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.
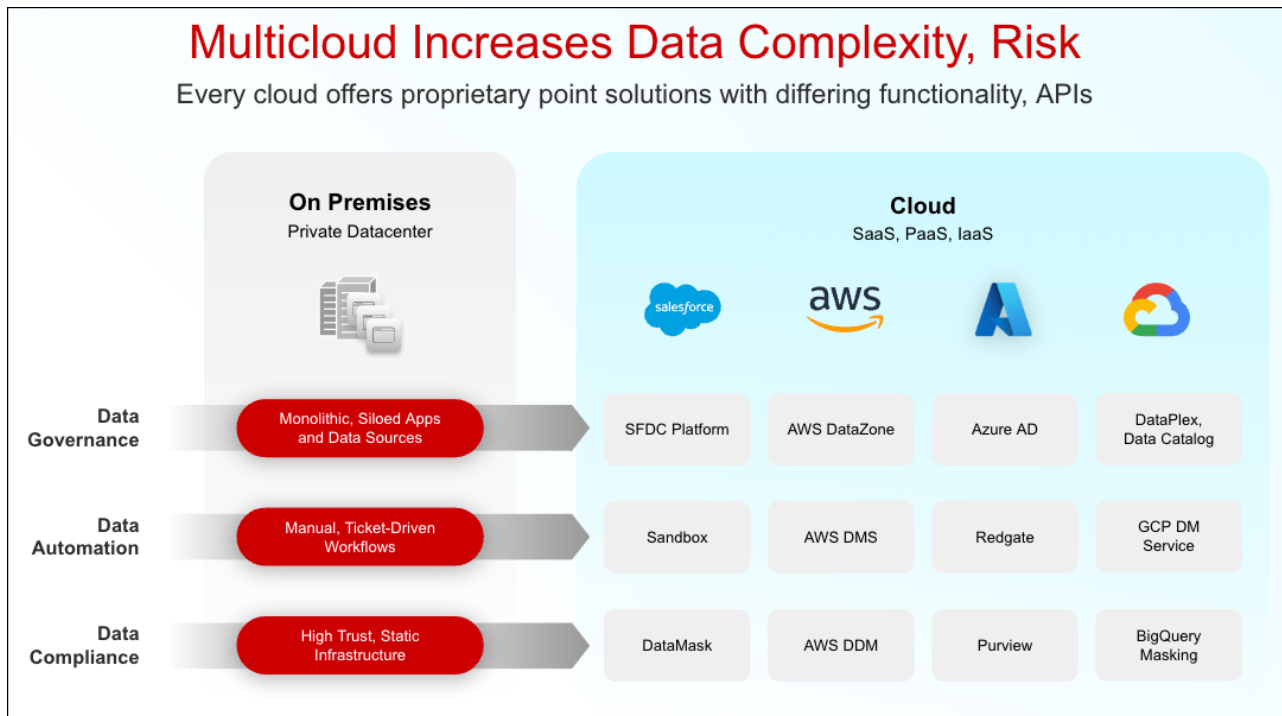
Data Control Tower (DCT) is an enabling Delphix platform that introduces a data mesh to unify data governance, automation, and compliance across all applications and cloud platforms.

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 New features

2.1 Overview

Data Control Tower (DCT) is a unified data management platform used to enable API-driven DevOps workflows with Continuous Data and Compliance, and centrally manage the entire data estate for all Delphix users. The core of DCT focuses on a centralized, container-based deployment architecture with a robust API library that enables automation and management across multiple Delphix Engines. The DCT container form factor allows users to self-host on their preferred public or private cloud environment. For more information, see Upgrade DCT.

2.2 New in DCT 2.2

2.2.1 Deployment

- Introducing Kubernetes and OpenShift support

2.2.2 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

2.2.3 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

2.3 DCT Features

- Automation APIs to enable automating compliant data within DevOps pipelines
- Reporting APIs to provide central reporting across many Delphix Engines
- Terraform Provider for Delphix support, for data infrastructure automation
- Engine versions 6.0.0.1 and above are supported for Continuous Data
- Connections to Engines over HTTPs (self-provided certificate authority when necessary)
- Username/password to authenticate with engines stored in DCT or provided via Password Vault
- HashiCorp Vault supported
- Audit logging via Nginx access log, self-configured Docker logging going to fluentd, splunk, etc.
- Native APIs keys supported for DCT authentication, OAuth2.0 provided via Nginx module
- OpenAPI 3.0 spec (self-generated client libraries with open-source tooling)

2.4 Supported APIs

API	Description
Engines	List engines connected to DCT.
Environments	List environments across Engines.
Datasets	List sources, dSources, and VDBs across Engines.
Snapshots	List Timeflow Snapshots for a dataset.
Bookmarks	Bookmark a point in time in a VDB to ease the process of restoring the VDB later to that point in time.
VDB Provisioning	Provision Oracle single instance, MSSQL, or Sybase ASE databases.
VDB Operations	Enable, Disable, Start, Stop, Delete, Refresh, Rewind VDBs.
VDB Groups	A new grouping mechanism used in conjunction with bookmarks that was previously supported by VDB Templates in the Self Service module.
Jobs	Monitor progress of asynchronous tasks.
Reporting	Query across multiple engines to list storage capability, VDB inventory, and dSource inventory.
Connectivity	Checks connectivity between an Engine and a remote host machine.

3 DCT concepts

3.1 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.

3.2 Concepts

3.2.1 Virtual Database (VDB) groups

Virtual Database (VDB) groups are a new concept to Delphix, that enables 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.

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.

> **Additional Information**
>
> Bookmarks can be generated from vdb-groups and can be shared with compatible vdb-groups (having the same underlying databases).
>
> 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.

3.2.2 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

3.2.3 DCT 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 that are 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
- Share (Refresh a VDB or VDB Group from a compatible sibling VDB or VDB Groups' bookmark) bookmarks with a compatible testing environment

> ⓘ Additional Information
> 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 6.0.13 and above.
> DCT Bookmarks, when created, initiates 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.
> Creating a bookmark for a point in time in the past, or for dSources, is not possible.

3.2.4 DCT 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 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.

3.2.5 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. The DCT 2.0 and above release supports 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).

A few 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.

3.2.6 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 40) 2.0. 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'}
```

3.3 Nuances

3.3.1 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.

3.3.2 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.

3.3.3 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).

3.3.4 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).

3.3.5 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'}
```

3.3.6 Supported data sources/configurations

DCT is currently compatible with the most common data sources and configurations supported by Delphix, which includes:

- Oracle Single Instance
- Oracle Clusters
- SQL Server Single Instance
- SQL Server Cluster (Availability Groups)
- ASE Single Instance
- ASE Clusters

3.3.7 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.

3.3.8 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: "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 (default start date is "when DCT was installed" and 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
```

3.4 Docker Compose

3.4.1 Installation and setup for Docker Compose

3.4.1.1 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	Memory	Storage	Port
Data Control Tower	4-Core	2GB	50GB	443

3.4.1.2   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[1].

This example uses a Docker installation[2] 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[3].

```
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
```

> **Notice**
>
> As of 26 April 2022, the latest version of Docker CLI (**20.10.15**[4]**+**) comes by default with Docker Compose V2. Therefore, separate installation for docker-compose would no longer be required. More information in this regard can be found in this Docker Compose V2[5] article.
>
> Docker Compose V2 is the supported and DCT recommended version of compose.

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[6] documentation for details.

---

1 https://docs.docker.com/engine/install/#server
2 https://docs.docker.com/engine/install/
3 https://docs.docker.com/compose/install/
4 https://github.com/docker/cli/releases/tag/v20.10.15
5 https://docs.docker.com/compose/#compose-v2-and-the-new-docker-compose-command
6 https://docs.docker.com/engine/install/linux-postinstall/

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

3.4.1.3   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[7]. 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:

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

3.4.1.4   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:6.0.0           "/sbin/tini -- /boot…"   7 seconds
ago    Up 4 seconds   0.0.0.0:443->8443/tcp   delphix-dct-proxy:3.0.0
a23f4fbe0220    delphix-dct-app:6.0.0             "java -jar /opt/delp…"   7 seconds
ago    Up 5 seconds                   delphix-dct-app:6.0.0
96ba8018fa03    delphix-dct-data-library:6.0.0    "/usr/bin/tini -- ./…"   7 seconds
ago    Up 5 seconds                   delphix-dct-data-library:6.0.0
8e5b1e671acc    delphix-dct-jobs:6.0.0            "/usr/bin/tini -- ./…"   7 seconds
ago    Up 5 seconds                   delphix-dct-jobs:6.0.0
96049058f025    delphix-dct-data-bookmarks:6.0.0  "/usr/bin/tini -- ./…"   7 seconds
ago    Up 5 seconds                   delphix-dct-data-bookmarks:6.0.0
20d1782cb3bb    delphix-dct-ui:6.0.0              "node ./index.js"       7 seconds
ago    Up 5 seconds                   delphix-dct-ui:6.0.0
4fae43c79e8d    delphix-dct-virtualization:6.0.0  "/usr/bin/tini -- ./…"   7 seconds
ago    Up 5 seconds                   delphix-dct-virtualization:6.0.0
83d7d661d8a0    delphix-dct-graphql:6.0.0         "/bin/sh -c 'BASE_UR…"   7 seconds
ago    Up 6 seconds                   delphix-dct-graphql:6.0.0
3dded474e28b    delphix-dct-postgres:6.0.0        "docker-entrypoint.s…"   7 seconds
ago    Up 6 seconds    5432/tcp       delphix-dct-postgres:6.0.0
```

---

7 https://download.delphix.com/folder

3.4.2 Custom configuration

3.4.2.1 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.

3.4.2.2 Bind mounts

Configuration of DCT is achieved through a combination of API calls and the use of Docker bind mounts[8]. 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 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
```

---

[8] https://docs.docker.com/storage/bind-mounts/

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.

```
$ docker ps

CONTAINER ID   IMAGE                            COMMAND            CREATED
STATUS         PORTS                                     NAMES
ac343412492a   delphix-dct-proxy:3.0.0   "/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.

3.4.3 Docker logs

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

DCT leverages the Docker logging[9] 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[10] on how to configure them. These changes will need to be made to the docker-compose.yaml file. This link[11] 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"
```

---

9 https://docs.docker.com/config/containers/logging/
10 https://docs.docker.com/config/containers/logging/configure/
11 https://docs.docker.com/compose/compose-file/compose-file-v3/#logging

3.4.4 Migration topics

3.4.4.1 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:

```
~$ 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:

3.4.4.2 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
```

3.4.5 Admin topics for Docker Compose

3.4.5.1 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[12] 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
```

3.4.5.2 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.).

---

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

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
> ```

3.4.5.3 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
```

3.5 Kubernetes

3.5.1 Installation and setup for Kubernetes

3.5.1.1 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.

3.5.1.2 Installation requirements (Kubernetes)

DCT requires a running Kubernetes cluster to run, 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	

Requirement	DCT Recommended Version	Comments
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[13].
kubectl	1.25.0 or above	To install kubectl follow the instructions at https://kubernetes.io/docs/tasks/tools/ .

> 📋 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.

3.5.1.3 Installing DCT

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

```
curl -XGET https://dlpx-helm-dct.s3.amazonaws.com/delphix-dct-7.0.0.tgz -o delphix-dct-7.0.0.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:

```
tar -xvf delphix-dct-7.0.0.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[14] page and login with your customer login credentials. Once logged in, select the **DCT Helm Repository** link and accept the Terms and

---

13 https://dlpx-helm-dct.s3.amazonaws.com/
14 https://download.delphix.com/folder/1144/Delphix%20Product%20Releases/DCT

Conditions. Once accepted, login credentials will be presented. Note them down and edit the `imageCredentials.username` and `imageCredentials.password` properties in the **values.yaml** file as shown below:

```
# Credentials to fetch Docker images from Delphix internal repository
    imageCredentials:
# Username to login to docker registry
    username: <username>
# Password to login to docker registry
    password: <password>

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

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:

```
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-7.0.0     7.0.0
```

> 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.

> 🗒 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).

3.5.2 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.

3.5.3 Admin topics for Kubernetes

3.5.3.1 Deployment upgrade for Kubernetes

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

Create a new folder called **dct-[version]**, where **[version]** is the latest version to which the platform is being upgraded (i.e. if on 5.0.2, it would be 6.0.0).

```
$mkdir dct-[version]
```

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

```
$cd dct-[version]
$curl -XGET https://dlpx-helm-dct.s3.amazonaws.com/delphix-dct-[version].tgz -o
delphix-dct-[version].tgz
```

> 🗒 This command will download a file named **delphix-dct-[version].tgz** in the folder dct-[version].

The downloaded file is then extracted using the following command:

```
$tar -xvf delphix-dct-[version].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-[version] 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[15]. 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[16]):

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

The last step is to run the helm upgrade command:

---

15 https://download.delphix.com/
16 https://download.delphix.com/

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

3.5.3.2 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.

3.6 OpenShift

3.6.1 Installation and setup for OpenShift

3.6.1.1 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

3.6.1.2 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

Requirement	DCT Recommended Version	Comments
		HELM repository from where DCT charts can be downloaded. The HELM repository URL is https://dlpx-helm-dct.s3.amazonaws.com[17].
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 .

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.

3.6.1.3 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:

---

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

```
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:

```
curl -XGET https://dlpx-helm-dct.s3.amazonaws.com/delphix-dct-7.0.0.tgz -o delphix-
dct-7.0.0.tgztar -xvf delphix-dct-7.0.0.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
```
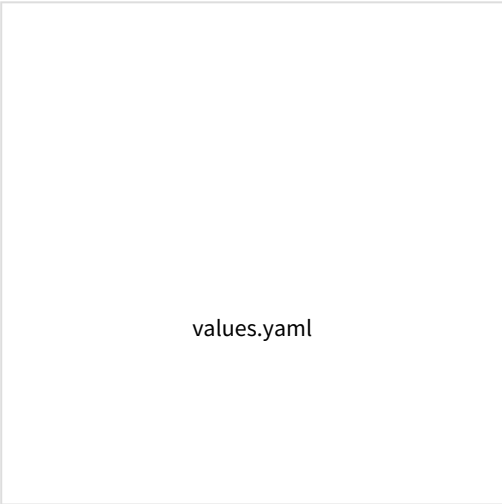
Copy the first value from the `openshift.io/sa.scc.supplemental-groups` line, before the slash (e.g. 1001000000).

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.

values.yaml

Deploy DCT

Run the following command to deploy the DCT chart:

```
helm install -f <path to edited values.yaml> dct-services apigw-repo/delphix-dct -
version=7.0.0
```

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
```

3.6.1.4 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:

3.6.2 OpenShift authentication

3.6.2.1 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.

3.6.2.2 Enable OAuth2 authentication

By default APIKey authentication will be enabled and when DCT starts it will generate a new API key(see page 48) 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
```

3.6.3 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.

3.6.4 Admin topics for OpenShift

3.6.4.1 Deployment upgrade for OpenShift

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

Create a new folder called **dct-[version]**, where **[version]** is the latest version to which the platform is being upgraded (i.e. if on 5.0.2, it would be 6.0.0).

```
$mkdir dct-[version]
```

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

```
$cd dct-[version]
$curl -XGET https://dlpx-helm-dct.s3.amazonaws.com/delphix-dct-[version].tgz -o
delphix-dct-[version].tgz
```

> This command will download a file named **delphix-dct-[version].tgz** in the folder dct-[version].

The downloaded file is then extracted using the following command:

```
$tar -xvf delphix-dct-[version].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-[version] 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[18]. 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[19]):

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

---

18 https://download.delphix.com/
19 https://download.delphix.com/

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

The last step is to run the helm upgrade command:

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

3.6.4.2 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.

3.7 Developer resources

3.7.1 API requests and reporting

3.7.1.1 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[20], 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[21]. The latest version can be installed with the following command:

```
pip install delphix-dct
```

The **Go** bindings can be found on go.dev here[22].

---

20 http://gateway.example.com/
21 https://pypi.org/project/delphix-dct-api/
22 https://pkg.go.dev/github.com/delphix/dct-sdk-go

3.7.1.2 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:

```python
import delphix.api.gateway
import delphix.api.gateway.configuration
import delphix.api.gateway.api.management_api
cfg = delphix.api.gateway.configuration.Configuration()
cfg.host = "https://localhost/v2"

# For example purposes

cfg.verify_ssl = False

# Replace the string with your own API key

cfg.api_key['ApiKeyAuth'] = 'apk 3.tEd4DXFce'
api_client = delphix.api.gateway.ApiClient(configuration=cfg)
engines_api =  delphix.api.gateway.api.management_api.ManagementApi(api_client)
print(engines_api.get_registered_engines())
```

The result should appear similar to the following:

```
{'items': [{'connection_status': 'ONLINE',
           'cpu_core_count': 2,
           'data_storage_capacity': 23404216320,
           'data_storage_used': 11589626880,
           'hostname': 'avm.delphix.com',
           'id': 1,
           'insecure_ssl': True,
           'memory_size': 8589934592,
           'name': 'vmname',
           'password': '******',
           'status': 'CREATED',
           'tags': [],
           'type': 'UNSET',
           'unsafe_ssl_hostname_check': False,
           'username': 'admin',
           'uuid': 'ec2fbfea-928b-07f8-94c4-29fea614624f',
           'version': '6.1.0.0'}]}
```

3.7.2 API references

To access the API list for DCT version 2.2.0, click the link below and the .html file with the API content will download.

DCT v2.0.0 API.html

4 Supported versions

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

5 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.
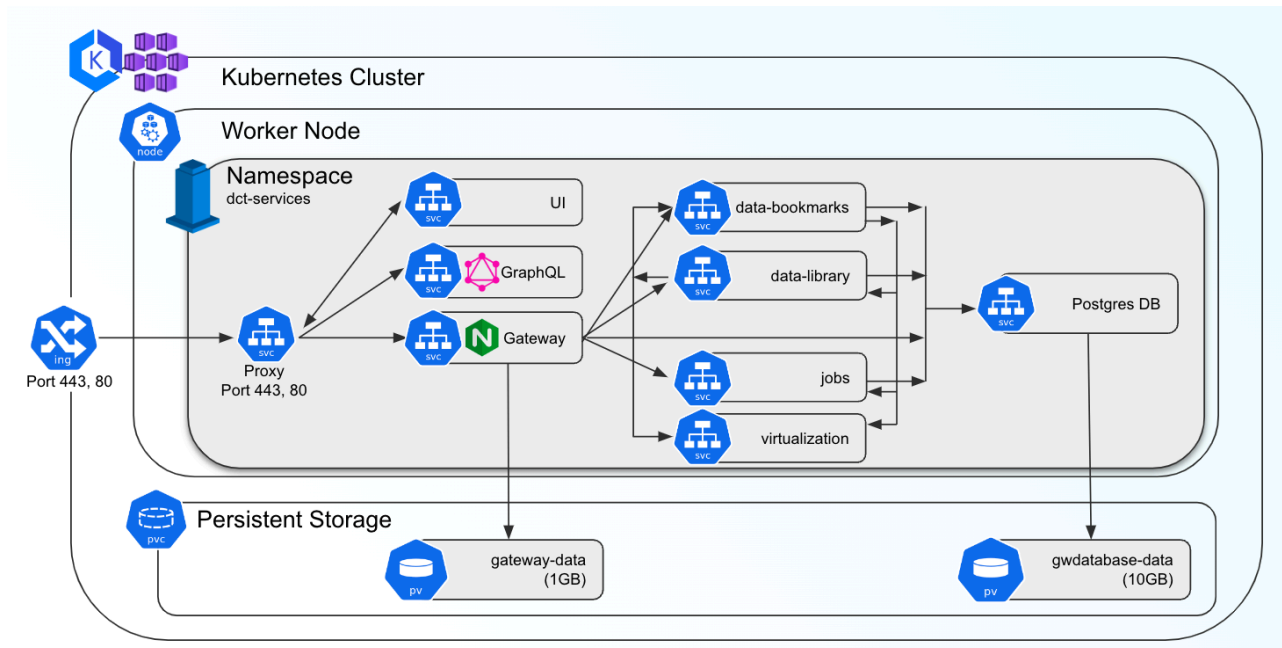
DCT is multi-cloud enabled, which means that a single DCT instance can be deployed to orchestrate (via HTTPS) Continuous Data and Continuous Compliance workloads with Delphix engines located in other networks. Alternatively, DCT can be localized to engines located within a network. DCT is a lightweight 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.
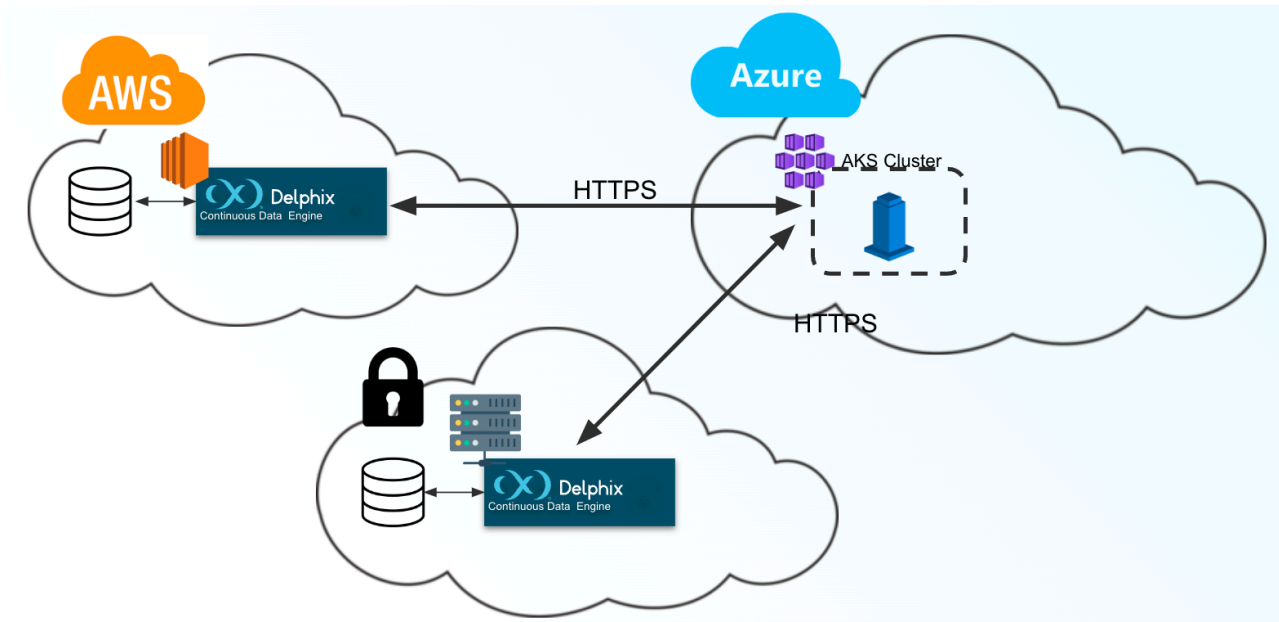
This section will explain all of the required steps to deploy DCT on your container platform of choice.

5.1 Engines: connecting/authenticating

5.1.1 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.

5.1.2 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.

5.1.3 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.

5.1.4 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.

5.1.4.1 Vault authentication and registration

First, DCT needs to be able to authenticate with the vault. DCT supports the Token[23], AppRole[24], and TLS Certificates[25] authentication methods. This is done by passing a command to the HashiCorp CLI[26]. 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"
   }
```

5.1.4.2 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>"
```

---

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

```
            "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.

5.1.4.3 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"
    }
}
```

5.1.5 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.

5.1.5.1 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.

5.2 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[27] 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).

---

[27] https://openresty.org/en/

5.2.1 API keys

5.2.1.1 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[28] 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 64) 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.

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"
}
```

---

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

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>'
```

5.2.2 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[29] with type **Bearer**. A cURL example using the access token retrieved above would appear as:

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

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

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

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

---

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

```
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".

5.2.2.1 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).

5.2.2.2 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.

5.2.2.3 Default password policy

By default, DCT does not enforce any password policy.

5.2.2.4 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.

5.2.2.5 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}'
```

5.2.3 LDAP/Active Directory

5.2.3.1 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.

Property Name	Description
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.
[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[30].

---

[30] mailto:john@mycompany.co

Property Name	Description
[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.

Edit LDAP Settings

☑ Enabled

☑ Auto-create Users

Hostname

activedirectory.acme.com

Port

636

Domains

\+

MSAD Domain Name

acme.com

Username Pattern

Search Base

CN=Users,DC=acme,DC=com

Group Attribute

department

Email Attribute

mail   \+   🗑

First Name Attribute

givenName

Last Name Attribute

sn

Object Class Attribute

person

Search Attribute

sAMAccountName

Cancel   **Save**

The following example requests enable LDAP authentication over SSL with an Active Directory server at address **activedirectory.company.co**[31] using the **us.company.co**[32] domain:

---

31 http://activedirectory.company.co
32 http://us.company.co

```
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.

5.2.4 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.

5.2.4.1 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.

5.2.4.2 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-meta-xml-blob>",
}'
```

> The IDP metadata must be JSON escaped. On a terminal with `./jq`[33] installed, this can be achieved with the following command: `jq --slurp --raw-input <<< 'xml-metadata-here'`

5.2.4.3 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.

5.2.4.4 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.

---

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

5.3 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**

5.3.1 Active Directory example

The following requests enable LDAP authentication over SSL with an Active Directory server at address **activedirectory.company.co**[34], using the **us.company.co**[35] 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"
 }]
}'
```

---

34 http://activedirectory.company.co
35 http://us.company.co

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[36] 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.

5.3.2 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="id9792398316760382115718051 6" IssueInstant="2022-12-01T10:07:12.
856Z" Version="2.0"
    xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
```

---

[36] mailto:john@mycompany.co

```
    <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"
        }
    ]
}
```

5.4 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).

6 Authentication

6.1 Introduction

DCT uses Nginx [/OpenResty](#)[37] 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 two supported methods for authentication; **API keys** and **OpenID Connect**. 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.

6.2 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](#)[38] 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](#)[39] 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.

6.2.1 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. T his 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:
```

---

[37] https://openresty.org/en
[38] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization
[39] https://portal.document360.io/#replace-https-certificate-for-apigw

```
    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` .

6.2.2 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 *Api Client* entity and setting the generate_api_key. The " name" attribute should be the desired name to uniquely identify the user of this key.

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

A response should be received similar to the lines below:

```
{
    "api_key_id": 5,
    "token": "5.vCfC0MnpySYZLshuxap2aZ7xqBKAnQvV7hFnobe7xuNlHS9AF2NQnV9XXw4UyET6"
}
```

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
5.vCfC0MnpySYZLshuxap2aZ7xqBKAnQvV7hFnobe7xuNlHS9AF2NQnV9XXw4UyET6'
```

The `id` referenced above is the numeric id of the API client. It is the integer before the period in the token. For example, the id of 1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0flaWajuKcBT3 is 1.

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

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

6.3 OAuth 2.0

6.3.1 Enable OpenID connect

DCT supports OAuth2 authentication via the OpenID Connect Discovery[40] specification. To add support for OAuth 2.0, first configure Nginx to communicate with your OAuth2 server: a bind mount[41] must be used . The file needs to be named **default.conf** and appear at `config/nginx/conf.d/default.conf` related to the root of the bind mounted directory. See the bind mount section about how to use docker cp to begin with the default version of the file.

There are three important sections of the file that need to be updated. First, the **open_id_connect_enabled** variable must be set to true and API keys need to be disabled like so:

```
local api_keys_enabled = false
local open_id_connect_enabled = true
```

The other two important configuration options are the discovery URL of the OAuth2 server and the specific attribute names of the JWT, to provide a unique ID and name for the user.

```
-- OpenID Connect implementation
if open_id_connect_enabled then
  local opts = {
    -- Replace the discovery URL with the discovery endpoint of your own OAuth2
server.
    discovery = "https://delphix.okta.com/oauth2/default/.well-known/oauth-
authorization-server",
```

---

40 https://openid.net/specs/openid-connect-discovery-1_0.html
41 https://portal.document360.io/dct-2-0-0/docs/dct-2-0-0-0-custom-configuration#bind-mounts

```
        ssl_verify = "yes",
        accept_unsupported_alg = false,
        accept_none_alg = false,
        redirect_uri = "",
    }

    local jwt, err, token = require("resty.openidc").bearer_jwt_verify(opts)
    if err then
        ngx.header["X-jwt-error"] = err
        ngx.status = 401
        ngx.log(ngx.ERR, "Invalid token: " .. err)
        ngx.exit(ngx.HTTP_UNAUTHORIZED)
        return
    end

    -- Replace "sub" with the attribute which is meant to be used
    -- as client_id or client_name in the JWT.
    ngx.var.client_id = jwt.sub
    ngx.var.client_name = jwt.sub

end
```

Once requests are authenticated, they must be matched with an existing API client in DCT. To do so, one of the claims of the JWT (Json Web Token) must correspond to the *client_id* of an API client. For instance, imagine someone is using a JWT with a *sub* claim with value *abc123* and the configuration above, which extracts the *sub* claim out of the JWT and sets it as *client_id* . We can create a corresponding admin API client with the following request:

```
curl -k --location --request POST 'https://<hostname>/v2/management/api-clients' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: apk
1.0p9PMkZO4Hgy0ezwjhX0Fi4lEKrD4pflejgqjd0pfKtywlSWR9G0fIaWajuKcBT3' \
--data-raw '{
    "name": "oauth2-test-api-client",
    "api_client_id": "abc123",
    "generate_api_key": false
}'
```

Set the *name* to a logical name corresponding to the application or person using the JWT. *is_admin* denotes whether this API client has admin access to DCT. *api_client_id* must be set to the exact string value found in the JWT claim extracted in the Nginx config above. *generate_api_key* is disabled in this example as the API client will use exclusively OAuth2 (JWT) to authenticate and not API keys.

6.4 Replace HTTPS Certificate for APIGW

By default to enable HTTPS, DCT creates a unique self-signed certificate when starting up for the first time. To replace this certificate and private key, use a [bind mount](#)[42]. The configuration needs to be bind mounted to a `/etc/config/nginx/ssl` directory inside the container. The default location for the configuration is `/etc/config/nginx/ssl/ssl.conf` and therefore should be placed in `nginx/ssl/ssl.conf` - inside the configuration directory on the host. The default configuration can be retrieved by using [docker cp](#)[43] to copy `/etc/config/nginx/ssl/ssl.conf` out of the proxy container and used as a starting point.

To replace the certificate and corresponding private key used by Nginx, place the updated certificate at `nginx/ssl/nginx.crt` and the private key at `nginx/ssl/nginx.key` inside the bind mounted directory on the host. Doing this could require the assistance of someone from 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. IT should be able to determine the correct certificate authority (CA) to sign this 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 include the FQDN as a Subject Alternative Name (SAN). The supported versions of TLS can be changed by altering the ssl_protocols line and the list of ciphers by altering the ssl_ciphers line. After this is done, restart DCT application with docker-compose.

---

42 https://portal.document360.io/dct-2-0-0/docs/dct-2-0-0-0-custom-configuration#bind-mounts
43 https://docs.docker.com/engine/reference/commandline/cp

7 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 drive a scalable approach to integrating Delphix into any custom script or automation toolchain.

Aside from the comprehensive API layer (see API references(see page 40) 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 <u>Delphix Ecosystem Hub</u>[44] for more detail.

---

[44] https://ecosystem.delphix.com/