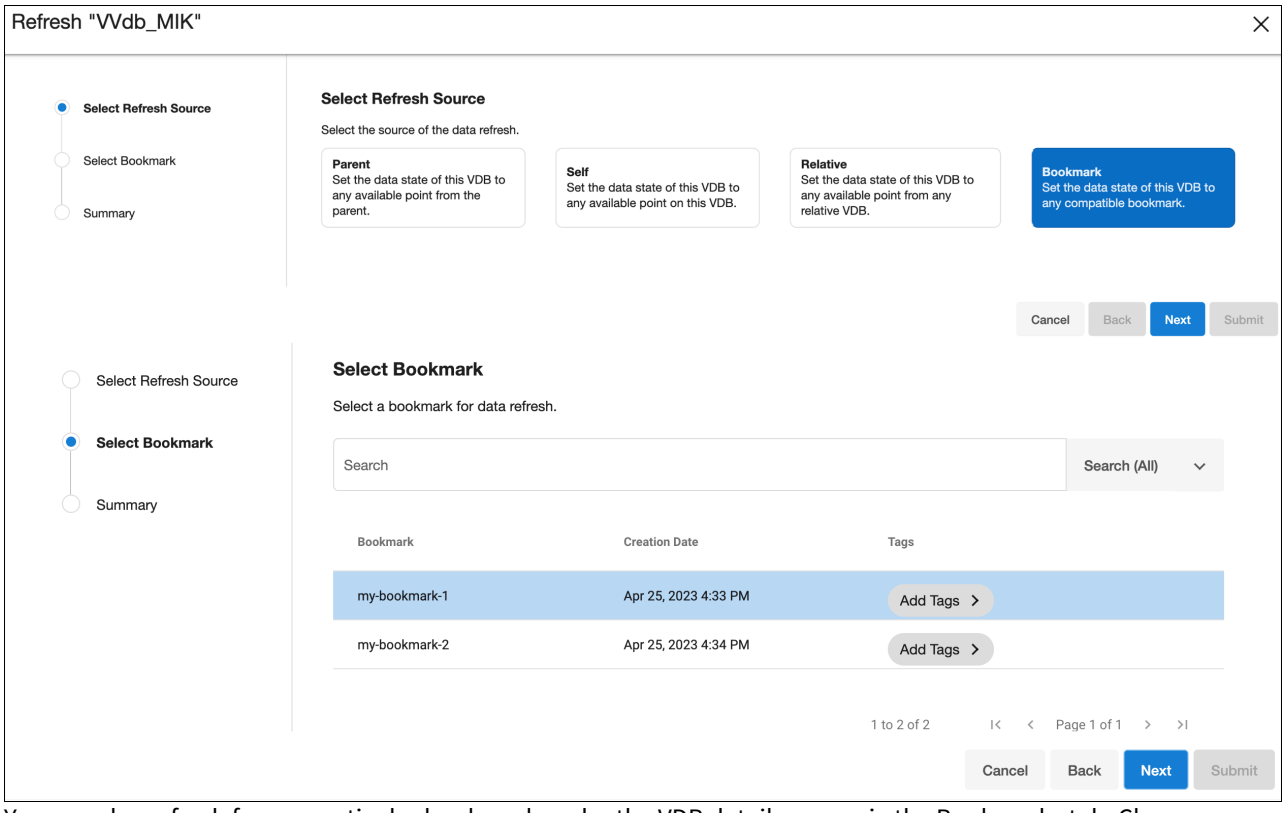
For all three refresh types, review the summary page once configurations are complete, then click submit:



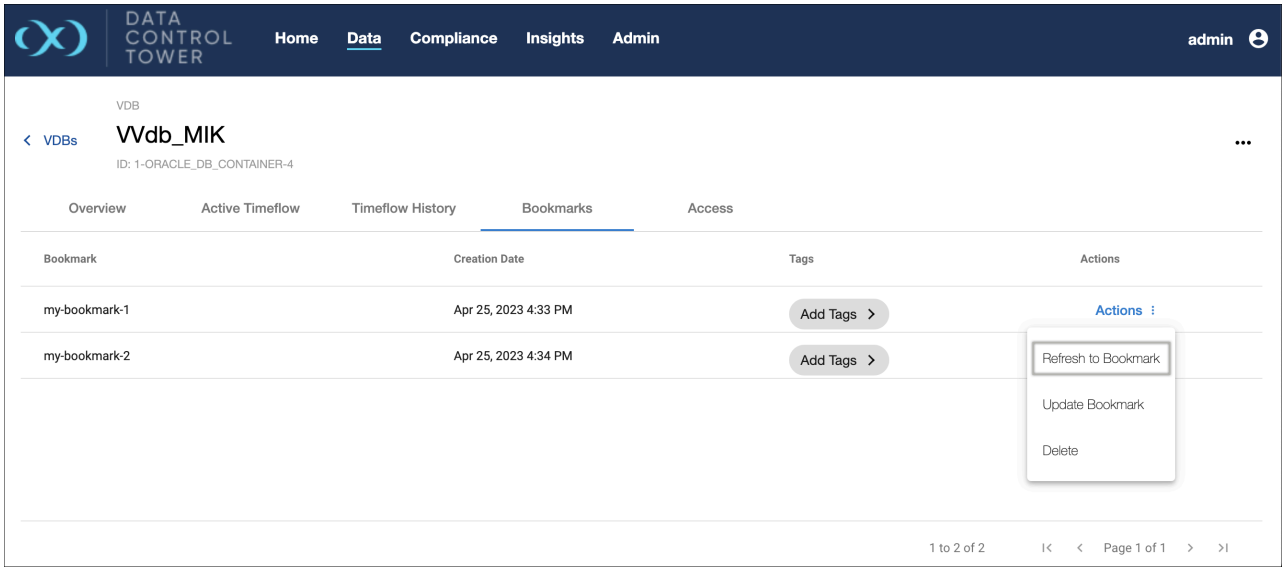
Refreshing happens asynchronously and takes a various amount of time. The DCT UI does not currently show asynchronous job progress or errors, please refer to the engine UI for this function.

Refresh to Bookmark

Bookmark is available as a **Select Refresh Source** option in the refresh wizard, which allows you to refresh from compatible bookmarks. Choose a bookmark from the list and click **Next**.



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**.



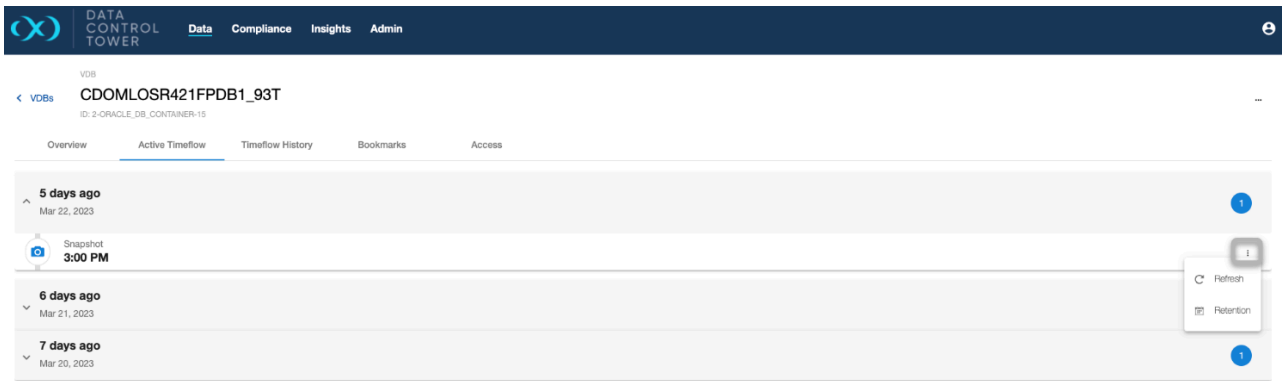
6.5.3 Active timelines UI

6.5.3.1 Active Timeline View

Active Timeline can be found by selecting “details” for any VDB located on the VDB list page in DCT and selecting the “Active Timeline” tab. This view serves as an operations console for any user with the appropriate entitlements to see and act upon the identified VDB granted by the DCT access control system.

This view shows 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.

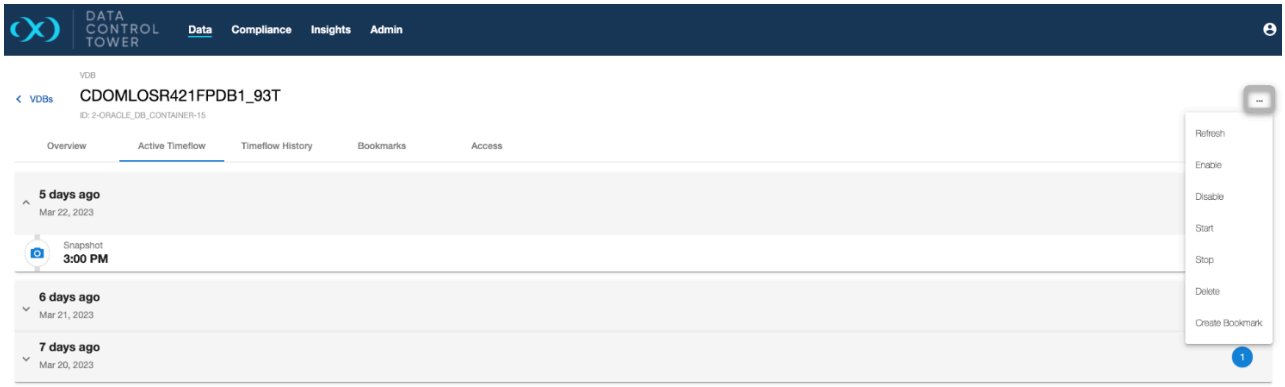
> ⚠ **Contextual snapshot menu**: By selecting the ellipsis next to the VDB of interest, users can refresh to a snapshot or create a bookmark (a nameable and shareable snapshot reference.



Alternatively, users can perform common Continuous Data actions via the ellipsis in the top right of the Active Timeline screen. This menu will offer VDB-wide, generalized operations including:

- **Refresh**: This is a generalized operation to realign your VDB's timeline with another. More information can be found in the Refresh a VDB(see page 151) article.

- **Start/Stop**: A way to manage target host bandwidth.
- **Enable/Disable**: An administrative operation required for upgrades/migrations.
- **Create Bookmark**: Create a special reference to a snapshot that has a custom name and retention period.



6.5.3.2 Additional notes

- **Refresh using the DCT UI**
  Data Control Tower has consolidated all contextualized time-based operations (e.g. refresh, rewind, rollback, restore, reset, etc.) across Continuous Data and Developer products into a general "Refresh". By clicking refresh, users will be prompted on what timeline they would like to align: Parent, Self, or Relative. From there, users will be taken to a wizard that will give relevant timeline options and points in time to perform the refresh.
- **Start/Stop**
  Starting and Stopping are geared toward bandwidth management in a target environment. Stopping a VDB will place it in stasis such that it can't be accessed, but also won't consume bandwidth. Starting it back up will re-enable it for regular activity.
- **Enable/Disable**
  Enabling and Disabling are geared toward administrative operations such as VDB migration or upgrade. Disabling a VDB removes all traces of it, including any configuration files, from the target environment to which it was provisioned. Re-enabling the VDB will restore those configuration files.
- **Create Bookmark**
  Bookmarks serve as a human-referenceable representation of time that can work for a single VDBs or VDB groups. Bookmarks also have the capability to be shared (**Refresh to relative** in the Refreshing a VDB(see page 151) article). From the top level menu, users will be able to name the reference and set a unique retention period.
  Bookmarks do not appear in the Active Timeline, to access existing bookmarks, users will have to navigate to the "Bookmarks" tab on the VDB detail view.

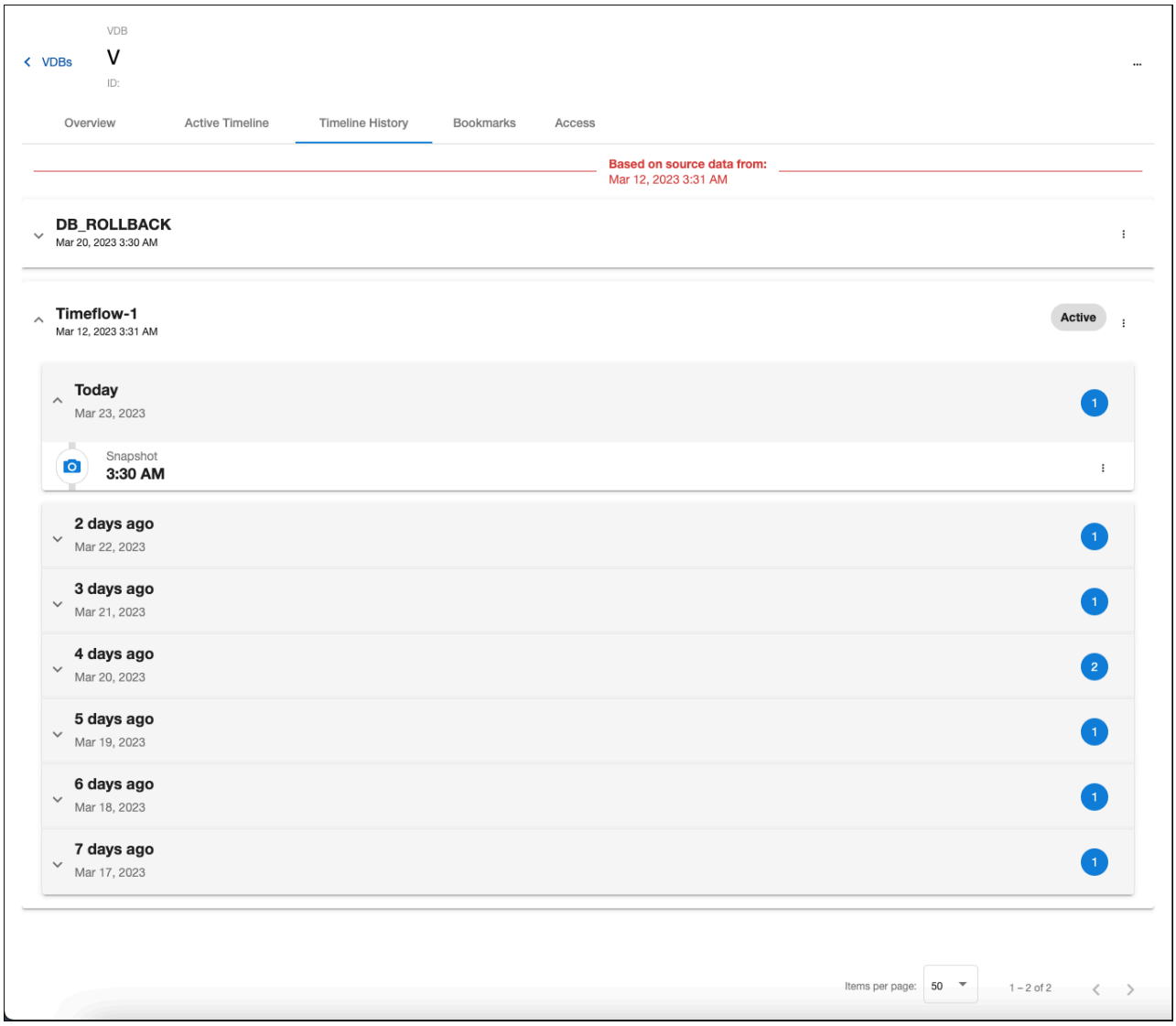
6.5.4 Timeline history UI

6.5.4.1 Timeline History view

The timeline history view can be found by selecting “details” for any VDB located on the VDB list page in DCT and 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 is a critical resource for developers as these can contain information like past test results. Using this page, developers can curate and access the complete chronology of their testing efforts with operations such as renaming individual timeflows, make active, refresh to snapshot, and create bookmark to drive organization and access.

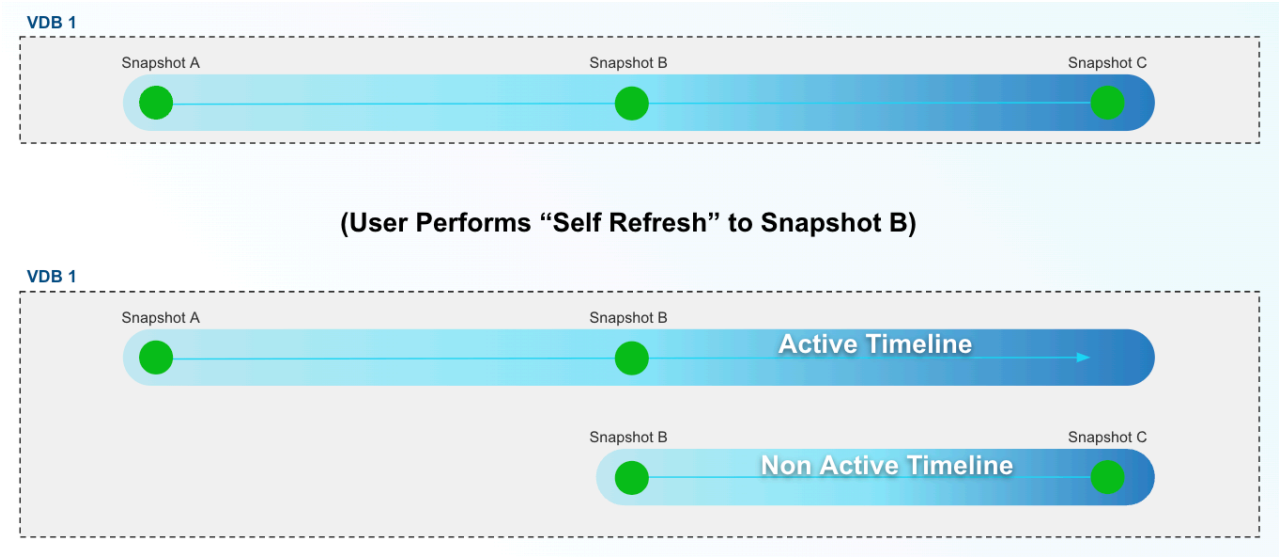
6.5.4.2 The Timeline History user interface

If entitled via the DCT Access Control system, developers can see and act upon VDBs in the VDB list view using the timeline history tab under "details". 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.



6.5.4.3 Non-active timelines

Non-active timelines (often referred to as “timeflows” in Self-Service) are a critical aspect of how Delphix Virtualization works and an important Developer Tool. 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 scenario, a developer has performed a “self-refresh” (formerly referred to as a “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 that are represented on the non-active Timeline, he or she can activate the past timeline by hitting the “Make Active” button in the DCT User Interface 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).

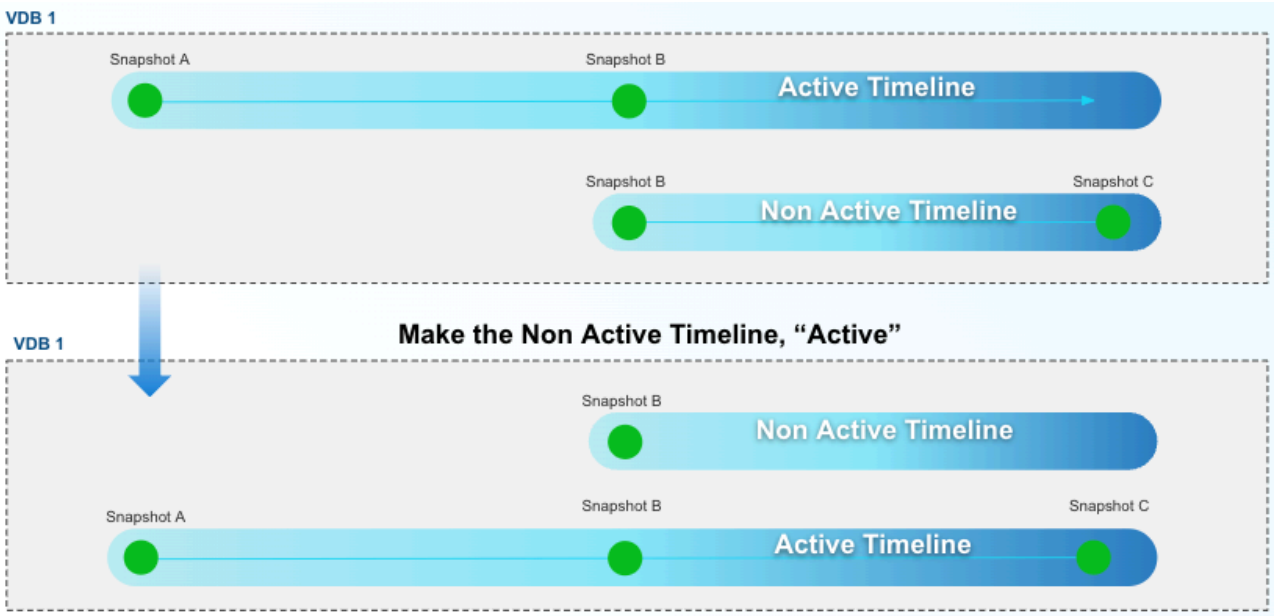
6.5.4.4 Time concepts within the Timeline History Tab

DCT’s new user interface 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.

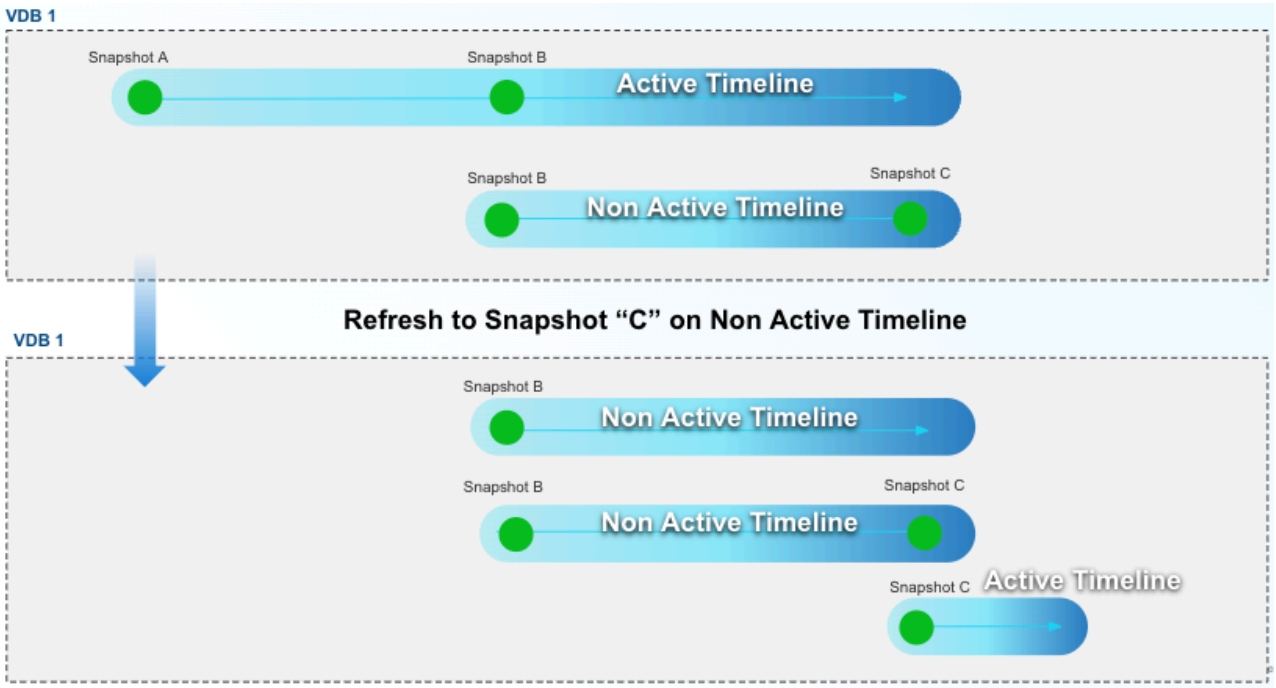
6.5.4.5 Timeflow operations

“Make Active”, a developer can reference any past timeflows and data represented therein by making a timeflow “Active”. This can be done by simply clicking on the timeflow of interest and selecting “Make Active”.



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 major underlying differences as 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.

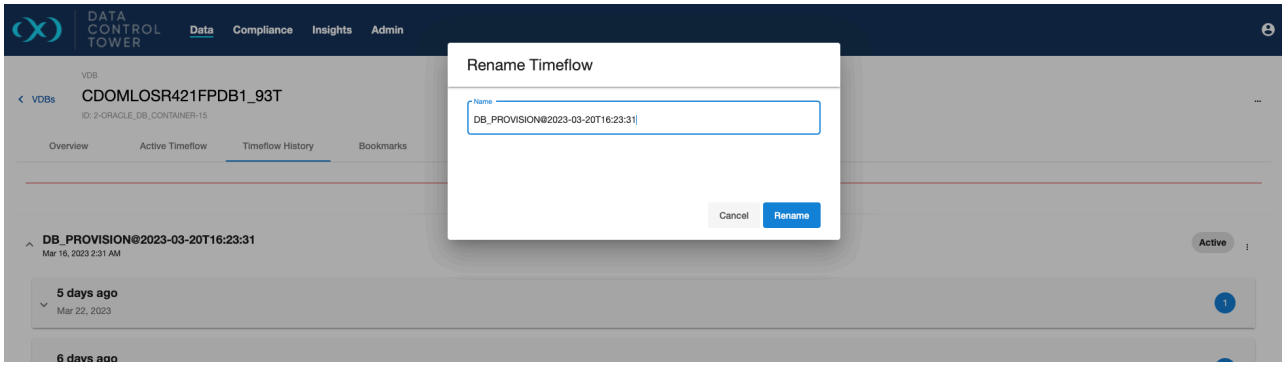


Create bookmark

On a snapshot will enable developers to convert that snapshot into a DCT Bookmark (see page 131)that enables developers 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.

Rename timeflow

Developers can curate time flows to correspond to tests such that they can easily reference results data. This action can be performed by clicking the ellipsis on the time flow 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 Timeflow name by default. Saving the dialog kicks off a job to rename the timeflow. The changes may take a short time to be reflected in the UI.



Delete timeflow

Developers have the option to delete time flows via the UI by selecting the ellipsis next to the corresponding time flow and selecting "delete". When clicked, this opens a confirmation dialog asking if the user wishes to delete the timeflow. If the dialog is confirmed, a job is kicked off to delete the timeflow. Note: This menu item is disabled for the currently-active timeflow.

6.5.4.6 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 timeflow documentation will provide some general examples:

Get a list of a VDB's timeflows

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'\''"
}'
```

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"
}'
```

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'\''"
}'
```

7 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.

7.1 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[44]: **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.

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.

44 https://documentation.delphix.com/continuous-compliance/docs/working-with-multiple-masking-engines

7.2 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(see page 170)), DCT will auto update synced jobs whenever one of those jobs has been modified (i.e. updated rule set, new algorithms, etc.).

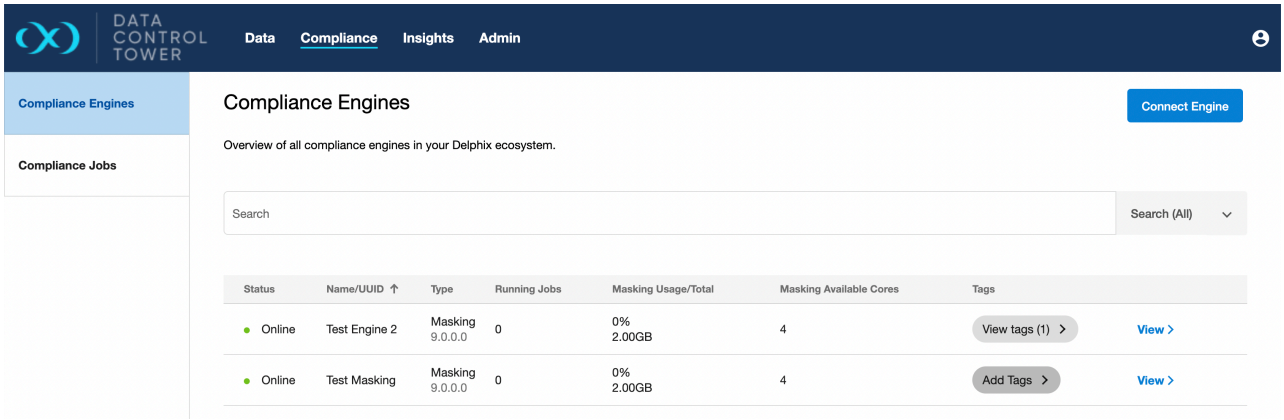
In order to transfer connector credentials with a job as part of the job move, you will need to associate those credentials using the connector credentials API. See sample code below on how to update credentials.

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"
}'
```

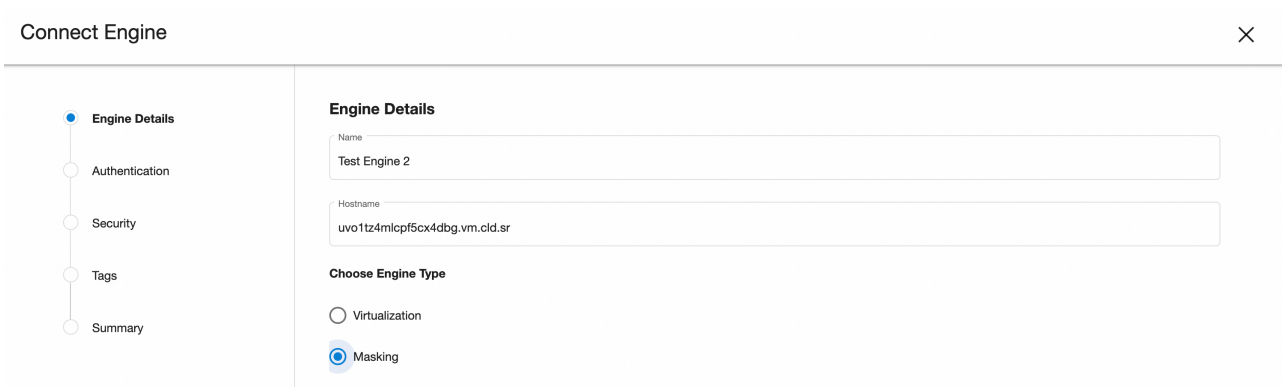
7.3 Managing engines (Continuous Compliance)

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.



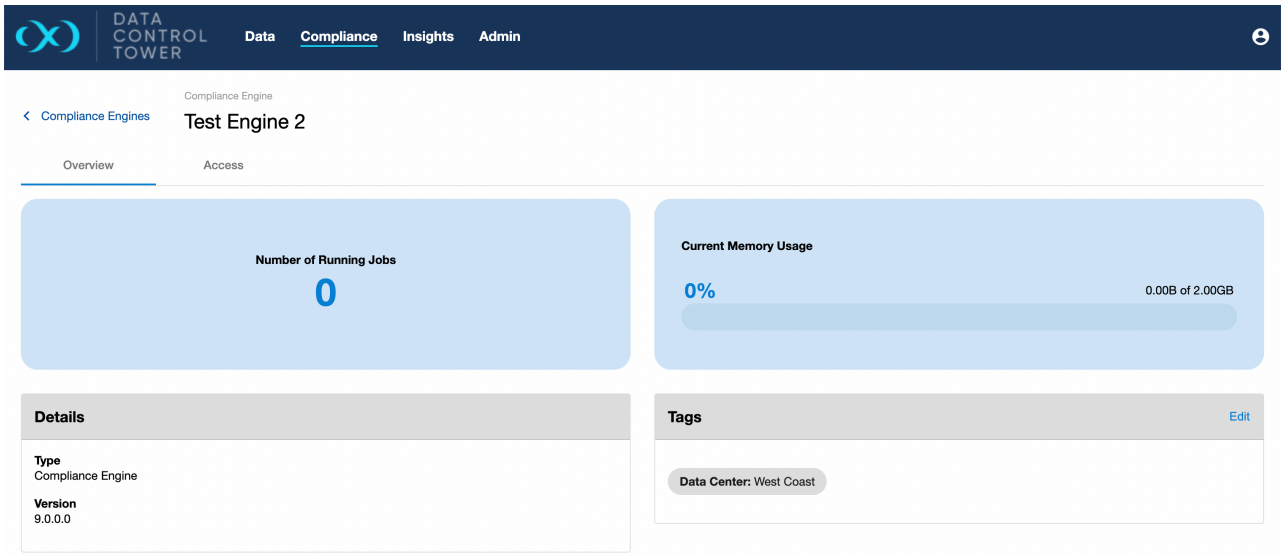
From this screen, administrators can manage engine connects via the “Connect Engine” button on the top right corner. By clicking this button, the below window will appear asking for connection details.

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.



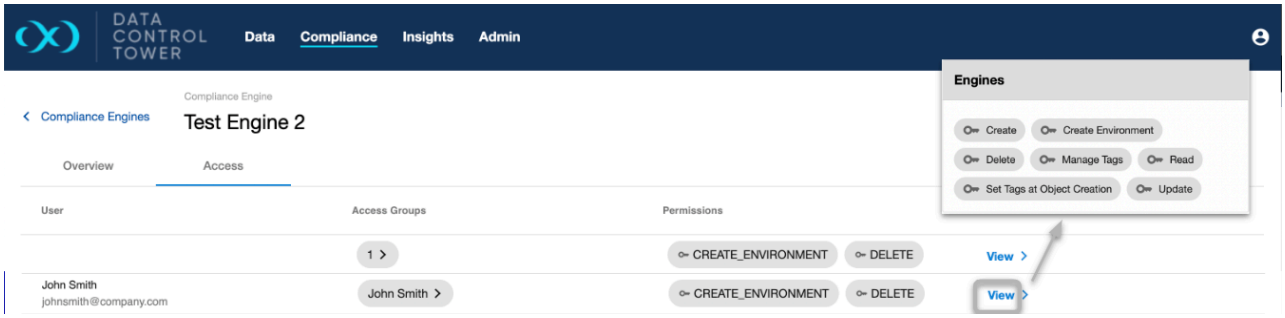
7.3.1 Engine overview

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.



7.3.2 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.

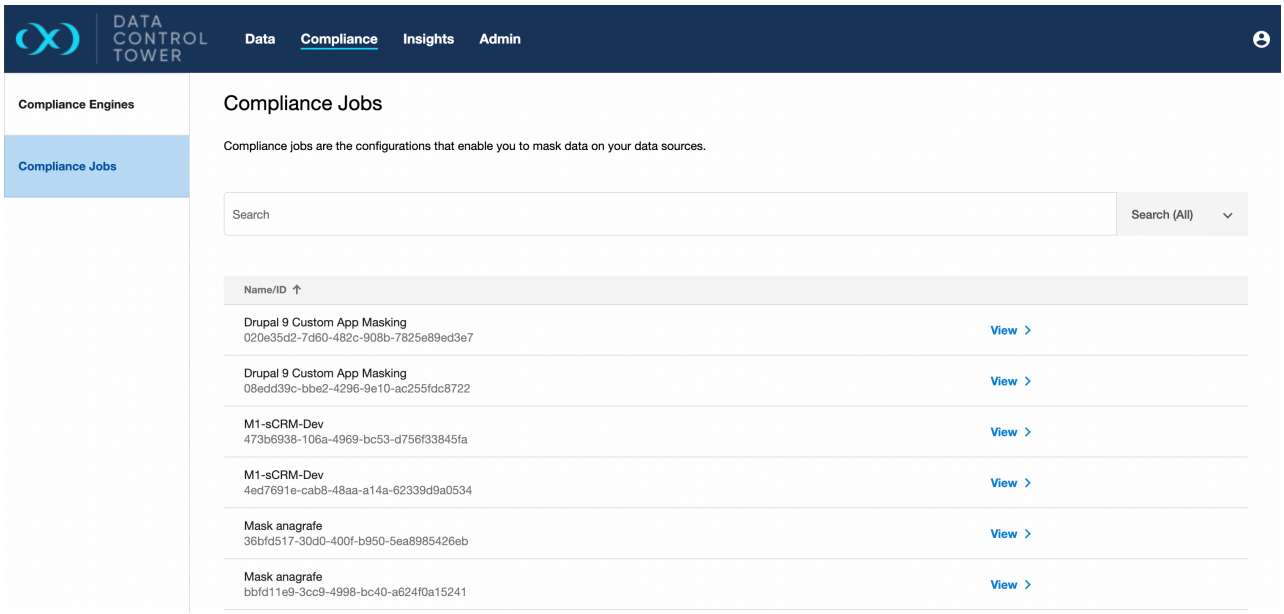


7.4 Compliance jobs

7.4.1 Job UI

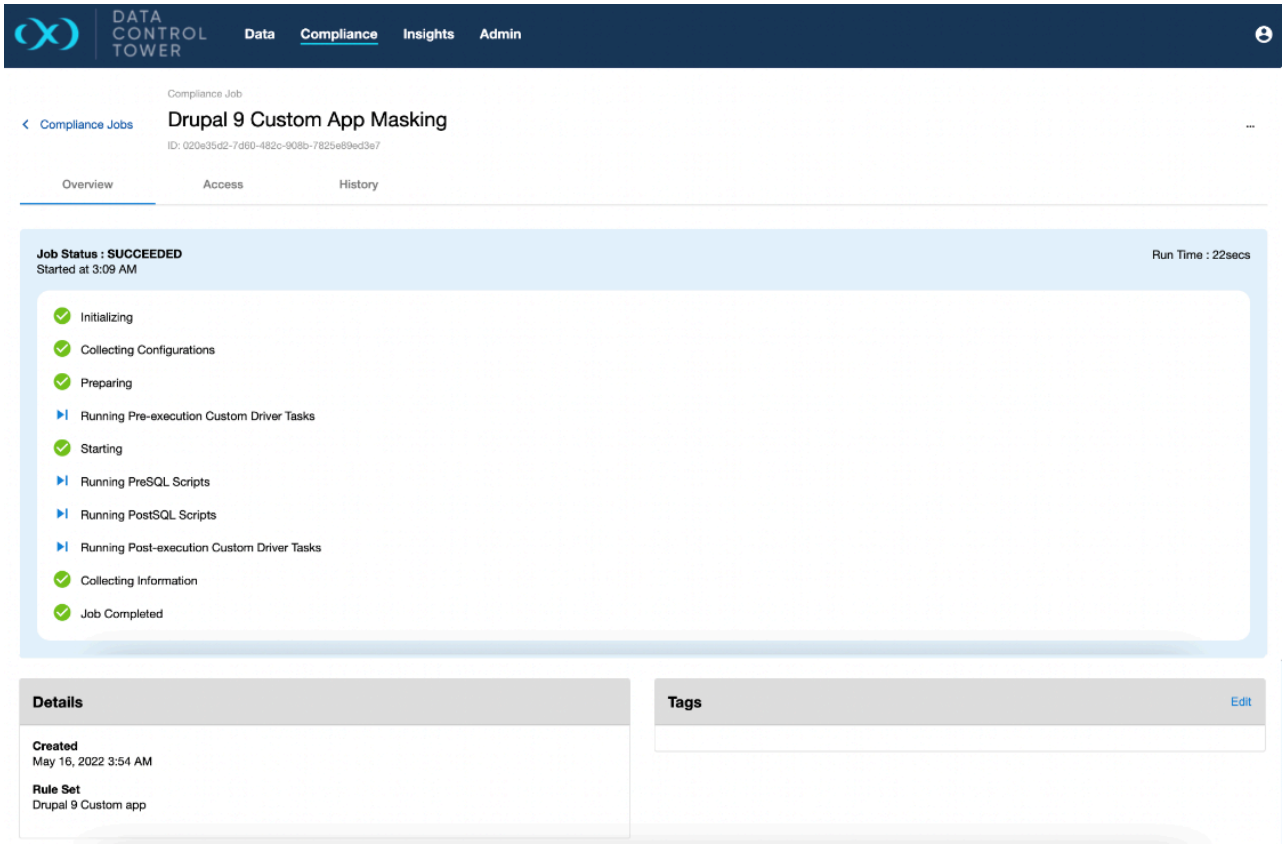
7.4.1.1 Global compliance jobs list

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.



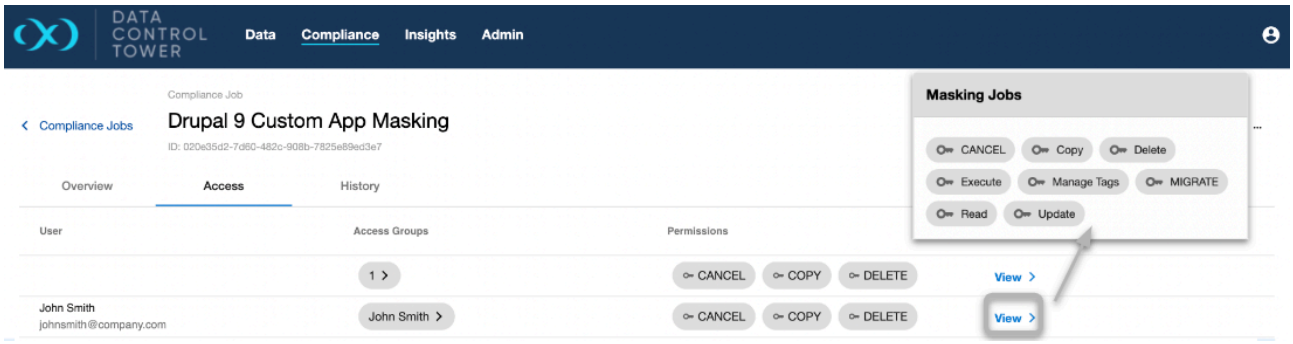
7.4.1.2 Compliance job overview

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.



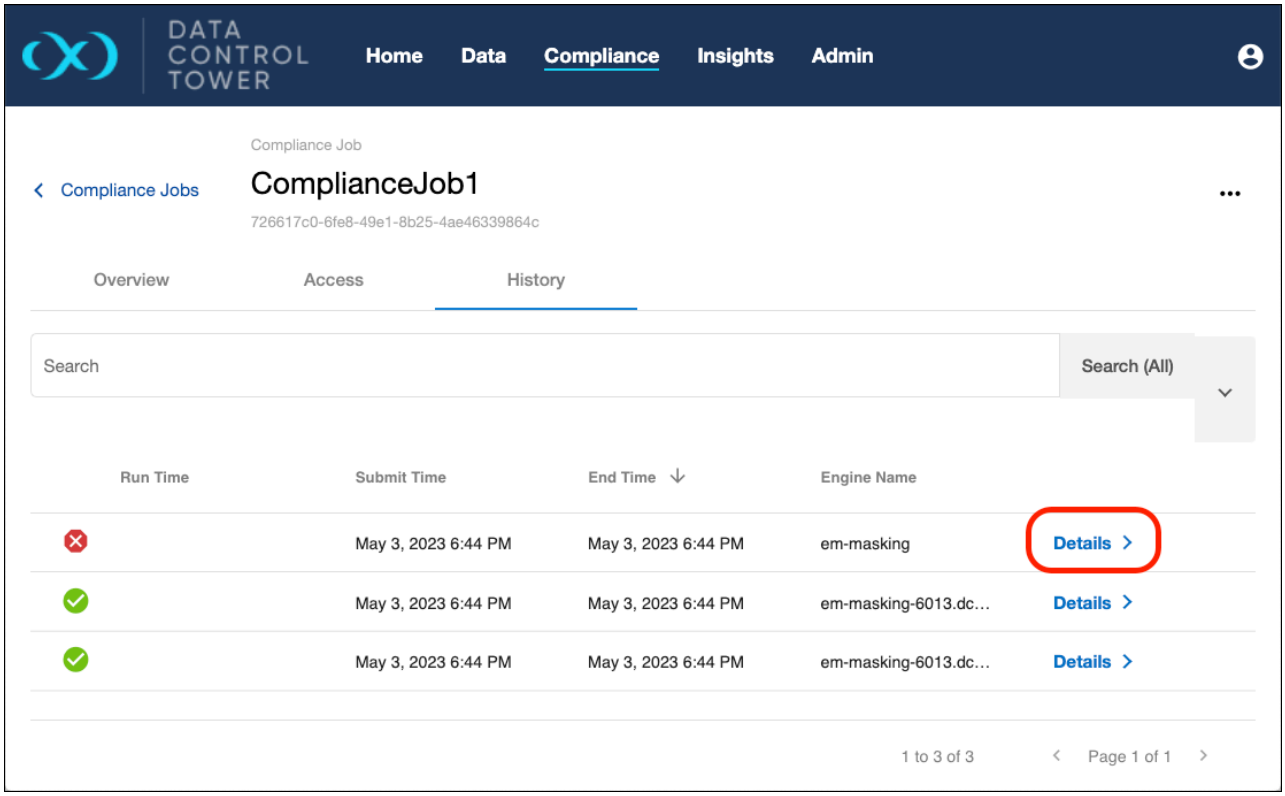
7.4.1.3 Access tab

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.



7.4.1.4 Execution history tab

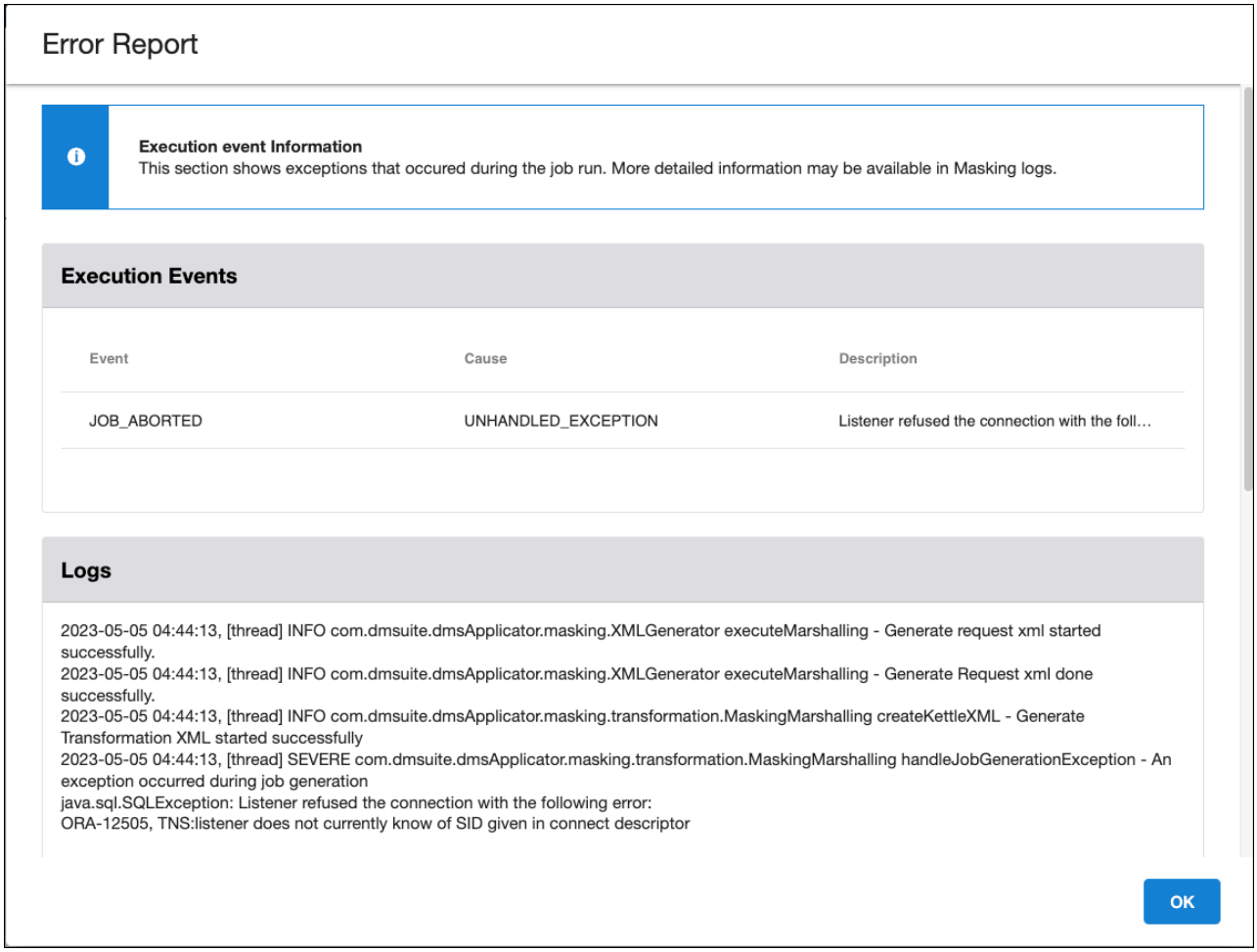
The compliance jobs page also includes job execution history. Execution information will be recorded and displayed on this tab, and includes the status (success, running, failed), run time, submit, and end timestamps, and engine on which it ran. Additionally, clicking the “details” button for an execution will display its report.



7.4.1.5 Execution details

The execution details view includes a list of execution events and the execution log. This is particularly useful when troubleshooting failed executions.

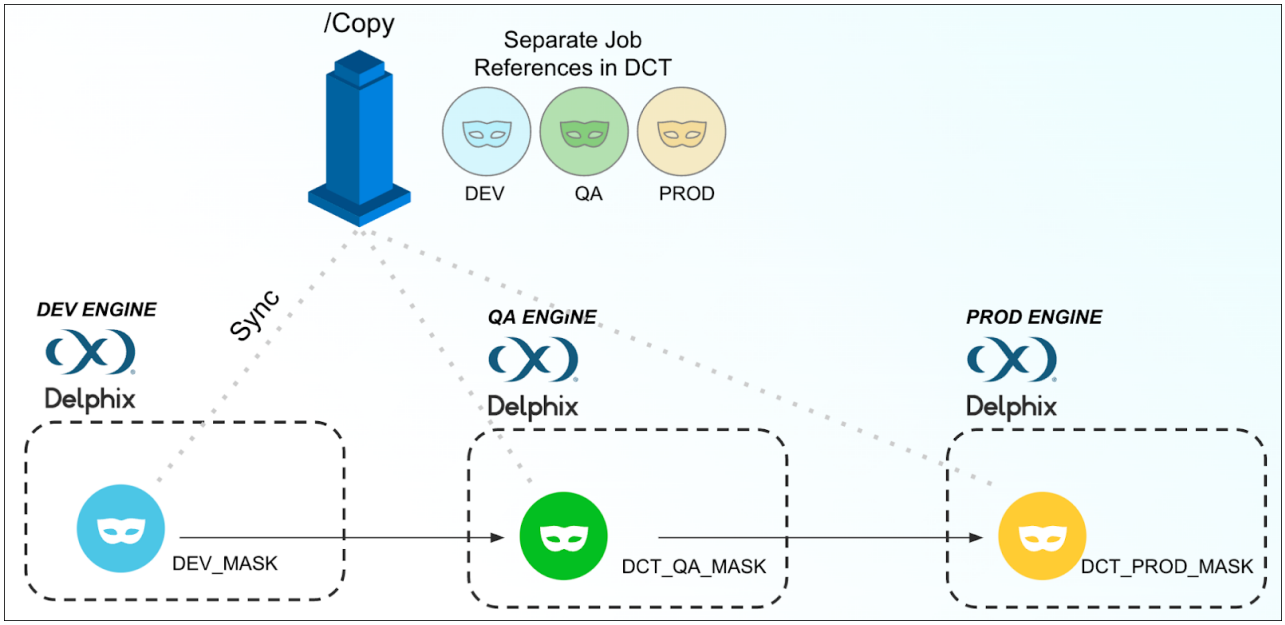
Details for successful executions may also have events and logs which include relevant information, such as warnings.



7.4.2 Copy job

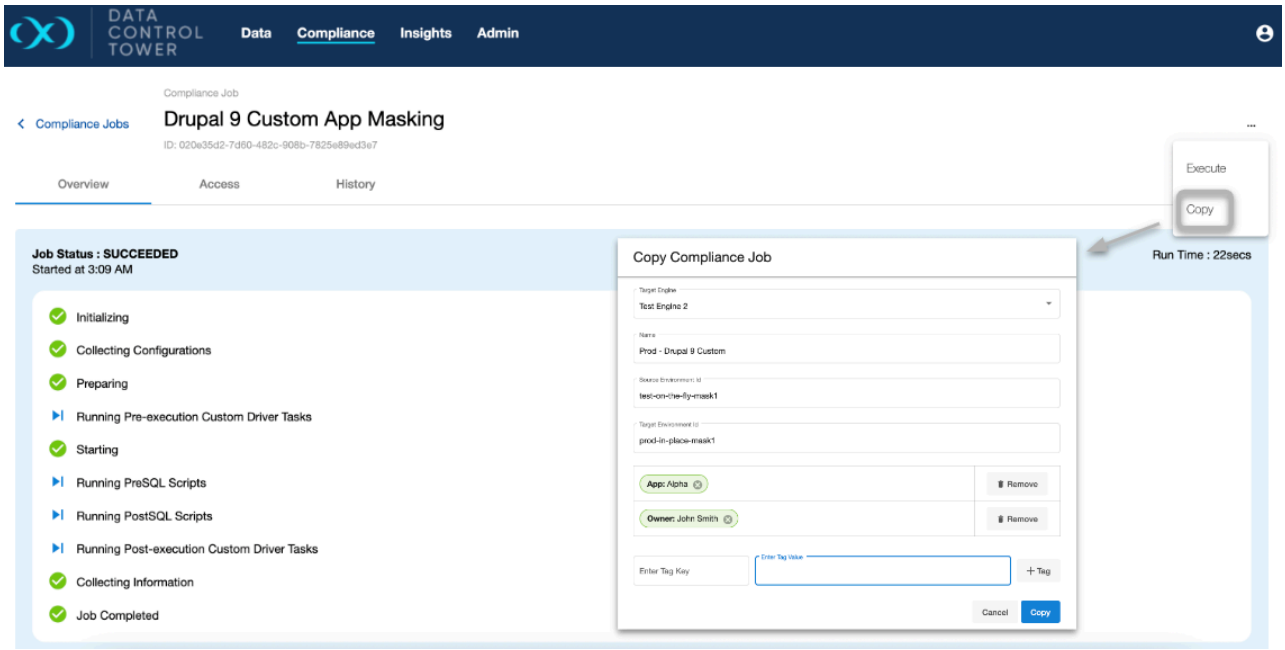
> ⚠ Compliance Engines limit any syncing operations while a profiling or masking job is running. When using DCT job **move**, **execute**, or **migrate** operations, please ensure that the target Compliance Engine is in an idle state. Future Compliance enhancements to DCT will remove this limitation.

The Masking Job **Copy operation** creates a duplicate of a job with a separate reference for that new copy. This operation supports SDLC workflows as DCT will maintain unique references for each instance of a masking job, enabling them to be managed independently.



7.4.2.1 User interface documentation

Job copy can be run via the DCT UI by accessing a compliance job's detailed view and selecting the ellipsis in the top right corner and clicking on "copy". This will open a window to select the target engine, the new name of the transferred job, source and target environment details, and relevant tags.



7.4.2.2 API documentation

For input, the user must specify the target engine along with the environment on the target engine that the job will be copied onto. The engine and environment pair is what uniquely identifies a copy of the Masking Job. Calling the COPY API against the same target engine and environment effectively serves as a re-sync and does not create a new DCT MaskingJob entity.

Example of **copying a MaskingJob to engine with ID 2 and environment named 'prod-env'**:

```
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"
}'
```

MaskingJob sync will not copy connector credentials to another engine. In order to make a copied job executable outside of DCT, the credentials must be set on the Connector itself. The connectors for a MaskingJob can be searched for, updated, and tested directly via DCT.

Example of **listing connectors for a MaskingJob**:

```
curl -X 'GET' \
  'https://<APPLIANCE_ADDRESS>/v3/masking-jobs/d53812ce-9186-485d-a388-44bc52087ead/
connectors' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>'
```

Example of **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"
}'
```

Example of **testing a connector**:

```
curl -X 'POST' \
  '<https://<APPLIANCE_ADDRESS>/v3/connectors/2-DATABASE-23/test'> \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
```

7.4.3 Execute job

> ⚠ 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.

The **Execute endpoint** creates a duplicate of a job while maintaining a single reference for both job instances. This operation supports Horizontal Scale workflows, as DCT will maintain a singular reference for all instances of a job across any number of connected Compliance engines.

As part of this endpoint, DCT will maintain all of these job instances in sync, so they can all be controlled from a single configuration point (it's recommended to dedicate a select engine or set of engines to the creation and updating of masking jobs and dependencies) and any changes are automatically propagated to the other job instances at the time of the next job execute operation. This enables users to identify a masking job on a configuration engine, copy it over to a dedicated compute engine (or set of engines), and run that job at a regular cadence through DCT. Whenever the job needs to be updated, the user simply updates the job on the configuration engine.

> Since all jobs connected via the job execute operation are under a single reference, every time a job is run, its run statistics will report back to DCT and will be recorded under that singular job reference.

Executing a MaskingJob requires only a reference to a target engine as input. DCT will take care of syncing the job to the target engine and executing it. DCT will create and manage the environment where the job is copied onto.

Example of **executing a MaskingJob on engine with ID 2**:

```
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 '{
  "engine_id": "2"
}'
```

This will return a DCT job that can be further polled for status updates. The job will only transition to the COMPLETED state when the entirety of the sync and execution has completed on the target engine.

When a MaskingJob is executed via DCT and the job is synced to the target engine, the default Connector is used for execution. Masking job sync never copies credentials, for security reasons. Since having credentials set on the target connector is required for execution, DCT enables this by allowing users to store connector credentials within DCT itself. A DCT MaskingJob now contains properties for the connector credentials. The expectation is that users will pre-store the credentials by using the UPDATE API on the MaskingJob. MaskingJob execution has a hard requirement that credentials be saved within a MaskingJob prior to allowing 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"
}'
```

Once a MaskingJob execution has been initiated, the EXECUTION APIs can be used to view and cancel running executions as well as search through execution history. Note that canceling an execution is a best-effort action that does not interrupt any of the job sync that may occur prior to the execution.

Example of **searching for executions of a particular MaskingJob**:

```
 'https://<APPLIANCE_ADDRESS>/v3/executions/search' \
  -H 'accept: application/json' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "filter_expression": "masking_job_id eq '\''d53812ce-9186-485d-
a388-44bc52087ead'\''"
}'
```

Example of **canceling an execution if and only if it is in the RUNNING state (denoted by the expected_status query parameter)**:

```
curl -X 'POST' \
  'https://<APPLIANCE_ADDRESS>/v3/executions/11397caa-6006-4eba-b575-ae3ad00c3762/
cancel' \
  -H 'accept: */*' \
  -H 'Authorization: <API_KEY>' \
  -H 'Content-Type: application/json' \
  -d '{
  "expected_status": "QUEUED"
}'
```

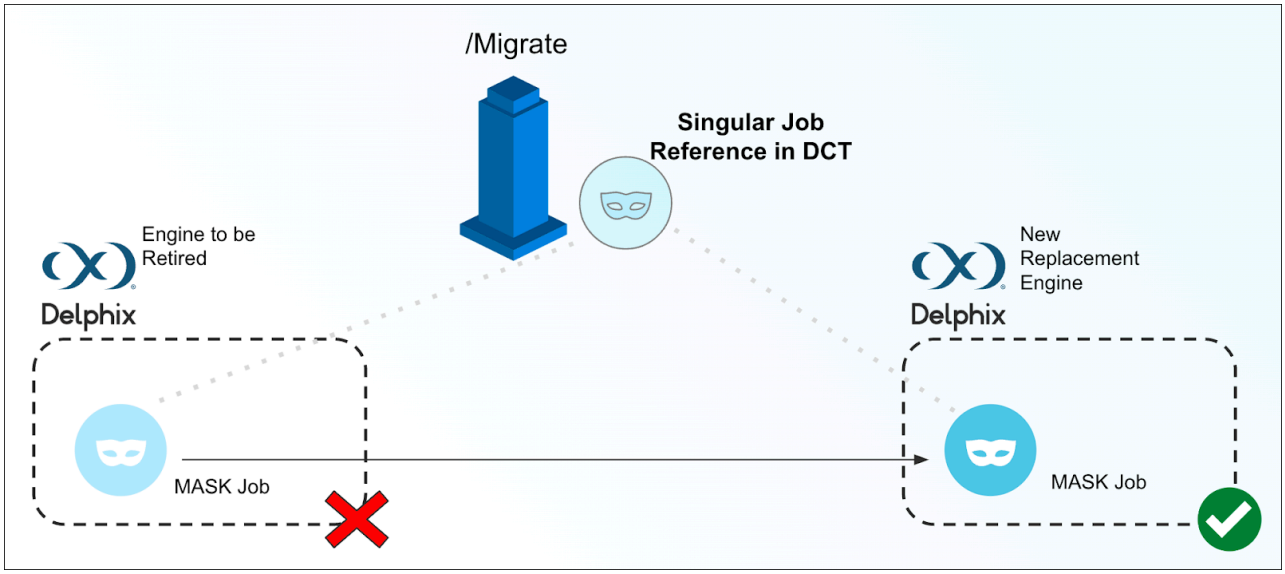
7.4.3.1 User interface

Execute a compliance job with the **Execute** action, available in the action menu on the top right corner of the job details page. This will open a window that lists Compliance engines with which a job needs to be executed. Once selected, click the "Execute" button to start the job on the selected engine. The screenshot below shows a selected engine.

7.4.4 Migrate job

> ⚠ 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.

The **Migrate endpoint** moves a job from one engine to another without any duplicates. This endpoint is useful for consolidating masking jobs (i.e. moving jobs to a fresh engine ahead of the original being retired or consolidating two development engines into a single one for administrative simplicity). This means that a job will continue to have only a single instance with no additional jobs being created. This job will maintain its same reference within DCT.



Example of **finding all MaskingJobs 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"
}'
```

7.4.5 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"
  }
}
```

8 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 187) 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[45] for more detail.

45 http://ecosystem.delphix.com

9 DCT concepts

9.1 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

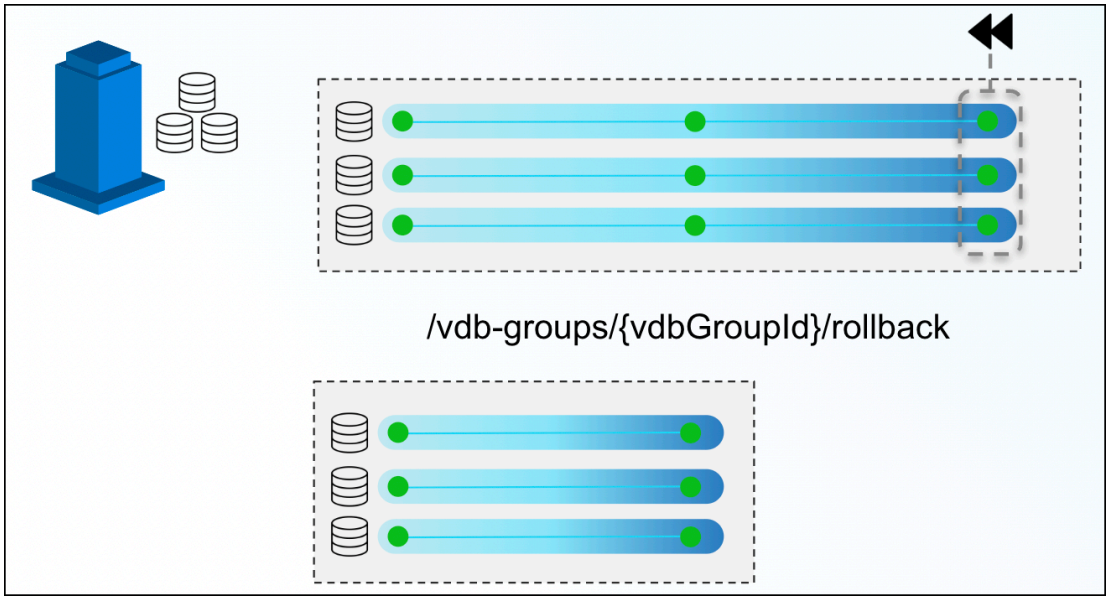
For **Infrastructure Connection Wizard**, only UNIX standalone environments can be added via the UI.

9.2 Concepts

9.2.1 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.

9.2.2 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

9.2.3 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

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.

9.2.4 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.

9.2.5 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.

9.2.6 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 187). 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'}
```

9.3 Nuances

9.3.1 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.

9.3.2 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.

9.3.3 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).

9.3.4 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).

9.3.5 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'}
```

9.3.6 Supported data sources/configurations

DCT is compatible with all Delphix-supported data sources and configurations.

9.3.7 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.

10 DCT Toolkit

10.1 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.

10.2 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.

10.3 Installation

Currently dct-toolkit is supported on Linux, Windows and macOS platforms. There are no special system requirements for running dct-toolkit.

Installing dct-toolkit is as simple as downloading the executable binaries from the downloads URL[46] and extracting it to the machine from where 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 via:

```
chmod 777 ./dct-toolkit
```

For **Windows** this can be done from cmd via:

```
CACLS files /e /p <username>:F
```

Where `<username>` is the user to whom the permission must be granted, `F` is the **Full Control** permission that needs to be granted.

For macOS currently the binary is not verified hence macOS will give warning that the binary is untrusted. To get around this issue for now you will need to open this binary and explicitly whitelist it. One easy way to achieve this is to press CTRL key and click on the binary file and then select Open. On the resulting warning message dialog box select Open. Now the binary is allowed to be executed from the terminal.

46 https://download.delphix.com/folder/4720/Delphix%20Product%20Releases/DCT/Toolkit%20(CLI)

10.4 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, the following command can be used:

```
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 configuration file will be: `<users home directory>/.dct-toolkit/dct-toolkit.properties`

`create_config` command takes in below mentioned options:

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 then the latest API version from the DCT instance.
`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 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 also be requested via:

```
./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 is 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

It's acceptable to configure non-sensitive properties in a properties file and sensitive properties (like `apiKey`) via an environment variable. 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 via the environment variable, dct-toolkit is ready for use.

10.5 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 command:

```
./dct-toolkit -h
```

This will print the list of all commands grouped by the appropriate entity names.

10.6 Examples

1. The list of available options for a particular command can be requested via:

```
./dct-toolkit <command_name> -h
```

2. There are currently a lot of 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, the following command can be used:

```
dct-toolkit provision_vdb_by_snapshot oracle source_data_id=<dsourceId> --
auto_select_repository unique_name=<uniqueName>
```

For provisioning a Sybase-ASE VDB, the following command can be used:

```
dct-toolkit provision_vdb_by_snapshot sybase source_data_id=<dsourceId> --
auto_select_repository --truncate_log_on_checkpoint
```

3. For getting the list of registered engines, the following command can be used:

```
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:

```
-A/--all-columns
```

Option:

```
dct-toolkit get_registered_engines -A
```

There is also an option to list only the selected columns that are useful to the user:

```
--columns/-c
```

Option:

```
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[47] expression to extract required objects from the JSON response. This can be requested with:

```
--jsonpath/-jp
```

Option:

```
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*.

4. 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).

```
dct-toolkit token_info --body=<JSON body to POST to DCT>
```

47 https://github.com/json-path/JsonPath

Help regarding the exact structure of JSON request and response body can be requested for all of the commands via:

```
-jh/--jsonhelp
```

Option:

```
dct-toolkit token_info -jh
```

5. dct-toolkit provides an easy to use and understand format for specifying tags. Example, this command to add two tags: `{key=purpose, value=testing}` and `{key=key1, value=value1}`:

```
dct-toolkit create_environment_tags environmentId=<envId> tags purpose=testing
key1=value1
```

6. 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 get completed and will report the end state of the job. There is an option to NOT wait for the completion of an asynchronous job. For example, to NOT wait, use the `--no-wait` option:

```
dct-toolkit refresh_environment environmentId=<envId> --no-wait
```

10.7 Logging

dct-toolkit has inbuilt logging support which will export all of the logs to a default location which is 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 **Setup** section above for more information.

11 Developer resources

11.1 API requests and reporting

11.1.1 Introduction

This article showcases example requests to the various data APIs supported by DCT.

DCT provides interactive API documentation that allows users to experiment with the APIs in their web browser. The interactive API documentation can be accessed by entering the hostname for DCT and the **/api** path into a browser's address bar. For example, if DCT is running on host `gateway.example.com`, then enter `https://gateway.example.com/api` into the browser's address bar.

To simplify development, Python and Go programming libraries are available. The **Python** bindings can be found on PyPi here[48]. The latest version can be installed with the following command:

```
pip install delphix-dct
```

The **Go** bindings can be found on go.dev here[49].

11.1.2 Engines

This section showcases some examples of querying the Engines endpoint for information about connected Delphix Virtualization Engines. These examples leverage the generated Python bindings:

```
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

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:

48 https://pypi.org/project/delphix-dct-api/
49 https://pkg.go.dev/github.com/delphix/dct-sdk-go

```
{'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'}]}
```


11.2 API references

(Open API documentation is only available to view online)