



DataHub

Last updated: 03/03/2026

This content applies to the latest CD version of Cumulocity.

Specifications contained herein are subject to change and these changes will be reported in subsequent versions.

Copyright © 2026 Cumulocity GmbH.

The name Cumulocity GmbH and all Cumulocity GmbH product names are either trademarks or registered trademarks of Cumulocity GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

This software may include portions of third-party products. Third-party terms are set out in a 3rd-party-licenses file linked to or included with each installation package.

Table of Contents

Table of Contents	3
CUMULOCITY DATAHUB OVERVIEW	5
DOCUMENTATION OVERVIEW	5
CUMULOCITY DATAHUB AT A GLANCE	5
DESIGN OF OFFLOADING PIPELINES	6
GETTING STARTED WITH CUMULOCITY DATAHUB	8
ACCESSING AND LOGGING INTO CUMULOCITY DATAHUB	8
HOW TO LOG INTO CUMULOCITY DATAHUB	8
OVERVIEW OF UI FEATURES	8
HOME	9
SETTING UP CUMULOCITY DATAHUB	10
PREREQUISITES	10
DEFINING CUMULOCITY DATAHUB PERMISSIONS AND ROLES	10
CUMULOCITY DATAHUB ROLES AND PERMISSIONS	11
ASSIGNMENT OF CUMULOCITY DATAHUB ROLES AND PERMISSIONS	12
SETTING UP THE INITIAL CONFIGURATION	12
DEFINING THE INITIAL CONFIGURATION	12
EDITING SETTINGS	15
DELETING SETTINGS	15
SETTING UP DREMIO USERS	15
OVERVIEW OF DREMIO USERS	16
PROPERTIES OF A DREMIO USER	16
ADDING A DREMIO USER	17
EDITING A DREMIO USER	17
DELETING A DREMIO USER	17
WORKING WITH CUMULOCITY DATAHUB	18
OVERVIEW	18
BASIC FUNCTIONALITY	18
OFFLOADING OVERVIEW	18
ALIGNING DATA MODELING AND OFFLOADING	19
MAPPING DOCUMENT DATA TO RELATIONAL DATA	19
MANAGING MIXED TYPES	20
MAPPING MEASUREMENT FRAGMENTS TO RELATIONAL DATA	20
DEALING WITH CASE-SENSITIVITY	20
GUIDELINES	21
LIMITATIONS	22
CONFIGURING OFFLOADING JOBS	22
DEFINING AN OFFLOADING CONFIGURATION	22
OFFLOADING CUMULOCITY BASE COLLECTIONS	29
OFFLOADING THE BASE COLLECTIONS	29
MANAGING OFFLOADING JOBS	35
SCHEDULING AN OFFLOADING JOB	35
MANAGING AN OFFLOADING PIPELINE	36
IMPORTING/EXPORTING OFFLOADING CONFIGURATIONS	36
MONITORING OFFLOADING JOBS	37
HISTORY PER OFFLOADING PIPELINE	37
DETAILS OF OFFLOADING JOB	38
MONITORING COMPACTION JOBS	39
STATUS OF ALL COMPACTION JOBS	40
HISTORY OF COMPACTIONS PER OFFLOADING PIPELINE	40
DETAILS OF COMPACTION JOB	40
QUERYING OFFLOADED CUMULOCITY DATA	42
OVERVIEW	42
ACCESS TO DATA LAKE CONTENTS	42

GETTING DATA LAKE SCHEMA INFORMATION	43
USING THE DREMIO UI	43
CONNECTING VIA JDBC	43
CONNECTING VIA ODBC	43
CONNECTING VIA DREMIO REST API	43
CONNECTING VIA CUMULOCITY DATAHUB REST API	43
CONNECTING OTHER CLIENTS	43
REFINING OFFLOADED CUMULOCITY DATA	44
ACCESSING AND LOGGING INTO DREMIO	44
SOURCES AND SPACES	44
CREATING VIEWS	45
JOINING TABLES/VIEWS	46
CUMULOCITY DATAHUB BEST PRACTICES	46
NAMING POLICIES	46
CAREFUL DEFINITION OF ADDITIONAL COLUMNS AND FILTER PREDICATE	46
DECOMPOSITION OF COMPLEX ADDITIONAL COLUMNS	46
EXAMPLES FOR ADDITIONAL FILTER PREDICATES	47
QUERYING ADDITIONAL DATA WITH CUMULOCITY DATAHUB	48
MODIFYING DATA IN THE DATA LAKE	48
OPERATING CUMULOCITY DATAHUB	50
CHECKING SYSTEM INFORMATION	50
TRACKING USAGE STATISTICS	50
VIEWING AUDIT LOGS	51
QUERY LOG	51
SYSTEM LOG	52
ENDPOINTS FOR MONITORING	52
ETL PIPELINE HEALTH	52
MANAGING THE DATA LAKE	53
FOLDER STRUCTURE	54
EMPTY PARQUET FILES	54
INTEGRATING CUMULOCITY DATAHUB WITH OTHER PRODUCTS	55
INTEGRATING CUMULOCITY DATAHUB WITH MICROSOFT POWER BI	55
PREREQUISITES	55
SETTING UP THE CONNECTION IN CUMULOCITY DATAHUB	56
WORKING WITH REPORTS	56

CUMULOCITY DATAHUB OVERVIEW

This section outlines the structure of the document and gives a high-level introduction into Cumulocity DataHub (CDH) and its concepts.

DOCUMENTATION OVERVIEW

The following sections will walk you through all the functionalities of Cumulocity DataHub in detail.

For your convenience, here is an overview of the contents of this document:

Section	Content
Getting started with Cumulocity DataHub	Log into Cumulocity DataHub and get an overview of the UI features
Setting up Cumulocity DataHub	Set up Cumulocity DataHub and its components
Working with Cumulocity DataHub	Manage offloading pipelines and query the offloaded results
Operating Cumulocity DataHub	Run administrative tasks
Integrating Cumulocity DataHub with other products	Learn how to integrate Cumulocity DataHub with other products

The [change log](#) provides an overview on features, changes, and other relevant information.

CUMULOCITY DATAHUB AT A GLANCE

The Cumulocity platform allows you to manage and monitor a variety of devices. The data emitted by these devices is stored in the Operational Store of Cumulocity, with older data potentially being removed (based on data retention settings). In order to run an ad-hoc query against recent device data, Cumulocity offers a [REST API](#), which is described in the Cumulocity OpenAPI Specification.

In addition to this simple ad-hoc querying, various use cases require more sophisticated analytical querying over the device data, potentially covering long periods of time. Cumulocity DataHub is the tool designed for this purpose.

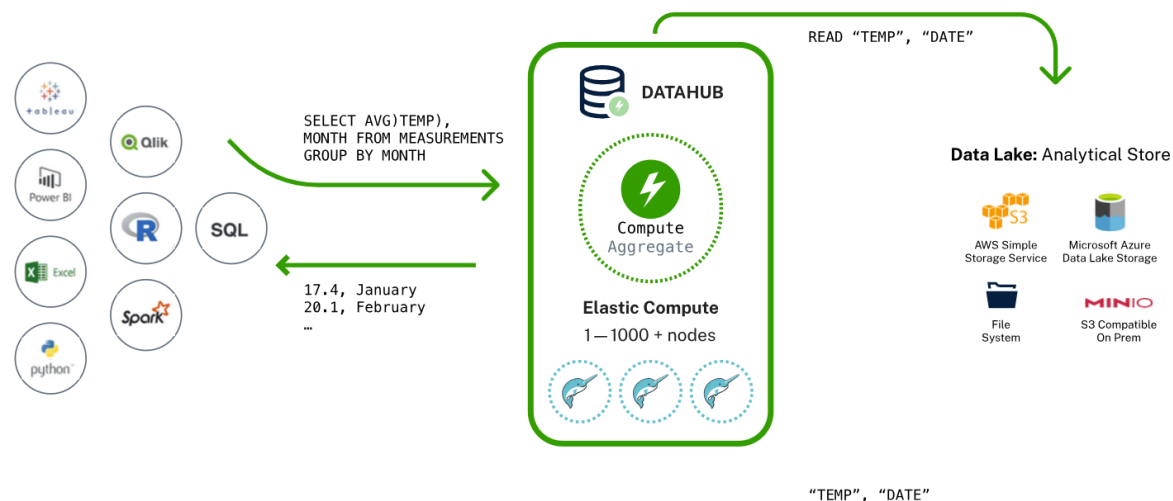
With Cumulocity DataHub, you can connect existing tools and applications to Cumulocity, such as:

- Business Intelligence/reporting tools (using ODBC, JDBC)
- Machine learning applications (mainly written in Python using ODBC)
- Arbitrary custom applications (using JDBC for Java applications, ODBC for .NET, Python, node.js, and others, or REST for web applications)

The main features of the Cumulocity DataHub application are:

- It allows you to use scalable and inexpensive storage by providing an easy-to-use data pipeline that extracts data from the Operational Store of Cumulocity to a **data lake** for long-term archival and efficient analytical querying.
- It offers an **SQL-based Query Interface** for querying the data lake and enables you to connect arbitrary applications that support ODBC, JDBC, or REST protocols.

The following diagram illustrates the high-level concepts.



The central component of Cumulocity DataHub is [Dremio](#), a distributed SQL engine that is used for the two purposes mentioned above. It offers an SQL API which can be accessed via JDBC, ODBC, and REST. Dremio is in charge of Extract-Transform-Load (ETL) pipelines that:

- Periodically extract data from the Operational Store of Cumulocity.
- Transform the data into a relational format.
- Persist the data as [Apache Parquet](#) files in the configured data lake.

When a user submits an SQL query, the query runs against data in the data lake. Thus, the Operational Store of Cumulocity is not accessed during query processing; the Operational Store is only accessed by the regular ETL process which extracts the data. Cumulocity DataHub manages those ETL processes and ensures their execution in a periodic manner.

The table below summarizes the main terms used throughout this documentation.

Component	Explanation
Cumulocity DataHub	Cumulocity application for offloading data from the Operational Store of Cumulocity to a data lake and querying the data lake contents; scheduler component (deployed as microservice) for triggering periodic offloading and UI component (deployed as web application) for defining, managing, and monitoring offloading pipelines
Cumulocity Operational Store	Internal datastore of Cumulocity where all data (alarms, events, inventory, measurements, ...) is stored in so-called base collections
Dremio	Internal SQL engine for extracting data from the Cumulocity Operational Store and writing to and reading from the data lake
Data lake	Storage container for offloaded data either on the basis of ADLS Gen2/Azure Storage (Azure), S3 (Amazon), NAS

INFO

Google Cloud Storage (GCS) is currently not supported.

DESIGN OF OFFLOADING PIPELINES

Offloading refers to moving data from the Operational Store of Cumulocity to a data lake in order to:

- Provide the data in a tabular/relational and condensed format which can be leveraged for efficient SQL-based querying.
- Build a low-cost long-term archive of data.
- Separate analytical workloads from operational workloads.

The starting point is one of the Cumulocity base collections, such as the measurements collection, that is to be offloaded into the data

lake. Once an offloading pipeline for this collection has been configured and started, a couple of actions take place.

INFO

Cumulocity DataHub only supports offloading for the following Cumulocity base collections: **alarms**, **events**, **inventory**, **measurements**. Offloading of other collections is currently not supported.

When an offloading job runs, the contents of the collection are offloaded. The document-based entities of the Operational Store of Cumulocity are transformed into a relational format by flattening the entries and mapping them to relational rows.

INFO

The mapping automatically extracts a standard set of attributes from each entity, such as `time`, `source`, `id`, and `type`. It transforms them into columns in the data lake table. Furthermore, it automatically transforms the contents of measurement fragments into columns of the table. Non-standard fields can also be offloaded to a limited extent. [Configuring offloading jobs](#) provides more details and examples for this transformation.

As a result of these extraction and transformation steps, the flattened data is stored in Parquet files in the data lake. Apache Parquet is a column-based storage format which enables compression and efficient data fetching. These Parquet files are managed in a folder structure based on a temporal hierarchy, because analytical queries commonly have a temporal background. For example, compute the average oil pressure of last month. In order to ensure a compact layout of the Parquet files, Cumulocity DataHub also regularly runs a compaction algorithm over these files in the background, which combines multiple smaller files in larger files. As the data is stored in a time-based hierarchical manner in the data lake, Cumulocity DataHub can efficiently prune partitions. In addition, queries can explicitly leverage this temporal structure to increase query performance.

IMPORTANT

You must not modify the folders in the data lake being created by Cumulocity DataHub as this will corrupt your offloading pipelines and neither data consistency nor completeness can be guaranteed any more.

The scheduler of Cumulocity DataHub runs the offloading pipelines in a periodic manner. The UI shows for each pipeline the corresponding schedule. Within each of these pipeline executions, newly arrived data is extracted from the Cumulocity collection and transformed and stored in the same way as described above. These incremental offloading tasks are designed to ensure a loss-free and duplicate-free offloading from the collection. For example, if one offloading execution fails, the next execution will automatically pick up the increments the failed one should have processed.

For each offloading pipeline, a so-called **target table** is created in Dremio that points to the corresponding data folders in the data lake. When you want to run queries with Dremio against the offloaded data, you must use these target tables.

GETTING STARTED WITH CUMULOCITY DATAHUB

The *Getting started* section describes how to access Cumulocity DataHub and walks you through the main UI features.

ACCESSING AND LOGGING INTO CUMULOCITY DATAHUB

You access Cumulocity DataHub via a web browser. It has been tested with the following web browsers:

- Firefox (latest version)
- Chrome (latest version)

INFO

Mobile devices like smartphones or tablets are not supported.

HOW TO LOG INTO CUMULOCITY DATAHUB

After the Cumulocity DataHub services have been subscribed for your tenant, you must log into your tenant, using your Cumulocity account. In the **application switcher** you will find the Cumulocity DataHub button. By clicking the button you will be taken to the home page of Cumulocity DataHub. The UI provides management and monitoring capabilities for Cumulocity DataHub. Alternatively your tenant administrator can give you the URL directly navigating to Cumulocity DataHub, which requires a login as well.

As working with Cumulocity DataHub requires you to have corresponding [Cumulocity DataHub permissions](#), you will get a warning after login if you lack those permissions.

When you want to log out, click the **User** button at the right of the top bar and select **Logout** from the context menu.

In addition to your Cumulocity account, you have a separate account (which is created during the initial configuration) for accessing Dremio. Contact your administrator for the Dremio account credentials. If required for your use case your administrator can also create additional Dremio users. On the **Home** page under **Next steps**, there is a direct link to the Dremio UI.

OVERVIEW OF UI FEATURES

Cumulocity DataHub provides the UI for managing and monitoring your offloading pipelines. The main navigation bar at the left provides links to the relevant pages. The access to these pages is restricted and depends on corresponding user roles/permissions as defined in [Defining Cumulocity DataHub permissions and roles](#).

Page	Description	Required role	Prerequisites
Home	Get an introduction to Cumulocity DataHub, access quick links with related functionality, or investigate the current offloading status	DataHub Reader, DataHub Administrator, or DataHub Manager	-
Offloading	Configure and manage your offloading pipelines	DataHub Administrator or DataHub Manager	-
Query log	View the query log	DataHub Reader, DataHub Administrator, or DataHub Manager	Tracking of usage statistics must be enabled

Page	Description	Required role	Prerequisites
Settings / Initial configuration	Set up Cumulocity DataHub	DataHub Administrator	-
Settings / Dremio users	Manage Dremio users	DataHub Administrator	-
Settings / Microsoft Power BI	Set up connection to Microsoft Power BI	DataHub Administrator	-
Microsoft Power BI	View Microsoft Power BI reports	DataHub Reader, DataHub Administrator, or DataHub Manager	Connection to Microsoft Power BI has been set up
Administration / Usage statistics	View usage statistics	DataHub Reader, DataHub Administrator, or DataHub Manager	Tracking of usage statistics must be enabled
Administration / Compaction status	View the latest compaction job status of your offloading pipelines	DataHub Administrator	-
Administration / System status	Check system status	DataHub Administrator	-

HOME

The Home screen is the starting point for working with Cumulocity DataHub. **Next Steps** provides a collection of links which directly navigate you, for example, to the Dremio UI or to the offloading configuration form. **Info** provides for the different connectivity options of Cumulocity DataHub, like ODBC or JDBC, the corresponding connection details.

Once you have offloading pipelines configured, the Home screen provides an offloading status overview. The overview is based on the latest execution of each currently configured offloading pipeline. The latest execution has either failed or succeeded or it is still running. Click **Reload** in the action bar to reload the status and refresh the overview. Running and failed pipelines are available in tables. Clicking on a table entry navigates you to the job history of the corresponding offloading pipeline. Successful pipeline executions are depicted in a graph, which has the execution time on the x-axis and the number of offloading records on the y-axis. You can use that graph to find offloading pipelines with an unexpected behavior. For example, the pipeline offloads a moderate number of records, but requires a very long timespan for the execution. More details of a successful execution are shown when hovering over an entry or clicking on it, which navigates you to the job history.

SETTING UP CUMULOCITY DATAHUB

This section describes how to set up Cumulocity DataHub.

PREREQUISITES

Before setting up Cumulocity DataHub, the following prerequisites must be checked:

The Cumulocity DataHub microservice and web application must be available as applications on your tenant. The web application provides the user interface to configure Cumulocity DataHub and to manage offloading pipelines, while the microservice provides the corresponding backend functionality. The web application is named **DataHub**, whereas the microservice is named **Datahub**. Both applications are deployed either as:

- Subscribed application: the applications were subscribed to the tenant by the management or super tenant
- Custom application: the applications were added to the tenant

If you have an Enterprise tenant, you can also subscribe your subtenants to both applications so that the subtenants can use Cumulocity DataHub as well.

See [Managing applications](#) for details on managing Cumulocity applications in general, including instructions for adding applications to a tenant.

See [Managing microservices](#) and [Monitoring microservices](#) for details on Cumulocity microservices, including instructions for:

- Adding microservices to a tenant
- Checking the status, permissions, and log files of a microservice

See [Managing tenants](#) for details on subscribing applications or microservices to a tenant or subtenant.

For the offloading of Cumulocity data, you need the connection settings and credentials for a cloud data lake service. During offloading, the data will be written into a data lake folder named after the tenant name.

INFO

This section provides instructions on how to configure the data lake so that it is accessible via Dremio. More details can be found in the [Dremio data source documentation](#). Note that you must not create the target table, which connects to the data lake, in Dremio; this is done by Cumulocity DataHub.

DEFINING CUMULOCITY DATAHUB PERMISSIONS AND ROLES

Dedicated permissions define what a user is allowed to do in Cumulocity DataHub. You assign these permissions to users by using roles, which group permission into bundles. During deployment of the Cumulocity DataHub applications the corresponding permissions as well as roles are created. If a role with the same name already exists, no new role will be created. The same holds for permissions.

If you do not have corresponding Cumulocity DataHub permissions, you will get a warning after login.

IMPORTANT

When offloading the inventory/events/alarms/measurements collection, Cumulocity DataHub does not incorporate access limitations for these collections as set in the Cumulocity platform. In particular, [inventory](#)

[roles](#) defining permissions to device groups are not incorporated in the offloading process. As a consequence, a user with Cumulocity DataHub permissions can access all data in the data lake irrespective of access restrictions the user has on the base collections.

CUMULOCITY DATAHUB ROLES AND PERMISSIONS

Cumulocity DataHub administrator

The administrator primarily sets up the data lake and Dremio account and conducts administrative tasks like inspecting audit logs or monitoring the system status. The administrator can also manage offloading pipelines, for example, defining and starting a pipeline.

For those tasks the default role DataHub Administrator is created. The permissions for this role are defined as follows:

Type	READ	ADMIN
DataHub administration	yes	yes
DataHub management	yes	yes
DataHub query	yes	no

While READ refers to reading the specific data, ADMIN refers to creating, updating, or deleting the specified data.

Cumulocity DataHub manager

The manager manages offloading pipelines such as defining and starting a pipeline. For those tasks the default role DataHub Manager is created. The permissions for this role are defined as follows:

Type	READ	ADMIN
DataHub administration	no	no
DataHub management	yes	yes
DataHub query	yes	no

Cumulocity DataHub user

The user executes SQL queries against the data in the data lake. For details on querying the data lake see [Querying offloaded Cumulocity data](#). To execute queries the following approaches can be used:

- Dremio UI: The Dremio account defined in [Setting up Dremio users](#) is used for logging into the Dremio UI and executing queries within that UI.
- Dremio API: Queries can also be executed using the Dremio REST API. The Dremio account defined in [Setting up Dremio users](#) is used for authenticating the requests against that API. Directly invoking Dremio APIs is discouraged; they might be removed or changed at any time without prior notice.
- Cumulocity DataHub proxy API: Cumulocity DataHub provides an API which proxies requests to the Dremio API. The Cumulocity user needs the role DataHub Reader in order to execute queries using the proxy API. The authentication against Dremio is done behind the scenes.

The permissions for the role DataHub Reader are defined as follows:

Type	READ	ADMIN
DataHub administration	no	no

Type	READ	ADMIN
DataHub management	no	no
DataHub query	yes	no

ASSIGNMENT OF CUMULOCITY DATAHUB ROLES AND PERMISSIONS

The roles DataHub Administrator, DataHub Manager, and DataHub Reader must be assigned to the respective users of your tenant. For assigning roles to users see [Managing permissions and roles](#). You need at least one user with the DataHub Administrator role to complete the Cumulocity DataHub configuration.

INFO

You do not necessarily need to use the predefined roles to enable Cumulocity users to work with Cumulocity DataHub. Alternatively, you can modify other roles the users are associated with and add the corresponding permissions to those roles. In that case you also must add the DataHub application to the user's applications.

SETTING UP THE INITIAL CONFIGURATION

The setup of Cumulocity DataHub requires you to configure a Dremio API user and access to a data lake. In the navigator, select **Initial configuration** under **Settings** to define those settings.

REQUIREMENTS

You need administration permissions to define the settings. See [Defining Cumulocity DataHub permissions and roles](#) for details.

DEFINING THE INITIAL CONFIGURATION

Dremio API user

In order to access the data lake contents, you can use ODBC, JDBC, Dremio REST API, or a proxy REST API. See [Querying offloaded Cumulocity data](#) for more details. The proxy REST API is served by the Cumulocity DataHub server, which acts as a proxy to Dremio. The proxy API requires a Dremio user for the interaction of Cumulocity DataHub server and Dremio. This Dremio API user can then also be used for data lake querying based on JDBC, ODBC, or Dremio REST API.

Therefore, you must configure in the initial configuration under **Dremio API user** the name and the password of that Dremio API user.

The name is composed of two parts, with the first part being fixed:

1. Tenant ID plus forward slash
2. String with a minimum length of three, starting with a character, and consisting of numbers, characters, dash, or underline

The password of the Dremio API user must have at least eight characters, including at least one character and one number.

INFO

When using the proxy REST API, all queries are processed using the same Dremio API user. The queries

are listed in the query log. Thus, the log shows all queries of all users having leveraged the proxy API.

Your follow-up application might require more than one Dremio user for accessing the data lake. You can define additional Dremio users for that purpose, using the instructions in [Adding a Dremio user](#).

Data Lake

Depending on the configuration of the environment, the data lake provider is either fixed or you can choose among different providers. For each data lake provider, you must specify corresponding settings to define the data lake to be used. See also [Managing the data lake](#) for details on managing a data lake.

✔ REQUIREMENTS

The setting **Partition Column Inference** must not be enabled as this lets Dremio assume a specific folder structure, which conflicts with the folder structure used by Cumulocity DataHub.

The following types of data lakes are currently supported:

Azure Storage

Azure Storage is a set of cloud storage services offered by Microsoft. Cumulocity DataHub supports Azure Data Lake Storage Gen2, which is part of these services. The following settings must be defined for this data lake:

Settings	Description
Azure Storage account name	The name of the Azure storage account
Azure Storage container	The name of the storage container; it must be between 1 and 63 characters long and may contain alphanumeric characters (letters and numbers) as well as dashes (-)
Root path	The root path within your data lake for storing the offloaded data. With the default path /, data is stored top-level in your storage container. You can also store data in a subfolder, provided the folder already exists. For example, for storage container <code>myContainer</code> and subfolder <code>mySubFolder</code> , use <code>/myContainer/mySubFolder</code> as root path. This option is especially useful to hide other data inside the container from Cumulocity DataHub, for example, when the container is also used by other users or applications.
Azure Storage shared access key	The access key used for authentication if "Shared Access Key" is used as authentication type
Application ID	The application ID used for authentication if "Azure Active Directory" is used as authentication type
OAuth 2.0 Token Endpoint	The authentication endpoint if "Azure Active Directory" is used as authentication type

Settings	Description
Client Secret	The client secret if "Azure Active Directory" is used as authentication type

While the other settings are fixed once the initial configuration was saved, the authentication type as well as the values of the selected authentication type can be changed afterwards. Click **Edit**, set new values, and either click **Save credentials** to save the update or **Cancel** to keep the old values.

✔ REQUIREMENTS

Note that the account type must be **StorageV2**, and the **Hierarchical namespace** feature must be activated for the corresponding Azure Storage account. It is for performance reasons recommended to set the **Blob access tier** to **Hot**. Also note that in case IP white-listing is activated, Cumulocity DataHub might not be able to access the data lake if the data lake and Cumulocity DataHub reside in the same Azure region. See also the corresponding [documentation](#).

Amazon S3

Amazon S3 is an object storage service offered by Amazon Web Services. The following settings must be defined for this data lake:

Settings	Description
AWS access key	The access key
Access secret	The access secret
Bucket name	The name of the S3 bucket; it must be between 1 and 63 characters long and may contain alphanumeric characters (letters and numbers) as well as dashes (-)
Root path in bucket	The root path within the S3 bucket; default root path is /; setting a subfolder allows you to hide other data in the bucket from Cumulocity DataHub

While the other settings are fixed once the initial configuration was saved, the **AWS access key** and the **Access secret** can be changed afterwards. Click **Edit**, set new values, and either click **Save credentials** to save the update or **Cancel** to keep the old values.

✔ REQUIREMENTS

An S3 bucket with default settings works. If specific security policies are applied, make sure that the minimum policy requirements listed in <https://docs.dremio.com/current/data-sources/object/s3/> are satisfied.

Server-side encryption is supported while client-side encryption is not. S3 offers three key management mechanisms:

SSE-S3: An AES256 key is generated in S3 and saved alongside the data. Enabling SSE-S3 requires to add the following key-value pair to the **Additional Properties** sector:

Name: `fs.s3a.server-side-encryption-algorithm`

Value: `AES256`

SSE-KMS: An AES256 key is generated in S3, and encrypted with a secret key provided by Amazon's Key Management Service (KMS). The key must be referenced by name by Cumulocity DataHub. Enabling SSE-KMS requires to add the following key-value pairs to the **Additional Properties** sector:

Name: `fs.s3a.server-side-encryption-algorithm`

Value: `SSE-KMS`

Name: `fs.s3a.server-side-encryption.key`

Value: Your key name, for example, `arn:aws:kms:eu-west-2:123456789012:key/071a86ff-8881-4ba0-9230-95af6d01ca01`

SSE-C: The client specifies an base64-encoded AES-256 key to be used to encrypt and decrypt the data. **Cumulocity DataHub does not support this option.**

NAS

NAS is a storage system mounted (NFS, SMB) directly into the Dremio cluster. It is only available for Cumulocity Edge installations.

Saving settings

Once all settings are defined, click **Save** in the action bar to the right. During the save process, the following steps are automatically conducted:

- A Dremio API user is created; the user has standard Dremio user privileges, not admin privileges.
- A data lake connection in Dremio is created using the provided data lake settings. For Dremio that connection is technically spoken a source. In our context we refer to it as **target table** as this data lake is used for storing the offloaded data.
- A source in Dremio is created which connects to the Operational Store of Cumulocity. That source is not visible to the Dremio API user.
- A space in Dremio is created which you can use to organize your custom Dremio entities such as views. The name of the space is your tenant ID concatenated with 'Space', for example, t12345Space.

EDITING SETTINGS

To edit the Dremio API user, click **Edit** in the **Dremio API user** section of the **Initial configuration** page. In the editor you can edit all user details, except for the username, which is fixed. In [Editing a Dremio user](#), all user details are described.

The data lake settings cannot be edited, except for the **Azure Storage** or **Amazon S3** credentials. For editing other values, you must delete the existing settings and define new settings. If you want to keep your offloading configurations, you must export the configurations to a backup file beforehand, delete the settings, define new settings, and import the configurations from the backup file. See [Importing/exporting offloading configurations](#) for details on import/export.

DELETING SETTINGS

Click **Delete** in the action bar to delete the settings. During deletion, all Dremio artifacts which were created when saving the settings are deleted, including the Dremio API user as well as additionally created Dremio users. Also the artifacts created by a corresponding Dremio user, like views, are deleted. All offloading pipelines and their histories are deleted; active pipelines are deleted after completing the current offloading. As mentioned in the previous section, you can use the import/export functionality to backup your offloading configurations. The data lake and its contents are not deleted, only the Dremio artefacts connecting to the data lake. To delete the data lake and its contents you must use the tooling of your data lake provider.

SETTING UP DREMIO USERS

In the [initial configuration](#) of Cumulocity DataHub, the Dremio API user is configured. This user is required for the proxy REST API, which allows you to interact with Dremio using Cumulocity DataHub. This user can also be used to directly interact with Dremio in applications, using JDBC, ODBC, or REST API.

Some use cases might require more than one Dremio user for the interaction with Dremio. For that purpose, additional Dremio users can be added.

✔ REQUIREMENTS

You need administration permissions to configure Dremio users. See [Defining Cumulocity DataHub permissions and roles](#) for details.

OVERVIEW OF DREMIO USERS

In the navigator, select **Dremio users** under **Settings** to get an overview of all Dremio users created by an administrator of your Cumulocity DataHub tenant.

The list of Dremio users with their corresponding properties is displayed. The context menu of each user provides actions to edit or delete a user.

If the initial configuration has not been completed yet, no users are shown. If the initial configuration has been completed, the list includes the Dremio API user configured in the initial configuration.

PROPERTIES OF A DREMIO USER

Username

The username is a mandatory setting. It must be a unique value, that is, no other Dremio user has the same username. It consists of the tenant ID plus forward slash and a string with a minimum length of three, starting with a character, and consisting of numbers, characters, dash, or underline. For example, the username may be `t47110815/myUser`.

First name, last name, and email

The first name, last name, and email of a Dremio user are optional settings.

Permissions for data lake and space

During the initial configuration of Cumulocity DataHub, a so-called source in Dremio is created, which connects Dremio with the data lake. Additionally, a so-called space is created in Dremio, in which Dremio artifacts like views can be organized.

The Dremio user can be assigned additional permissions for the data lake source and the space. If the user has the permission for the data lake source assigned, the user can manage grants on that source for other users as well. The same applies to the space permission. Data lake permission and space permission are independent of each other; the setting of one permission does not affect the setting of the other.

Having the corresponding permission assigned, the user can grant other Dremio users, which do not necessarily relate to Cumulocity DataHub, different permissions on the data lake source or the space, for example, for reading data from the data lake or creating a table in the data lake.

⚠ CAUTION

Granting permissions to other users should be done very carefully in order to avoid that sensitive information is exposed to the wrong users. In particular, permissions should never be granted to all users as in that case all Dremio users of the Cumulocity instance can access the data lake source or space respectively. For that reason, the system regularly checks for each tenant whether its resources are accessible by all other tenants. If that is the case, the permissions are automatically restricted. Additionally, an alarm is generated whose type is defined as `CDH_privilegeGrantedToPublicOn_` followed by the name of the asset being exposed.

For example, IoT data has been offloaded to the data lake using Cumulocity DataHub. A data scientist from a different business unit now wants to access the data lake contents. A Dremio account must be created for the data scientist. Then, a Dremio user created by Cumulocity DataHub, having the data lake permission, grants read access on the data lake source to the Dremio account of the data scientist.

ℹ INFO

Dremio refers to the permissions as **privileges**. Privileges include for example SELECT, ALTER, CREATE TABLE, or DROP. A Dremio user with the corresponding permissions can grant permissions to other users via the Dremio UI. In the UI, browse to the data lake or space and select **Edit details** in the context menu. In the editor, the list of privileges for all users is shown, with the option to update privileges and users.

Alternatively, you can use the [Dremio SQL API](#) to modify privileges.

For each user, including the Dremio API user, the `manage grants` permissions on data lake and space are initially not set.

Password

The password must have at least 8 characters with at least one letter and one number.

ADDING A DREMIO USER

To add a Dremio user, select **Dremio users** under **Settings** and click **Add user** at the right of the top menu bar. In the editor, provide the corresponding Dremio user properties.

Click **Save** to save the settings and create the new user. Click **Cancel** to cancel the creation of the user.

EDITING A DREMIO USER

The **Dremio user** section under **Settings** displays the list of users. For each user, there is a context menu on the right side. Select **Edit** from that menu to edit a user. Except for the username, all settings can be changed. The password can also optionally be changed by clicking **Change password**. Click **Save** to apply the new settings.

DELETING A DREMIO USER

In the context menu of the Dremio user list, select **Delete** and click **Confirm** in the subsequent confirmation dialog to delete a Dremio user. The Dremio API user defined in the initial configuration cannot be deleted that way. This user can only be deleted if the settings under **Initial configuration** are deleted. In the latter case, all Dremio users associated with this Cumulocity DataHub instance are deleted.

WORKING WITH CUMULOCITY DATAHUB

This section describes how to offload data from the Operational Store of Cumulocity to a data lake using Cumulocity DataHub.

OVERVIEW

Cumulocity DataHub provides functionality to configure, manage, and execute offloading pipelines that extract and transform data from the Operational Store of Cumulocity and offload it to a data lake.

You need configuration or administration permissions to work with offloading pipelines. See [Defining Cumulocity DataHub permissions and roles](#) for details.

Section	Content
Configuring offloading jobs	Configure pipelines for offloading data into a data lake
Offloading Cumulocity base collections	Examine the result schemas for offloaded Cumulocity base collections
Managing offloading jobs	Schedule and manage offloading pipelines
Monitoring offloading jobs	Monitor the results of offloading jobs
Monitoring compaction jobs	Monitor the results of compaction jobs
Querying offloaded Cumulocity data	Query offloaded Cumulocity data in follow-up applications
Refining offloaded Cumulocity data	Use Dremio to refine offloaded Cumulocity data
Cumulocity DataHub best practices	Learn more about best practices when working with Cumulocity DataHub

BASIC FUNCTIONALITY

On the **Offloading** page you do the offloading management and monitoring tasks:

- Selecting a Cumulocity base collection to offload
- Defining and validating an offloading configuration
- Editing, copying, or deleting an offloading configuration
- Importing/exporting offloading configurations
- Scheduling or manually triggering offloading executions
- Viewing the history of offloading executions

OFFLOADING OVERVIEW

In the main panel of the **Offloading** page, you will find all pipelines as well as their current status.

In the action bar you have a search field to search for all offloading configurations whose task name, description, filter predicate, additional columns, or UUID contain the search string. You can use the **Active/Inactive** filter to show/hide configurations being active or inactive respectively. The action bar also provides buttons for adding an offloading configuration, reloading the list of configurations and their status, and importing/exporting configurations.

Below the action bar you find the current list of configurations.

Offloading list

Each offloading configuration provides the following information:

Active

The toggle shows the current job state and can be used to activate or deactivate an offloading job.



Job name

The name of the job refers to the task name as defined in the configuration process. The sort control allows for sorting by job name.

Target table name

The target table name refers to the target table in Dremio, with which the data offloaded by this offloading pipeline can be queried. The sort control allows for sorting by target table name.

Offloading status

The offloading status is empty if the offloading has not been executed yet. For running and completed executions, the start time is shown and either a calendar icon  for a scheduled execution or a user icon  for a manually triggered execution. For a running execution, the elapsed time is additionally shown. For a completed execution, the failure/success status, the number of offloaded records and the runtime are shown as well. For running and completed executions, click the offloading status to navigate to the detail view for that execution in the job history.

The sort control allows you to sort by successful/failed jobs. The filter control allows you to filter by execution status.

Compaction status

The compaction status is empty if the offloading has not been executed yet. If the compaction has been executed, the status of the most recent run is shown. This includes the execution time and whether the execution was successful or not, indicated by a success or failure icon. In case of a successful run, the runtime is shown as well. The sort control allows for sorting by successful/failed jobs. The filter control allows for filtering by execution status. The compaction status is only available for users with administration permissions.

Additional information

When expanding a configuration, the job schedule, the additional columns, and the filter predicate are shown as well as additional information. This includes links that allow you to explore the table and views associated with this offloading configuration in the Dremio UI. The links are available if the pipeline has been executed at least once.

Context menu

In the context menu of a configuration you find controls for managing the offloading process as described in more detail in the next sections.

ALIGNING DATA MODELING AND OFFLOADING

MAPPING DOCUMENT DATA TO RELATIONAL DATA

Cumulocity DataHub allows to offload data from the Cumulocity platform into a data lake and the subsequent analysis of the offloaded data using SQL. The data within the Cumulocity platform is stored in a document-based format in an operational database. For the offloading, Cumulocity DataHub must transform the data from the document-based format into a relational, columnar-based format. Following the [domain model of Cumulocity](#), a document comprises attributes and fragments. These document structures are flattened and mapped to columns of a relational table. Using that relational format, the data is persisted as Parquet files in the data lake and can be queried like a relational table.

For that mapping from a document structure to columns of a relational table schema, the following cases apply:

- **Default columns:** For each collection in the operational database, a set of default columns, like `id` or `creationTime`, is defined, with the according attributes/fragments existing in each document. The relational schema always contains these default columns, all having a fixed type. Thus, each offloading for a collection has this set of default columns defined in its target table in the data lake.
- **Additional columns:** The documents may contain additional top-level attributes/fragments being either based on the Cumulocity domain model, like `c8y_Position`, or being custom, like `myCustomFragment`. They are also mapped to relational columns. The offloading of these columns is optional. You can select/deselect them during the offloading configuration process.
- **Missing columns:** During design time of an offloading, the data in the associated collection may not yet contain all columns you want to use in the offloading configuration. In such a case you can adapt the schema by adding a column with a corresponding

type. When you have added such a collection column to the schema, you can select it in your offloading configuration.

- **Derived columns:** Given one of the above columns, you can derive a new column using the value of the original column and modifying it. For example, given a column with temperature values, you can define a new column by appending "Celcius" to each value.

See [Set additional result columns](#) for details on how to work with additional, missing, and derived columns.

The data in the operational store is document-based and organized within collections like alarms or measurements. The operational store does not impose a schema for a collection. However, Dremio as internal query engine of Cumulocity DataHub reads data from the collections and derives a relational schema based on the data. With new data entering the platform, the schema of the collections may also evolve. For example, a new firmware of a device introduces new data fragments in the documents. To keep track with potential schema evolutions, the system periodically inspects a sample of the documents in the operational store for new schema information. This process also includes deriving the type of the columns from the sample data.

MANAGING MIXED TYPES

If instances of the attribute have diverging types, then the system can apply a type coercion mechanism to resolve such a mixed type constellation. The coercion mechanism derives a single, suitable type from the diverging types. For example, INTEGER and FLOAT are coerced to FLOAT while TIMESTAMP and VARCHAR are coerced to VARCHAR. You can configure how to deal with mixed types for each offloading pipeline. By default the system automatically resolves the mixed type by evolving the schema. That schema evolution uses the type coercion to introduce a new column with the coerced type. Alternatively the system stops the pipeline to allow for corrections. [Dealing with mixed types](#) describes how to configure those strategies.

! IMPORTANT

Even though the system can resolve mixed types, it is strongly advised to avoid them as they may introduce additional adaptation effort. Take care that your data model does not mix up types and that you feed only type-consistent data into the Cumulocity platform.

MAPPING MEASUREMENT FRAGMENTS TO RELATIONAL DATA

The offloading configuration mechanisms differ when dealing with series-value fragments of measurements. As additional fragments are often added dynamically, Cumulocity DataHub automatically picks up each series at runtime without the need to reconfigure the offloading pipeline.

Each series must have a mandatory **value** of type NUMBER and an optional **unit** of type STRING. If the value is not of type NUMBER, Cumulocity DataHub determines a type for each series at offloading runtime. It evaluates the runtime type of each value and derives the column type for the corresponding offloading run. If all values for a series can be cast to BOOLEAN, FLOAT, STRUCT or LIST consistently, this will be the type of the resulting column. Otherwise, DataHub will use VARCHAR. If the use case mixes types for the same series, the aforementioned mixed type handling applies.

i INFO

The Cumulocity platform supports the time series model, which is an internal data model for the measurements collection. For details on how to activate that model see [Enhanced time series support](#). Cumulocity DataHub supports that data model when offloading the measurements collection. When you switch from the default data model to the time series model, measurements offloadings still work. The switch back from the time series model to the default model is not supported. In that case the offloading cannot guarantee that all data is offloaded into the target table. To ensure completeness, re-configure the offloading to use a different target table when switching to the default data model.

DEALING WITH CASE-SENSITIVITY

Case-sensitivity issues arise when attributes have the same name, except for their case. For example, a device measurement has an attribute named **c8y_pressure**, while a subsequent measurement has an attribute named **C8Y_Pressure**. When offloading such data with mixed cases, the resulting table schema as well as the table contents in the data lake may not meet your expectations.

Along the offloading workflow, different components are involved with a different handling of case-sensitivity. The operational store is configured to read the data case-sensitively, which then also applies to querying via the Cumulocity REST API. Thus, given two measurements each with an attribute named `c8y_pressure` and `C8Y_Pressure` respectively, only the document with `c8y_pressure` is returned when querying for `c8y_pressure`. Dremio and its SQL interface in turn are case-insensitive. In the offloading process, Dremio queries the operational store and derives a schema for the table in the data lake. This schema consists of default columns, additional result columns, and, for the measurements collection, columns derived from series values fragments.

Case-sensitivity and default column names

For each base collection [default columns](#) are defined, based on attributes in the operational store. When feeding data into the platform, the correct case for the associated attribute must be used. For example, an alarm has the attribute `status`; the corresponding value is then stored in the `status` default column in the data lake table. When using `Status` as attribute name, not the data in the document will be offloaded, but a default value, in this case ACTIVE, will be set in the data lake.

Case-sensitivity and additional result column names

In addition to the default columns, the data in the operational store may have additional top-level attributes, which can be used as additional result columns in the offloading process. If the same top-level attribute is defined with different cases in the documents, not all associated attribute values will be offloaded into the corresponding column. Dremio maintains one column in its collection metadata. For all documents whose attribute name exactly equals that column name the corresponding value will be offloaded. For documents with a different attribute name case, the value in the data lake will be set to NULL. Thus, using different cases in attribute names will cause data loss. For example, a top-level attribute in an alarm is named `owner`; Dremio uses this name in its metadata on the alarms collection. The `owner` value of all alarms is then offloaded into the `owner` column in the data lake. If an alarm has an attribute `Owner`, not the associated value will be stored in the `owner` column in the data lake, but NULL instead. You also have to take the case-sensitivity of the column name into account when adding a column missing so far.

Case-sensitivity and measurement column names

For the measurements collection operating in the non-time-series mode, duplicate column names can occur. Section [Configure additional settings and complete configuration](#) describes the background and how to enable a name sanitization mechanism, which generates new columns in case of duplicate names. This configuration option is only available in the legacy non-time-series mode.

Duplicate names in one document

If two attribute names in a document only differ by case, only one of the values will be used for the offloading. For example, an alarm has an attribute `owner` as well as an attribute `Owner`. Then only one attribute value will be offloaded and the other one will be ignored.

! IMPORTANT

It is strongly advised to avoid feeding data into the platform which is ambiguous with respect to case-sensitivity. This problem mainly refers to the naming of attributes in documents sent to the platform. Such data might cause unexpected results in the data lake, including data loss.

Dealing with case-sensitivity issues

The system does not raise a warning when case-sensitivity issues occur, but processes the data as described above. If your application may be prone to generating data causing case-sensitivity issues, you should regularly check your data lake contents for unexpected entries. For example, you can run a SQL query to check for unexpected NULL values in the data lake.

When a case-sensitivity issue has occurred, several steps have to be conducted. If possible, the document in the operational store having introduced the issue needs to be adapted so that the expected case is used for the attribute name. The collection schema in Dremio might need to be adapted. Also the data lake might need a cleanup, including a rerun of the offloading process for that data. Also the component generating the data should be adapted so that case-sensitivity issues do not occur anymore. Adapting the schema in Dremio and cleaning up the data lake typically require additional support.

GUIDELINES

When modelling your data, take the following guidelines into account:

Description

The data type of an attribute should be static as otherwise mixed type constellations may occur.

When modelling measurements and large volumes of them are likely to be generated, leverage the time series data model. When using this model, the offloading and query performance is typically better compared to the default data model.

When modelling measurements, you should separate the measurements by specifying measurement types. Then each measurement type can be modeled within a separate offloading pipeline, which in turn leads to a cleaner data architecture in the data lake as well as better query performance.

Avoid offloading lists with many entries as this leads to broad tables in the data lake.

Avoid having a large number of columns in the data lake as this adversely affects query performance and complicates data access in follow-up applications.

When defining attribute names in your data model, avoid special characters in attribute names. The attribute names are used as column names in the resulting offloading table and special characters may hinder working with those columns in follow-up applications.

When modelling data within arrays, ensure that the position of the values within the array is fix throughout the documents being fed into the platform. Otherwise further processing might run into problems.

LIMITATIONS

When modeling your data, you must be aware of the following limitations:

Description

If the collection to be offloaded has JSON attributes consisting of more than 32,000 characters, its data cannot be offloaded. One specific case where this limitation applies is the Cumulocity application builder, which stores its assets in the inventory collection when being used.

If the collection to be offloaded has more than 800 JSON attributes, its data cannot be offloaded. This limitation also includes nested JSON content, which will be expanded into columns during offloading. Therefore, measurements documents with more than 800 series/series value fragments are not supported.

CONFIGURING OFFLOADING JOBS

The following steps describe how to set up an offloading pipeline.

DEFINING AN OFFLOADING CONFIGURATION

To define an offloading configuration, click **Offload collection** to start a wizard which guides you through the main steps:

- [Select collection](#)
- [Configure target table](#)
- [Set additional result columns](#)
- [Set filter predicate](#)
- [Set task details](#)
- [Configure additional settings and complete configuration](#)

The wizard prepopulates settings for the different steps to ease the configuration process. You can modify those settings according to your needs.

Select collection

In the dropdown box select one of the Cumulocity base collections, which are:

- alarms
- events
- inventory
- measurements

INFO

You can define multiple offloading pipelines for each Cumulocity collection. As an example for multiple pipelines, you can filter the alarms collection by different criteria with each one resulting in a separate pipeline.

In [Offloading Cumulocity base collections](#) you will find a summary of the default attributes being offloaded per base collection.

Configuring inventory collection

The inventory collection stores data related to devices and managed objects. In order to confine the offloading pipeline to the data you need, there are different views defined over the collection, with each one defining a subset of all inventory entries. Select the view fitting best to your needs.

- **All devices:** This view provides all documents with device-related data, indicated by having the `c8y_isDevice` fragment set.
- **All device groups:** This view provides all documents with data related to device groups, indicated by having the `c8y_isDeviceGroup` fragment set.
- **Inventory data tagged for DataHub:** This view provides all documents that are specifically tagged for DataHub and not related to devices. Corresponding documents have the fragment `c8y_DataHubInclude` set, but not the fragments `c8y_isDevice`, `c8y_isDeviceGroup`, or `c8y_DataHubExclude`. You can utilize the fragment `c8y_DataHubInclude` in your data-generating application to configure a custom view that offloads only selected documents.
- **All data:** This view provides all documents, except for those having the fragment `c8y_DataHubExclude` set. You can utilize the fragment `c8y_DataHubExclude` in your data-generating application to configure a custom view which excludes selected documents. Using this view is not recommended. First, it includes the offloading of data typically not required in your application, and second, the schema detection may suffer from the heterogeneity of the data.
- **All remaining inventory entries:** This view provides all documents, except for those having the fragments `c8y_isDevice`, `c8y_isDeviceGroup`, `c8y_DataHubInclude`, or `c8y_DataHubExclude` set. Using this view is not recommended. First, it includes the offloading of data typically not required in your application, and second, the schema detection may suffer from the heterogeneity of the data.

INFO

The Cumulocity DataHub Edge version currently does not support inventory views.

Older offloading configurations not yet based on a view are still supported. They are configured to directly read from the inventory collection, without an intermittent view. When editing such an offloading, the above list contains an additional option **Raw inventory collection**, which is automatically selected. It is advisable to select one of the other views to ensure that only relevant data is offloaded.

INFO

The view may be empty as no documents in the inventory collection qualify for the view definition. Then the offloading configuration cannot be completed as no schema can be derived.

Configure measurements collection

Measurements in the **measurements** base collection may have different types. For example, the collection may contain temperature, humidity, and pressure measurements. As the resulting table in the data lake must only contain measurements of one specific type, you must additionally specify the **measurement type** to which the offloaded measurements are restricted. To identify existing measurement types, Cumulocity DataHub automatically inspects a subset of the data, including initial as well as latest data. In the measurement type dropdown field, these auto-detected types are listed. If a specific type you are looking for has not been detected, you can manually enter it in this field. Alternatively, you can click **Refresh** next to the dropdown field to manually re-trigger the detection of measurement types. As this might be a performance-intensive process, you should trigger it only if you know that the expected measurement type is present in data recently inserted into the collection. You can trigger such a refresh only every five minutes for performance reasons.

Click **Next** to proceed with the next configuration step. Click **Cancel** to cancel the offloading configuration.

Configure target table

Once you have selected a collection for offloading, you must specify the target table in the data lake. The **Target table name** denotes the folder name in the data lake. In this folder, which will be automatically created, the offloaded data will be stored. In Dremio a table is created with the same name, pointing to this data lake folder. This table is used when querying the corresponding data lake folder and thus the offloaded data. The target table name must follow these syntax rules:

- It must start with an alphanumeric character (letters and numbers).
- It may contain alphanumeric characters, underscores (_) and dashes (-).
- Each underscore or dash must be preceded by an alphanumeric character.
- The name must be at least two characters long.

Each pipeline must have its own target table in the data lake. Thus, you must select distinct target table names for each offloading configuration.

For each base collection, a default set of data fields is derived. This set defines the default schema of the target table with the columns capturing the data fields. The set is fix for each collection, including the views of the inventory collection, and cannot be modified. Select **Show default schema** to show the columns of the default schema with their corresponding name and type.

Click **Next** to proceed with the next configuration step. Click **Finish** to jump directly to the final step. Both steps will fail if the associated base collection is empty, as it prevents necessary schema investigations. In such a case you must ensure that the base collection is not empty before you can proceed with the offloading configuration. Click **Previous** to go back one configuration step. Click **Cancel** to cancel the offloading configuration wizard.

Set additional result columns


Each base collection has a set of default columns, which are always offloaded. If you have fed data into Cumulocity with additional top-level fields, you can include them as well in the offloading process by setting them as additional result columns. You can also use additional result columns to offload data fields in the base collection which are not part of the default schema. Additionally, if columns are not yet present in the data or have not been auto-detected, you can add them to the collection schema used by Dremio. See also [Mapping document data to relational data](#) for the relationship between data in the operational store and columns in a data lake table.

While default columns are per default selected for the offloading process, the selection of additional result columns is optional. If selected, the corresponding values in the documents of the operational store are offloaded into the associated column in the data lake.

Auto-detected columns


To ease the configuration process, Cumulocity DataHub auto-detects additional result columns. Using a sample of the base collection, Cumulocity DataHub searches for additional top-level fields and provides them as additional result columns. For the specific case of the inventory collection, a sample of the selected view is used to derive the additional columns. Therefore they can vary with the view being selected. As the auto-detection logic relies on a sample, not all additional top-level fields present in the data might be captured.

Overview of additional result columns

When entering the configuration step for additional result columns, all columns and their properties are shown in a table. You can use the filter controls to filter for columns by name or column type. Click the expand icon  to get further details for a column. In the context menu of a column you find actions for editing, duplicating, or deleting the column. The column name can also be edited inline by clicking into the name field, adapting the name, and clicking once outside the field. If you enter the additional result columns step for an active offloading pipeline, you cannot modify the columns.

Each additional result column has the following properties:

- **Selected:** With this checkbox, you define whether the column is included in the offloading pipeline or not.
- **Column name:** The column name is the name the column will have in the target table. The column name must be unique, non-empty, and contain at least one non-whitespace character.

- **Column type:** The column type denotes whether the column is missing, has been auto-detected, manually added by the user, or derived from another column.
- **Source definition:** The source definition is the actual SQL expression, which defines what the data in this column looks like.
- **Data type:** The data type defines which kind of data the column contains, for example, DOUBLE for double values or VARCHAR for strings. When expanding an additional column by clicking the expand icon , sample data of the column is shown. Additionally, the complete data type definition for complex types like LIST or STRUCT is shown.

Add a missing column

Click **Collection column** to add a column not yet present in the schema Dremio associates with the collection. In the upcoming dialog you need to specify the name of the column as well as its type. The name must be unique. When the name shall contain special characters like spaces or quotes, you need to escape it with double quotes, for example "Column with spaces". Select a type from the type dropdown. For the complex types LIST and STRUCT, you need to specify the structure, for example STRUCT(NestedColumn1 VARCHAR, NestedColumn2 BOOLEAN). Click **Confirm** to add the column to the schema or **Cancel**. If the offloading pipeline is executed, the column in the data lake table will be NULL until the data in the operational store contains the corresponding values.

INFO

When you manually add a column to the schema, it will not be considered by the schema learning periodically executed by Dremio. Thus, if the type evolves, these changes will not be captured. For example, the column is defined as STRUCT(NestedColumn1 VARCHAR, NestedColumn2 BOOLEAN). Over time, the data contains a new substructure NestedColumn3 of type INTEGER. Then the column will still be defined as STRUCT(NestedColumn1 VARCHAR, NestedColumn2 BOOLEAN). To capture this change, you need to manually adapt the type definition.

Add a derived column

Click **Derived Column** to add a derived column, which opens a dialog box for defining the column. You must define a unique column name as well as a source definition. Regarding the source definition, the first step is to specify a field from the base collection in the source definition editor. Then you can optionally apply SQL functions to adapt the data of this field to your needs, for example, by trimming whitespace or rounding decimal values. The source definition editor supports you in this process with content completion and syntax highlighting. The **Change data type** controls helps you to define a function which changes the data type of the source definition. For example, the source definition is of type VARCHAR and corresponding values are always either true or false. Then you can select BOOLEAN in the **Change data type** dropdown box to define a function which casts the VARCHAR values to BOOLEAN. Different target data types are available in the control, with some of them having options for dealing with non-matching values. For example, if you want to cast all values to type INTEGER and the non-matching literal N/A is processed, you can configure the casting function to use value 0 instead. If you have selected a target data type, click **Apply** to apply or **Cancel** to revert that type change. Note that functions you can apply to the source definition are not limited to the data type change functions provided under **Change data type**. In the source definition editor you can apply all SQL functions supported by Dremio, as listed under [SQL Function Categories](#).

If you want to derive columns from nested content, you can specify the nested fields using the prefix "src." and the path to the nested field. For example, if you have a top-level field "someField" with a nested field "someSubField", add "src.someField.someSubField" as additional result column. In the same way you can access nested arrays. If you have a top-level field "someField" with a nested array field "someArraySubField", add "src.someField.someArraySubField[0]" as additional result column to access the first array entry.

To validate the source definition and preview its results click **Load samples**. The system retrieves data of the associated collection, per default from the last 24 hours, and evaluates the source definition against that data. Results being **NULL** are filtered out. The maximum number of results is limited to 100. You can adjust the timeframe from which data is sampled using the time controls at the right top. The timeframe covers at maximum the last seven days. To search for specific sample values, filter the current list of sample results with the filter controls at the top. The type of the sample results depends on the source data and the source definition. For complex types like **STRUCT** browse through the nested content of a sample entry by clicking at the nodes within the entry. If you want to set the source definition to a specific path of an entry, navigate to that path and click the hand icon right next to the path. You can also copy the path using the copy icon next to the path. Once you modify the source definition, the current sample results typically do not match anymore. Click **Reload** to retrieve a list of sample results with respect to the new source definition.

Click **Save** to add the column, which will be selected for offloading by default. If the source definition is invalid, for example when accessing an unknown column, you get an error message like *Column "UnknownColumn" not found in any table*. You must fix the source definition before you can proceed. Click **Cancel** to cancel the configuration.

Edit an additional result column

In the context menu of an additional result column, select **Edit** to open the dialog for editing the column. Adapt the settings according to

your needs. Click **Save** to update the column with the new settings or **Cancel**.

For auto-detected columns the source definition cannot be modified. If you want to modify the source definition, you must duplicate the auto-detected column and modify the source definition as required.

Duplicate an additional result column

In the context menu of an additional result column, select **Duplicate** to open the dialog for duplicating the column. The source definition of the duplicate column is the same as of the original column and can be adapted to your needs. The new column name initially uses the original column name plus a counter suffix to make the name unique. You can change the name as required. You can also rename the original column. New as well as original column name must be unique.

Click **Save** to complete or **Cancel**.

A common use-case for duplication is to change the data type of an auto-detected column. For example, duplicate the column "statusOrdinal" and apply the corresponding casting function in the source definition editor. Use as new column name "statusOrdinal" and rename the original column to "statusOrdinal_Old". In the additional columns list select "statusOrdinal" and deselect "statusOrdinal_Old".

Delete an additional result column

In the context menu of an additional result column, select **Delete** to open the dialog for deleting the column. Click **Confirm** to proceed or **Cancel** to cancel the deletion. Auto-detected columns cannot be deleted.

When deleting an additional result column, the data will no longer be included in the next offloading run. Data which has already been offloaded to the data lake is not affected by the deletion of the column. Thus, the column itself will still be present in the data lake, but will have value NULL once the additional result column has been deleted.

Click **Next** to proceed with the next configuration step. Click **Previous** to go back one configuration step. Click **Cancel** to cancel the offloading configuration.

Set filter predicate

Optionally you can define a filter predicate. Per default, all entries in the base collection are offloaded to the data lake; you can use the predicate to filter out entries you do not want to persist in the data lake. For example, you can filter out invalid values or outliers. In the **Additional filter predicate** field, you can specify such a filter in SQL syntax. For example, for the alarms collection the filter might be `status='ACTIVE' AND severity='WARNING'` to only persist active alarms with a severe warning. The filter predicate functionality supports complex SQL statements, that is, a combination of **AND/OR**, clauses like `IN(...)` / `NOT IN(...)`, and functions such as `REGEXP_LIKE(text, 'MyText\S+')`.

In the filter predicate you can query all standard attributes of the base collection as well as the custom fields. The additional result columns defined in the previous configuration step cannot be accessed by their name in the filter predicate. You must use the source definition as defined in the corresponding column instead.

INFO

For querying the attribute `id`, you must use `_id`. For examples on querying different attributes and guidelines for filters, see also [Cumulocity DataHub best practices](#).

When defining an additional filter predicate, you can click **Validate** to validate your predicate. If the validation fails, you will get an error description. You must fix these errors before you can proceed.

Click **Next** to proceed with the next configuration step. Click **Previous** to go back one configuration step. Click **Cancel** to cancel the offloading configuration.

Set task details

The task configuration step includes the offloading task name and the description. The **Offloading task name** is an identifier for the offloading pipeline. It must have at minimum one non-whitespace character. Even though the task name does not have to be unique, it is advisable to use a unique name.

In the **Description** field, you can add a description for this offloading pipeline. The description is optional, but we recommend you to use it, as it provides additional information about the pipeline and its purpose.

Click **Next** to proceed with the next configuration step. Click **Previous** to go back one configuration step. Click **Cancel** to cancel the offloading configuration.

Configure additional settings and complete configuration

The final step provides a summary of your settings, the configuration of additional settings, and a result preview. The summary includes the settings from the previous steps as well as the internal UUID of this configuration. The UUID is generated by the system and cannot be modified. With the UUID you can distinguish configurations having the same task name, for example, when browsing the audit log or the offloading status. In the summary, you also get the schedule with which the offloading pipeline will be executed once it is started, for example, "every hour at minute 6". With the **Inactive/Active** toggle at the end of the summary you select whether the periodic offloading execution should be activated upon save or not.

In the offloading preview you can inspect how the actual data will be stored in the data lake. For this purpose, an offloading preview is executed, returning a sample of the resulting data. The header row of the sample data incorporates the column name as well as the column type. Use **Hide time columns** to either show the default columns with a temporal notion or not. Note that the preview does not persist data to the data lake.

Offloading frequency

Per default each active offloading pipeline is executed once an hour, at the same minute. You can adapt the offloading frequency by setting in the dropdown box the hours per day at which the offloading will to be executed. As with the default setting, the exact minute of the hour for the execution is selected by the system. The hours are defined with respect to UTC as timezone. You must select at least one hour; otherwise the configuration cannot be saved.

Compaction strategy

In the additional settings, you can define the compaction strategy for the offloading pipeline. The compaction strategy refers to how Cumulocity DataHub automatically combines multiple smaller files in the data lake into one or more larger files. Cumulocity DataHub periodically executes the compaction for an offloading pipeline as a large number of small files may adversely affect the query performance. The compaction is executed once per day; the compaction schedule cannot be modified.

Cumulocity DataHub automatically sets the compaction strategy, but allows you to optionally change the strategy. Available compaction strategies are:

- **Monthly & daily compaction:** Cumulocity DataHub selects for each day all files in the data lake which contain data from that day. These files will be combined into one or more larger files containing all data for this day. Additionally, all days for one month are combined into one or more larger files containing all data for this month. This results in summary files for each day and for each month, while the original files are deleted.
- **Daily compaction:** Cumulocity DataHub selects for each day all files in the data lake which contain data from that day. These files will be combined into one or more larger files containing all data for this day. This results in summary files for each day, while the original files are deleted.
- **No compaction:** Compaction is disabled. This setting is not recommended and must be used with caution as it has most likely negative impact on the query performance.

You can change the compaction strategy of an already running offloading pipeline by deactivating the pipeline, editing the compaction strategy, and reactivating the pipeline. If a compaction was already executed in the past, changing the compaction strategy does not revert the previous compaction results.

View materialization

In the additional settings, you can enable/disable view materialization for an offloading pipeline based on the alarms, events, or inventory collection. For these three collections, additional views over the target table are defined in the tenant's space in Dremio. The `__latest` view maintains the latest status of all entities, excluding intermediate transitions of an entity. For large tables, the maintenance of the view might adversely affect overall performance. For that reason, the `__latest` view can be materialized so that the latest state of each entity will be persisted in the data lake. If that setting is activated for a pipeline, the materialized view will be created with the next offloading run and updated for each subsequent run. If you deactivate the setting for a pipeline, the view is still available, but no more materialized.

INFO

When view materialization is activated, additional data is stored in the data lake, which might affect your storage costs.

Duplicate column names

Another setting, which applies only for the measurements collection, is the handling of duplicate column names. During offloading, measurement values are transformed into a relational format. Corresponding column names of measurement values are constructed by concatenating path and unit/value. This may lead to columns having the same name except for their case. Then the entries would all be offloaded into the same column. As this may be an unwanted behaviour in the offloading process, the names can be sanitized. When activated, for each generated column name, which would be equal to another column name in terms of case-insensitivity, a new column will be created, whose name includes the originally derived name plus a unique suffix.

The following two example documents from two different offloading runs would be processed as follows.

First document:

```
{
  "id": "4711",
  ...
  "time": {
    "date": "2020-03-19T00:00:00.000Z",
    "offset": 0
  },
  "type": "c8y_Temperature",
  "_seriesValueFragments": [{
    "unit": "C",
    "value": 17.3,
    "path": "c8y_TemperatureMeasurement.T"
  }]
}
```

Second document:

```
{
  "id": "4711",
  ...
  "time": {
    "date": "2020-03-20T00:00:00.000Z",
    "offset": 0
  },
  "type": "c8y_Temperature",
  "_seriesValueFragments": [{
    "unit": "C",
    "value": "NaN",
    "path": "c8y_temperaturemeasurement.T"
  }]
}
```

The two paths `c8y_TemperatureMeasurement.T` and `c8y_temperaturemeasurement.T` are equal in terms of case-insensitivity. Without name sanitization, only the column `c8y_TemperatureMeasurement.T.unit` will be created, which stores all unit entries. Analogously, one column `c8y_TemperatureMeasurement.T.value` will be created, which stores all value entries. In the latter case, the column would have a mixed type of DOUBLE and VARCHAR, which Dremio would then coerce to type VARCHAR for the column.

The first time an offloading run processes multiple fragments with the corresponding column names being equal with respect to case-insensitivity, the sanitization also generates distinct column names, with each name having a unique suffix.

This handling of duplicate column names does not apply when the time series data model is used. This model does not support separating incoming names which differ in case. All of them will go to the same data lake column in the offloading process. The case of the column name is determined by the first occurrence that was encountered.

Raising alarms

Offloading as well as compaction runs may fail due to various reasons such as network issues and timeouts. As described in [History per offloading pipeline](#) and [History of compactions per offloading pipeline](#), the offloading and compaction job histories provide details for successful and failed runs. Additionally, an alarm can be raised within the Cumulocity platform in case of a failure.

Under **Create alarm on** you can activate raising alarms for offloading as well as compaction failures. Per default, the setting is activated for offloading and deactivated for compaction failures. When activated and an offloading run fails, an alarm is raised. If the offloading fails multiple times in a row, the associated alarm is updated with each new failure. The more successive runs fail, the higher the severity of the alarm will be, ranging from warning up to critical. Each alarm comprises information which offloading pipeline has failed and how often

it has failed in a row. The same applies to alarms being raised for compaction failures. The type of an alarm is named *CDH_offloading_* or *CDH_compaction_* respectively, followed by the UUID of the offloading pipeline. Such an alarm is available in the Cumulocity [Device Management application](#).

The alarm will be active until it is cleared. The latter is the case when either an offloading run completes successfully, or the offloading configuration is deleted. Then, the active alarm is cleared, no matter if the alarms setting is activated or not. The alarm remains active if the offloading is unscheduled or raising alarms is deactivated. Again, the same applies to alarms being raised for compaction failures.

Raising events on completion

The sections [History per offloading pipeline](#) and [History of compactions per offloading pipeline](#) describe the offloading and compaction job history respectively, both providing details for successful and failed runs. Additionally, an event can be raised within the Cumulocity platform when an offloading or compaction job has completed, either with success or failure. Under **Create events on** you can activate raising events for completed offloading and compaction runs. When activated and an offloading or compaction run is completed, a new event with execution details is raised. Per default, the setting is deactivated in order to prevent sending events unnecessarily.

The event is available in the Cumulocity [Device Management application](#). Each event comprises the standard event properties like time and type. The latter is named *CDH_offloading_* or *CDH_compaction_* respectively, followed by the UUID of the offloading pipeline. Additionally, each event comprises the name, ID, execution ID, and status of the associated offloading or compaction job. The status provides details of the offloading run like start and end time, failure/success, number of offloaded records.

The option to send and receive such events allows you to act upon completion of an offloading run. Given the event type of the offloading you are interested in, for example *CDH_offloading_92622259-c78b-408e-95e2-30944de2cc95*, you can use the [Events REST API](#) to retrieve associated events for that offloading pipeline. Alternatively, you can use the [Event notification API](#).

Dealing with mixed types

Each offloading pipeline must ensure that the columns of the result table in the data lake have a unique data type each. A mixed type situation occurs if an offloading run detects a data type not matching the expected column data type. For example, the type of a column is INTEGER. Then, the offloading processes the literal N/A, which is of type VARCHAR. To resolve such a mixed type constellation, you can either use the **Automatically evolve schema** or the **Stop pipeline** strategy.

Automatically evolve schema: This is the default strategy. The system automatically evolves the schema by introducing a new column for the data with a new type. The name of that column is built by combining the original column name, the literal *CDH*, and the new data type. For example, such a column is named *ValvePressure_CDH_DOUBLE* if the new type is *DOUBLE*. Each new value will from now on be stored in the new column, having the new type. It is no more stored in the original column, where it will be *NULL* instead. In the job history of the pipeline the job having detected the mixed type is marked as successful.

Stop pipeline: The system stops the pipeline in order to allow for corrective actions like modifying the data or adapting the additional result columns. After those corrections you must manually re-activate the pipeline. In the job history of the pipeline the job having detected the mixed type is marked as erroneous.

When a mixed type constellation has been detected, an alarm will be additionally raised in the Cumulocity platform with further details like involved column and types. When schema evolution is selected, an alarm is raised having type *CDH_schemaEvolved_* plus offloading UUID and alarm type WARNING. When pipeline stop is selected, an alarm is raised having type *CDH_pipelineStopped_* plus offloading UUID and alarm type CRITICAL. Such an alarm is always raised, independent of the configuration for raising alarms as described in the previous section. The alarm must be manually cleared. It is only automatically cleared if the offloading pipeline is deleted.

For more details on data modeling and mixed types see also [Aligning data modeling and offloading](#).

Completing the offloading configuration

Finally, click **Save** to save the offloading pipeline. Otherwise click **Cancel** to cancel the offloading configuration. You can also navigate back to adapt previous settings, using the **Previous** buttons.

OFFLOADING CUMULOCITY BASE COLLECTIONS

OFFLOADING THE BASE COLLECTIONS

The following tables summarize the resulting schemas for each of the Cumulocity base collections. These schemas additionally include the virtual columns *dir0*, ..., *dir3*, which are used for internal purposes. The columns are generated during the extraction process, but neither do they have corresponding data in the Operational Store of Cumulocity, nor are they persisted in the data lake. Do not use *dir0*, ..., *dir3* as additional columns or rename them accordingly in your offloading configuration.

INFO

For each offloading run, the current data in the collection is considered. If data has been modified multiple times or deleted between two successful offloading runs, these changes will not be captured in the offloading process and will not be reflected in the data lake. Relevant for the offloading is the current snapshot of the collection when starting an offloading run. For example, after the first offloading execution, the status of an alarm is ACTIVE. Then it changes its status from ACTIVE to INACTIVE and afterwards back to ACTIVE. When the next offloading is executed, it will persist the latest status ACTIVE, but not the intermediate status INACTIVE, because it happened between two offloading runs.

Offloading the alarms collection

The alarm collection keeps track of alarms which have been raised. During offloading, the data of the alarm collection is flattened, with the resulting schema being defined as follows:

Column name	Column type
id	VARCHAR
count	INTEGER
creationTime	TIMESTAMP
creationTimeOffset	INTEGER
creationTimeWithOffset	TIMESTAMP
lastUpdated	TIMESTAMP
lastUpdatedOffset	INTEGER
lastUpdatedWithOffset	TIMESTAMP
YEAR	VARCHAR
MONTH	VARCHAR
DAY	VARCHAR
time	TIMESTAMP
timeOffset	INTEGER
timeWithOffset	TIMESTAMP
severity	VARCHAR
source	VARCHAR
status	VARCHAR
text	VARCHAR
type	VARCHAR

INFO

The column `firstOccurrenceTime` is not included in the default schema. If you want to include it in the offloading, it must be added manually.

The alarms collection keeps track of alarms. An alarm may change its status over time. The alarms collection also supports updates to incorporate these changes. Therefore an offloading pipeline for the alarms collection encompasses additional steps:

1. Offload those entries of the alarms collection that were added or updated since the last offload. They are offloaded with the above mentioned standard schema into the target table of the data lake.
2. Additional views on the target table are defined in the tenant's space in Dremio. Their names are composed as target table name plus `_all` or `_latest`. The following examples use "alarms" as target table name:
 - **alarms_all** - A view with the updates between two offloading executions, not including the intermediate updates.
 - **alarms_latest** - A view with the latest status of all alarms, with all previous transitions being discarded. For offloading configurations with view materialization enabled, the materialized state in the data lake is used.

The views are provided in your Dremio space. For details on views and spaces in Dremio, see [Refining offloaded Cumulocity data](#). In the main panel of the **Offloading** page you find in the details section of an offloading configuration links which navigate you to the corresponding table and views in the Dremio UI.

Offloading the events collection

The events collection manages the events. During offloading, the data of the events collection is flattened, with the resulting schema being defined as follows:

Column name	Column type
id	VARCHAR
creationTime	TIMESTAMP
creationTimeOffset	INTEGER
creationTimeWithOffset	TIMESTAMP
lastUpdated	TIMESTAMP
lastUpdatedOffset	INTEGER
lastUpdatedWithOffset	TIMESTAMP
YEAR	VARCHAR
MONTH	VARCHAR
DAY	VARCHAR
time	TIMESTAMP
timeOffset	INTEGER
timeWithOffset	TIMESTAMP
source	VARCHAR
text	VARCHAR

Column name	Column type
type	VARCHAR

Events, just like alarms, are mutable, that is, they can be changed after their creation. Thus, the same logic as for alarms applies.

Additional views over the target table are defined in the tenant's space in Dremio. Their names are defined as target table name plus `_all` or `_latest`. The following examples use `events` as target table name:

- **events_all** - A view with all captured states of all events.
- **events_latest** - A view containing only the latest state of all events without prior states. For offloading configurations with view materialization enabled, the materialized state in the data lake is used.

The views are provided in your Dremio space. For details on views and spaces in Dremio, see [Refining offloaded Cumulocity data](#). In the main panel of the **Offloading** page you find in the details section of an offloading configuration links which navigate you to the corresponding table and views in the Dremio UI.

Offloading the inventory collection

The inventory collection keeps track of managed objects. During offloading, the data of the inventory collection is flattened, with the resulting schema being defined as follows:

Column name	Column type
id	VARCHAR
creationTime	TIMESTAMP
creationTimeOffset	INTEGER
creationTimeWithOffset	TIMESTAMP
lastUpdated	TIMESTAMP
lastUpdatedOffset	INTEGER
lastUpdatedWithOffset	TIMESTAMP
YEAR	VARCHAR
MONTH	VARCHAR
DAY	VARCHAR
name	VARCHAR
owner	VARCHAR
type	VARCHAR
c8y_IsDevice	BOOLEAN
c8y_IsDeviceGroup	BOOLEAN

The inventory collection keeps track of managed objects. Note that Cumulocity DataHub automatically filters out internal objects of the Cumulocity platform. These internal objects are also not returned when using the Cumulocity REST API. As described in [Configure inventory collection](#), pre-defined views over the inventory collection allow you to confine your offloading to the relevant data. Those views all share the above schema.

A managed object may change its state over time. The inventory collection also supports updates to incorporate these changes. Therefore an offloading pipeline for the inventory encompasses additional steps:

1. Offload those entries of the inventory collection that were added or updated since the last offload. They are offloaded with the above mentioned standard schema into the target table of the data lake.
2. Additional views over the target table are defined in the tenant's space in Dremio. Their names are defined as target table name plus `_all` and `_latest` respectively. The following examples use *inventory* as target table name:
 - **inventory_all** - A view with the updates between two offloading executions, not including the intermediate updates.
 - **inventory_latest** - A view with the latest status of all managed objects, with all previous transitions being discarded. For offloading configurations with view materialization enabled, the materialized state in the data lake is used.

The views are provided in your Dremio space. For details on views and spaces in Dremio, see [Refining offloaded Cumulocity data](#). In the main panel of the **Offloading** page you find in the details section of an offloading configuration links which navigate you to the corresponding table and views in the Dremio UI.

INFO

The fields **childDevices** and **childAssets** are not part of the default offloading columns. They were included in previous versions, but lead to problems for a high number of list items in those fields. In such a case, the columns were no more readable by Dremio. If they must be included in the offloaded data, they can be defined as additional result columns. However, you must ensure that the number of list items in those fields does not exceed the Dremio limit configured in your environment.

Offloading the measurements collection

The measurements collection stores device measurements. The corresponding table contains all measurements for a pre-selected measurement type. In the main panel of the **Offloading** page, you find a link in the details section of an offloading configuration that navigates you to the corresponding table in the Dremio UI.

You must select a measurement type, so that all offloaded data is of the same type. During offloading, the data of the measurements collection is flattened, with the resulting schema being defined as follows:

Column name	Column type
id	VARCHAR
creationTime	TIMESTAMP
creationTimeOffset	INTEGER
creationTimeWithOffset	TIMESTAMP
YEAR	VARCHAR
MONTH	VARCHAR
DAY	VARCHAR
time	TIMESTAMP
timeOffset	INTEGER
timeWithOffset	TIMESTAMP
source	VARCHAR
type	VARCHAR

Column name	Column type
fragment_name1.property_name1.value	Depends on data type, often FLOAT
fragment_name1.property_name1.unit	String
...	
fragment_nameM.property_nameN.value	Depends on data type, often FLOAT
fragment_nameM.property_nameN.unit	String
my_custom_property_name1	Depends on data type
...	
my_custom_property_nameN	Depends on data type

The entries in the measurements collection can have a different structure, depending on the types of data the corresponding device emits. While one sensor might emit temperature and humidity values, another sensor might emit pressure values. For details on measurement creation via API see the corresponding [Cumulocity REST API](#) documentation. See also [Mapping measurement fragments to relational data](#) for details on how a measurement fragment is mapped into a relational structure.

Each measurement document must have the ID of the associated source, a measurement type, and the measurement time. Within the document, there are one or more fragments. Each fragment comprises related measurements, with each measurement being modelled as a property. For example, the fragment `c8y_Steam` contains the properties `Temperature` and `Humidity`. Such a measurement property must itself contain a mandatory property `value` and should contain an optional property `unit`. A measurement document with one fragment having one measurement property is flattened in the data lake into a column `fragment_name.property_name.value` and, if set, a column `fragment_name.property_name.unit`. Documents with multiple fragments, each containing multiple measurements, are flattened in an analogous way, indicated in the above table with `fragment_name1.property_name1.value` to `fragment_nameM.property_nameN.value`.

Example

The following excerpt of a measurement document in the base collection is processed as follows:

```
{
  "id": "4711",
  ...
  "time": "2020-03-19T00:00:00.000Z",
  "type": "temperatureMeasurement",
  "c8y_Steam": {
    "Temperature": {
      "unit": "C",
      "value": 2.079
    },
    "Humidity": {
      "unit": "%RH",
      "value": 13.37
    }
  }
}
```

The fragment `c8y_Steam` is flattened into two measurements and represented in the target table in the data lake as

```
| ... | c8y_Steam.Temperature.unit | c8y_Steam.Temperature.value | c8y_Steam.Humidity.unit | c8y_Steam.Humidity.value | ... | — | — |
| — | — | | ... | C | 2.079 | %RH | 13.37 | ... |
```

! IMPORTANT

Try to ensure that the data you feed into the measurements base collection is consistent. If measurements of the same type vary in the fragment structures, the resulting target table might not have the expected schema. A common problem, for example, are varying data types of the values like one value being 2.079 and another one NaN.

MANAGING OFFLOADING JOBS

The following steps describe how to start and manage an offloading pipeline.

SCHEDULING AN OFFLOADING JOB

Once you have defined an offloading configuration and saved it, you can start the offloading pipeline.

Starting periodic offloading

Click the **Active** toggle in an offloading configuration to activate the periodic execution of the offloading pipeline, if it was not already activated when configuring the pipeline. The scheduler component of Cumulocity DataHub will then periodically trigger the pipeline.

The initial offload denotes the first execution of an offloading pipeline. While subsequent executions only offload data increments, the initial offload moves all collection data from the Operational Store of Cumulocity to the data lake. Thus, the initial offload may need to deal with vast amounts of data. For this reason, the initial offload does not process one big data set, but instead partitions the data into batches and processes the batches. If the initial offload fails, for example due to a data lake outage, the next offload checks which batches were already completed and continues with those not yet completed.

If the same pipeline has already been started and stopped in the past, a new start of the pipeline opens a dialog asking you whether you want to flush the existing data or append the data to the existing data. The latter option offloads only data that has been added after the last execution. The first option flushes the data lake. Then the next execution will offload the complete collection.

CAUTION

The option to flush already offloaded data in the data lake should be used with caution as data deleted in the data lake cannot be recovered.

Before restarting the periodic offloading, you may have changed the result schema by adding or removing columns (via adding or removing additional result columns). When you restart the pipeline, existing data in the data lake is not modified, but the new data being offloaded incorporates the new schema. When querying such a data set comprising different schemas, the system computes a merged schema and (virtually) fills it up with null values where fields have not yet been provided. This usually works without problems if additional attributes are included or removed from the offloading configuration. However, schema merging might fail or lead to unexpected behavior in certain cases. One example is if you change data types, for example, if the old configuration contained "myCustomField1" as a string and you change it to a number via "CAST(myCustomField1 AS Integer) AS myCustomField1". Therefore you should take care that the data you offload is consistent.

A previous offloading pipeline may have already written into the same target table, that is, the data is stored in the same folder on the data lake. In this case, when starting the new offloading pipeline, you are asked whether you want to flush the existing data or append the data to the existing data. You should only append the data if old and new data share the same schema. Otherwise, you might end up with a table consisting of disparate data, which hinders meaningful analysis. If the new data differs from the old data, you should use a new target table. Alternatively, you can flush the existing table if its old content is not needed anymore. Again, you should be careful when flushing a table as the data most likely cannot be recovered.

Scheduling settings

The scheduler is configured per default to run the offloading pipeline once an hour. The precise minute of the hour at which the offloading starts is assigned by the system to balance the load on the Operational Store of Cumulocity, that is, to avoid that all offloading jobs from different tenants run at the same time.

Stopping periodic offloading

Use the **Active** toggle in an offloading configuration to stop the periodic offloading. Then the scheduler stops scheduling new jobs; currently running jobs will complete.

MANAGING AN OFFLOADING PIPELINE

In the context menu of each offloading pipeline, you will find actions for managing and monitoring the pipeline.

Editing/showing an offloading pipeline

Click **Edit** to edit the current settings. Only inactive pipelines can be edited. Note that you cannot change the Cumulocity base collection selected for this pipeline. Additional filter predicates and additional result columns can be changed.

Note that these changes are not applied to already exported data. A change to the offloading pipeline only affects data to be exported in upcoming offloading runs.

In case of an inventory offloading, the view can be changed as well, which also affects the additional result column settings. If you change a view and you have selected additional columns in the current view, the system checks whether the target view contains these columns. If not, a dialog opens where you can either select or deselect each column missing in the target view. If selected, the system automatically creates the missing column in the target view. If deselected, the column will be removed from the list of selected additional columns. A special case is if the selected column exists in both views, but with different types. When selecting the column, the type of the current view will be used to update the column type of the target view.

For active pipelines, click **Show** to browse through the configuration. You cannot edit the settings.

Copying an offloading pipeline

Click **Copy** to copy the current configuration. The new configuration is an identical copy of the selected configuration except for the task name and the target table, both of which will have a unique suffix appended. You can change the settings according to your needs.

Deleting an offloading pipeline

Click **Delete** to delete a configuration. Only inactive pipelines can be deleted. Data in the data lake which has already been exported by this offloading pipeline is not deleted. To delete the actual data in your data lake, you must use the tooling offered by the data lake provider such as AWS S3 Console or Azure Storage Explorer.

Triggering a manual offloading job

If the periodic offloading is enabled, you can also manually trigger an offloading job between two scheduled executions. For example, you might not want to wait for the next scheduled execution to offload recent data into the data lake. Click **Offload now** to trigger a manual offloading. As with periodic offloading, a manual offloading execution processes only incremental data that has been added since the last offloading execution (independent of whether this last execution was triggered manually or by the scheduler).

However, we recommend you to rely on the periodic offloading instead of triggering it manually.

Monitoring an offloading pipeline

Click **Show offloading history** to examine the execution history of a pipeline. See [Monitoring offloading jobs](#) for details.

IMPORTING/EXPORTING OFFLOADING CONFIGURATIONS

The import/export functionality allows you to backup your offloading configurations to a file. You can use the backup when editing the data lake settings or to copy offloading configurations from one Cumulocity DataHub instance to another. Import/export includes the configuration settings; it includes neither the runtime status of an offloading pipeline nor already exported data.

Export of offloading configurations

The action bar provides an **Export** button, which exports all offloading configurations and manually added collection columns. The button is disabled if no offloading configurations are defined. If you click **Export**, all offloading settings are exported into a file. The file is located in the local download folder used by your browser.

CAUTION

You must not modify the contents of the export file as this might corrupt the import step.

Import of offloading configurations

The action bar provides an **Import** button, which imports offloading configurations from a file with previously exported configurations.

Click **Import** to open the import dialog. Either drop the file in the import canvas or click into the canvas to browse your file system to select the import file. Once the file is selected, a table with all configurations in the file is shown.

For each offloading configuration, the table lists the task name, the internal ID of the original configuration, the target table name, and the description. The **Status** column indicates whether an offloading configuration can be imported. If it is green, the configuration is valid and can be imported. If it is yellow, the configuration can be imported, but some of its settings are ignored as they are not supported by the tenant. If it is red, the configuration duplicates an existing configuration and therefore cannot be imported. It is a duplicate if an existing configuration has the same target table name or the same internal ID. The **Import** column provides checkboxes to select the configurations which are to be imported.

A global checkbox can be used to activate or deactivate all imported configurations after the import process has completed.

In addition to the offloading configurations, also the manually added collection columns referred by the offloadings are considered. For each of those collection columns, one of the following cases can occur:

- The collection column defined in the export file already exists with the same type in the collection in the target environment.
- The collection column defined in the export file does not exist in the collection in the target environment. In this case the column will be automatically created during the import process.
- The collection column defined in the export file already exists with a different type in the collection in the target environment. In this case the column in the target environment will be updated with the type from the export file during the import process.

To import the selected configurations, click **Import**. Click **Cancel** to cancel the import process.

For the specific case of inventory offloadings, their definition may not yet be based on views as described in [Configure inventory collection](#). When importing such an offloading, it will be configured so that it still reads directly from the inventory collection. It is advisable, however, to change the configuration and use a view instead in order to ensure that only relevant data is offloaded.

MONITORING OFFLOADING JOBS

Once you have configured and started your offloading pipelines, they regularly offload data to the data lake. The Cumulocity DataHub UI provides insights into the execution status of the different pipelines so that you can investigate whether everything is running as expected. For the case of offloading failures, you can also configure the offloading pipeline to raise an alarm as described in [Raising alarms](#).

An overview of the offloading status is shown on the [Home screen](#), while the details are available in the job history.

✔ REQUIREMENTS

You need administration or management permissions to access the offloading job histories. See [Defining Cumulocity DataHub permissions and roles](#) for details.



HISTORY PER OFFLOADING PIPELINE

If you want to examine the execution history for a particular pipeline, select **Offloading** in the navigation bar and select the one you are interested in.

Click **Show offloading history** in the context menu of the offloading configuration to show the history of offloading executions.

The list shows the execution history, with each execution consisting of the following details:

Component	Description
Status icon	The status of the execution, which is either running, successful, or error

Component	Description
Execution mode icon	The type of execution, which is either <i>scheduled</i> (calendar icon ) or <i>manual</i> (user icon )
Records	The number of records which have been offloaded during this execution
Execution time	The point in time the execution was started
Runtime	The execution runtime of the offloading run
Next execution time	The point in time for which the next execution is scheduled, provided offloading is activated; for a manual execution it is empty

The system is configured to keep a limited history of the last job executions.

Click **Reload** to refresh the list.

You can filter the entries by their status or timestamp by using the filter controls at the top. Click **Apply** to filter entries with the current filter settings. Click **Reset filter** to reset the current filter settings. Per default the entries from the last seven days are shown.

The page navigation buttons at the bottom can be used to traverse the history entries.

DETAILS OF OFFLOADING JOB

For a given offloading job, you can examine additional details of its execution.

In the corresponding list of jobs click the job you are specifically interested in. A details view encompasses the following information:

Schedule

Component	Description
Runtime	The execution runtime of the offloading run
Execution mode	The mode of the execution, which is either <i>manual</i> or <i>scheduled</i>
Start time	The point in time the execution was started
End time	The point in time the execution has ended
Scheduled execution time	The point in time for which the execution was scheduled
Previous execution time	The point in time the previous execution was started; for a manual execution it is empty
Next execution time	The point in time for which the next execution is scheduled, provided offloading is activated; for a manual execution it is empty

Results

Component	Description
Records	The number of records which have been offloaded during this execution

Job details

Component	Description
Job name	The name of the pipeline
Job ID	The internal ID of the job
Job execution ID	The Dremio ID of this execution
Source collection	The name of the Cumulocity base collection
Target table	The folder name in the data lake
Target folder	The path to the target table in the data lake
First run	Indicates whether the execution was the first run for this offloading pipeline
Data model	The data model, which is either <i>Time series</i> or <i>Standard</i> , used for a measurements offloading; only available for measurement pipelines

Offloading results

During offloading Dremio organizes the data in newly created files within the data lake, following a temporal folder hierarchy. For each of those files the following information is provided:

Component	Description
File size	The size of the file
Fragment	The hierarchical ID of the fragment
Partition	The partition with which the file is associated
Path	The path to the file in the data lake
Records	The number of records stored in the file

MONITORING COMPACTION JOBS

During offloading, data from the Operational Store of Cumulocity is written into files in the data lake. In order to ensure a compact physical layout of those files, Cumulocity DataHub automatically runs periodic compaction jobs in the background. For each offloading pipeline, a corresponding compaction job is set up and scheduled. Cumulocity DataHub UI provides insights into the compaction status of the different pipelines so that you can investigate whether everything is running as expected. For the case of compaction failures, you can also configure the offloading pipeline to raise an alarm as described in [Raising alarms](#).

✔ REQUIREMENTS

You need administration permissions to access the compaction job histories. See [Defining Cumulocity DataHub permissions and roles](#) for details.

STATUS OF ALL COMPACTION JOBS

In the navigator, select **Compaction status** under **Administration** to get an overview of the latest status of the compaction jobs for each pipeline. The list shows the corresponding last compaction job for all pipelines. Each compaction consists of the following details:

Component	Description
Status icon	The status of the execution, which is either running, successful, or failed
Execution time	The point in time the execution was started
Runtime	The runtime of the execution
Next execution time	The point in time for which the next execution is scheduled

Click **Reload** to refresh the status being shown.

You can filter the entries by their status by using the filter buttons in the action bar. The pagination buttons can be used to traverse the history entries.

HISTORY OF COMPACTIONS PER OFFLOADING PIPELINE

If you want to examine the compaction history for a particular offloading pipeline, select **Offloading** in the navigation bar and select the offloading job you are interested in.

Click **Show compaction history** in the context menu of the offloading configuration to show the compaction history.

The list shows the execution history with each execution consisting of the following details:

Component	Description
Status icon	The status of the execution, which is either running, successful, or failed
Execution time	The point in time the execution was started
Runtime	The runtime of the execution
Next execution time	The point in time for which the next execution is scheduled

The system is configured to keep a limited history of the last compaction jobs.

Click **Reload** to refresh the list.

You can filter the entries by their status or timestamp by using the filter controls at the top. Click **Apply** to filter entries with the current filter settings. Click **Reset filter** to reset the current filter settings. Per default the entries from the last seven days are shown.

DETAILS OF COMPACTION JOB

For a given compaction job, you can examine additional details of its execution.

In the corresponding list of jobs click the job you are specifically interested in. A details view encompasses the following information:

Schedule

Component	Description
Runtime	The runtime of the execution
Start time	The point in time the execution was started

Component	Description
End time	The point in time the execution has ended
Scheduled execution time	The point in time for which the execution was scheduled
Next execution time	The point in time for which the next execution is scheduled

Job details

Component	Description
Job name	The name of the pipeline
Job ID	The internal ID of the job
Job execution ID	The Dremio ID of this execution
Target table	The folder name in the data lake
Target folder	The path to the target table in the data lake
Daily run	Indicates whether the job is a daily execution job
Monthly run	Indicates whether the job is a monthly execution job
Recovery executed	Indicates whether the job has run a recovery from a job that has previously failed

Daily compaction results

During daily compaction the files are merged, following a temporal hierarchy. As a result, a folder for each day of the month is built with one or more file(s) combining all values measured for this day. For each of those files the following information is provided:

Component	Description
File size	The size of the file
Fragment	The hierarchical ID of the fragment
Partition	The partition with which the file is associated
Path	The path to the file in the data lake
Records	The number of records stored in the file

Monthly compaction results

During monthly compaction the files are merged, following a temporal hierarchy. As a result, a folder for each month is built with one or more file(s) combining all values measured for this month. For each of those files the following information is provided:

Component	Description
File size	The size of the file
Fragment	The hierarchical ID of the fragment

Component	Description
Partition	The partition with which the file is associated
Path	The path to the file in the data lake
Records	The number of records stored in the file

QUERYING OFFLOADED CUMULOCITY DATA

Cumulocity DataHub offers an SQL interface so that you can efficiently query offloaded device data and leverage the results in your own applications. A prerequisite for running SQL queries over device data is that you have configured and executed offloading pipelines that replicate and transform data from the Operational Store of Cumulocity to the data lake.

OVERVIEW

As described in [Cumulocity DataHub at a glance](#), Cumulocity DataHub manages offloading pipelines which periodically extract data from the Operational Store of Cumulocity, transform the data into a relational format, and finally store it in a data lake. Instead of querying the Operational Store, you run your queries against the data lake. The distributed SQL engine Dremio provides the query interfaces to access the data lake.

Different standard interfaces exist for that purpose, namely JDBC, ODBC, and REST. You can also use the Dremio UI. In order to work with one of those interfaces, select **Home** in the navigation bar. Under **Quick links** you will find starting points for the different interfaces.

ACCESS TO DATA LAKE CONTENTS

You need a separate Dremio account to run SQL queries. The Dremio account is required to authenticate your requests when running queries against the data lake using Dremio. In the initial configuration step, a corresponding Dremio user has been created. Contact the administrator for the Dremio account settings.

When you have established a connection, you can run SQL queries against your tables in the data lake (to which new data is appended whenever the offloading pipeline has successfully run). The source you refer to in the query is defined by your tenant ID and the target table you have specified in the offloading configuration. The identifier to be used as the source in a SQL query is defined as follows for the different data lake providers:

- **Azure Storage:** YourTenantIdDataLake. **FileSystem** .YourAccountName.TargetTableName with **FileSystem** denoting the file system within your Azure Storage account
- **Amazon S3:** YourTenantIdDataLake. **Bucket** .YourAccountName.TargetTableName with **Bucket** denoting the bucket within your Amazon S3 account
- **Others:** YourTenantIdDataLake.YourAccountName.TargetTableName

For example, if your tenant ID is **t47110815** and you have defined an offloading configuration to write the alarms collection to the target table **JohnsAlarms** in an Azure Storage account containing a file system named **Dremio**, then an example query would be:

```
SELECT * FROM t47110815DataLake.Dremio.t47110815.JohnsAlarms;
```

You can look up the paths to the tables in Dremio's UI. Click on your data lake under "Sources" at the left, then navigate to the table in the right canvas. When you hover over the table name, a small "copy" icon with the tool tip "Copy Path" will appear right of the table name. Clicking on it will copy the table name into your clipboard.

INFO

The offloading pipeline must be executed at least once with corresponding data being offloaded before you can run a query.

GETTING DATA LAKE SCHEMA INFORMATION

Each table in the data lake is associated with an offloading pipeline. The schema of the table depends on the configuration of the offloading pipeline. It comprises the schema of the base collection for which the pipeline is configured as well as optionally configured additional result columns. In [Offloading Cumulocity base collections](#) you will find the default schema per base collection. In order to get the overall schema of the table you have different options:

- Navigate to the offloading page providing an overview of all offloadings. In the context menu of the corresponding offloading pipeline, select **Edit** or **Show**. Navigate to the final step of the wizard, which provides the overall schema plus sample data. Prerequisite for that procedure is that you have the administrator or manager role as specified in section [Defining Cumulocity DataHub permissions and roles](#). Also the offloading pipeline feeding data into the according table in the data lake must still exist.
- Log into the Dremio UI with the aforementioned Dremio account. Click on your data lake under "Sources" at the left, then navigate to the table in the right canvas. Hover over the target table to get the schema information.
- Run the following SQL query, with the table path adapted accordingly, to get the schema as well as sample data, using below listed connectivity options:

```
SELECT * FROM t47110815DataLake.Dremio.t47110815.JohnsAlarms LIMIT 5;
```

USING THE DREMIO UI

You can use the Dremio UI to interactively run queries against the data lake. See [Refining offloaded Cumulocity data](#) for more details.

CONNECTING VIA JDBC

If you have a Java client, you can use JDBC to run SQL queries against the data lake. You must download the [Dremio JDBC driver](#). You can obtain the JDBC connection string and the required driver version from Cumulocity DataHub by clicking the **JDBC** icon in the **Quick links** section of the **Home** page. When setting up your JDBC client, use as username and password the credentials from your Dremio account.

CONNECTING VIA ODBC

If you want to use an ODBC client to run SQL queries against the data lake, you must configure the platform-specific driver, following the associated [Dremio installation instructions](#). To obtain the ODBC connection string, click the **ODBC** icon in the **Quick links** section of the **Home** page. When setting up your ODBC client use as username and password the credentials from your Dremio account.

CONNECTING VIA DREMIO REST API

Dremio offers an [SQL REST API](#) which you can use to run SQL queries against tables in the data lake. You must authenticate with your Dremio account against Dremio in order to use the API.

Note that the API might change any time and no guarantees are provided. Dremio does not send any CORS headers, so direct access from a browser-based application is not possible. It is highly recommended to use Cumulocity DataHub's REST API, see below.

CONNECTING VIA CUMULOCITY DATAHUB REST API

The Cumulocity DataHub server also can handle REST requests for Dremio query processing, serving as a proxy to Dremio. Cumulocity DataHub offers two REST APIs for running queries against Dremio. The standard REST API for small to moderate query result sizes and a high-performance REST API for large query result sizes. See the [Cumulocity DataHub REST API documentation](#) in the Cumulocity OpenAPI Specification for details on the endpoints. When using this API, you authenticate with your Cumulocity account, not with your Dremio account.

CONNECTING OTHER CLIENTS

Dremio offers support for connecting a variety of clients, including reporting tools like PowerBI and common analytics languages like Python. The [Dremio documentation](#) discusses how to connect these clients to Dremio and leverage its query capabilities.

See also [Integrating Cumulocity DataHub with other products](#) to learn how other products can connect to Cumulocity DataHub and leverage its query capabilities.

REFINING OFFLOADED CUMULOCITY DATA

In addition to SQL querying using standard interfaces, you can utilize Dremio functionality to further refine and curate your offloaded data.

For a detailed description of all functionalities Dremio provides you can consult the [Dremio documentation](#).

ACCESSING AND LOGGING INTO DREMIO

You access Dremio via a web browser. It has been tested with the following web browsers:

- Firefox (latest version)
- Chrome (latest version)

INFO

Support for mobile devices like smartphones or tablets has not been tested.

To access Dremio, navigate to the home page. Under **Quick links** click on the **Dremio** icon. This will direct you to the **Login** screen of Dremio.

INFO

Your Dremio user does not have administration rights in Dremio.

How to log into Dremio

On the **Login** screen, enter your Dremio account credentials. Click **Login** to enter Dremio.

When you log in successfully, you will be taken to the home page of Dremio.

When you want to log out, click on your username and select **Log out**.

SOURCES AND SPACES

On the home page of Dremio you will find at the left under **Datasets** two panels called **Spaces** and **Sources**.

Sources

In the **Sources** panel there is the data source **YourTenantIdDataLake** , for example, **t47110815DataLake** . This source has been auto-configured for you and points to your data lake.

INFO

Terminology-wise, Cumulocity DataHub replicates data from the Operational Store of Cumulocity into the data lake. For Dremio the data lake, comprising the target tables, is a data source as it allows reading data from it.

When you click on your data source it will be shown in the main panel. Clicking on the source in the main panel navigates into the data source. Here, you see a list of all target tables of your offloading pipelines. Clicking one of these target tables opens an SQL editor which

allows you to run queries against that target table.

INFO

You might also see a folder named `c8y_cdh_temp`. The folder is used for Cumulocity DataHub internal purposes and must not be deleted or altered.

Spaces

A space in Dremio helps in organizing your data sets. Cumulocity DataHub auto-configures a space which is named `YourTenantIdSpace`, for example, `t47110815Space`. A dataset in the space is referred to in queries as `YourTenantIdSpace.YourDataset`. As described in [Offloading Cumulocity base collections](#), for the inventory, events, and alarms collections there are preconfigured views providing either all or latest data.

Job history

The **Job History** tab displays jobs/queries you have executed. It allows you to view details of a job and offers filter capabilities (time range, job status, query type, and queue). The **Profile** view inside the job detail view is very useful to investigate optimization potentials in your queries.

INFO

The job history only contains queries that you have actively triggered; the jobs related to the data extraction are hidden.

CREATING VIEWS

With Cumulocity DataHub, you can replicate data from a Cumulocity collection to a data lake using a default transformation of the data. As requirements for subsequent data analysis of the offloaded device data may vary over time, you should configure your offloading pipeline so that all potentially relevant data is included.

Depending on your use cases, you will often find the need to provide a view on the data, which limits, filters, or transforms the data, such as converting Celsius to Fahrenheit or extracting data from JSON fragments.

In Dremio, you can create such a view by defining a corresponding query and saving it as a new dataset. When saving that new dataset, you must select your space as the location and can freely select a name for the view. Once that is done, you can work with the new dataset as with any other source and run queries against it. This includes in particular querying this view from other clients as described in [Querying offloaded Cumulocity data](#).

INFO

Such a view is per default not materialized, that is, it is not stored persistently. Each time you query the view, the underlying query defining the view is run against the source data. For the pre-defined alarms/events/inventory views, you can optionally activate view materialization during the offloading configuration.

Example

Consider the case that you want to visualize data in a reporting tool. The raw data volume is too high, so you want to instead show the hourly average of the column `myValue`. You can easily do that by creating a view with the following SQL statement and saving it as a view/virtual data set:

```
SELECT DATE_TRUNC('HOUR', "time") AS "time", AVG(myValue) AS hourlyAvg
FROM myTable
GROUP BY DATE_TRUNC('HOUR', "time")
```

The creation (and update) of views can be done via the Dremio SQL API, too. This is especially useful to automate tasks. The above example can be created or updated as follows.

```
CREATE OR REPLACE VDS YourTenantIdSpace.YourDesiredViewName AS
SELECT DATE_TRUNC('HOUR', "time") AS "time", AVG(myValue) AS hourlyAvg
FROM myTable
GROUP BY DATE_TRUNC('HOUR', "time")
```

JOINING TABLES/VIEWS

Views you have defined and target tables from your data lake can be joined as well. In Dremio you can either define joins using the SQL editor or a graphical interface.

A general use case for joining is to enrich your alarms, events, or measurement values with metadata from the inventory collection, for example:

```
SELECT *
FROM t47110815DataLake.Dremio.t47110815.alarms
JOIN t47110815DataLake.Dremio.t47110815.inventory
USING(id)
```

CUMULOCITY DATAHUB BEST PRACTICES

Learn from well-established usage patterns in order to ensure a robust and scalable processing of your SQL queries.

NAMING POLICIES

When defining an offloading configuration, you must specify the task name, the target table, and a description. You should ensure that you provide reasonable names for each of these settings so that afterwards you can easily find the offloading pipelines you are interested in. A reasonable naming scheme also facilitates writing queries.

Also when defining an offloading configuration, you must always define a target table that is unique among the currently saved configurations. You should not re-use a target table from an old offloading configuration which was deleted in the meantime. Otherwise, you might run into the problem that your target table consists of data from multiple configurations with potentially different schemas.

CAREFUL DEFINITION OF ADDITIONAL COLUMNS AND FILTER PREDICATE

An offloading configuration allows you to specify additional columns to be offloaded as well as filter predicates for filtering the data. For both settings, you should carefully think about which data you actually need for your processing. Data being filtered out cannot be retrieved any more. Even if you adapt the filter predicate afterwards, the data which would have qualified in previous offloading executions will not be offloaded. You can, however, stop an offloading, change the configuration to include additional fields, and so on, and then restart it. When it is restarted, Cumulocity DataHub will ask you whether you want to flush existing data or append. Flushing will delete all data in the data lake so that with the next offloading execution the complete collection will be offloaded again. Note that Cumulocity DataHub can only import data that is still present in the Operational Store of Cumulocity, that means, be careful with this option and keep in mind that data retention policies in Cumulocity might have deleted old data. On the other side, data which will definitely be irrelevant for further analysis should not be included in the offloading process.

DECOMPOSITION OF COMPLEX ADDITIONAL COLUMNS

You may have stored complex data using JSON fragments as an additional column in a Cumulocity collection. As described in [Configuring offloading jobs](#), you can add additional columns so that they are included in the offloading process. Within these additional columns, you can apply functions to decompose the fragments into flat data.

Writing these functions is often an iterative process that requires multiple adaptations of the underlying logic. Leverage the Dremio SQL editor and define a dummy offloading configuration which moves a small portion of the data into the data lake for testing purposes. You can use the filter predicate to retrieve such a portion of the data; see below for time filter examples. Then you can open the table created by the offloading configuration with Dremio; using Dremio's SQL editor, you can develop the extraction logic. When your decomposition logic for your additional columns is complete, you can copy the column transformations and use them to define a corresponding offloading configuration in Cumulocity DataHub. Once that is done, the dummy offloading pipeline can be deleted.

EXAMPLES FOR ADDITIONAL FILTER PREDICATES

When defining an offloading configuration, you can define an additional filter predicate to filter out unwanted entries in the offloading process.

Examples for time filters

Depending on the collection, you can use different time filters. All collections support `creationTime` which represents the timestamp when the entity was persisted by the platform (UTC timezone). Mutable entities (alarms, events, inventory) also support `lastUpdated` which is the timestamp when the entity was last changed (UTC timezone). `time` is the application timestamp written by the client; it is supported in alarms, events, and measurements.

⚠ CAUTION

Dremio offers functions like `CURRENT_TIME()` to determine the current time or date the system has when executing the query. You should try to avoid filter logic depending on such dynamic data as the offloading results are inherently non-deterministic.

The below time filters are examples only; you can use much more complex or simpler combinations with a mixture of AND/OR-connected conditions.

Alarms/events To offload all alarms or events which have the application time set to between 2020-02-08 14:00:00.000 and 2020-02-08 15:00:00.000, use:

```
src."time"."date" >= {ts '2020-02-08 14:00:00.000'} AND
src."time"."date" <= {ts '2020-02-08 15:00:00.000'}
```

To offload all alarms or events which were persisted after 2020-02-08 15:00:00.000, use:

```
src."creationTime"."date" > {ts '2020-02-08 15:00:00.000'}
```

Restricting the offloading to alarms and events last modified before 2020-02-08 14:00:00.000, use:

```
src."lastUpdated"."date" < {ts '2020-02-08 14:00:00.000'}
```

Inventory To offload all data which was persisted between 2020-02-08 14:00:00.000 and 2020-02-08 15:00:00.000, use:

```
src."creationTime"."date" >= {ts '2020-02-08 14:00:00.000'} AND
src."creationTime"."date" <= {ts '2020-02-08 15:00:00.000'}
```

or, to offload all data that was last updated after 2020-02-08 14:00:00.000, use:

```
src."lastUpdated"."date" > {ts '2020-02-08 14:00:00.000'}
```

Measurements To offload all data with application time between 2020-02-08 14:00:00.000 and 2020-02-08 15:00:00.000, use:

```
src."time"."date" >= {ts '2020-02-08 14:00:00.000'} AND
src."time"."date" <= {ts '2020-02-08 15:00:00.000'}
```

or, to offload all data received by the platform after 2020-02-08 14:00:00.000, use:

```
src."creationTime"."date" > {ts '2020-02-08 14:00:00.000'}
```

Examples for alarms filters

To filter by current critical alarms, use:

```
status != 'CLEARED' AND severity = 'CRITICAL'
```

Examples for events filters

To filter by position, provided the field c8y_Position exists, use:

```
src.c8y_Position.lat > 49.8146 AND src.c8y_Position.lat > 8.6372
```

To filter by text, use:

```
text LIKE '%Location updated%'
```

Examples for inventory filters

To limit devices to a specific firmware version, provided the field c8y_Firmware exists, use:

```
src.c8y_Firmware.version='v1.32'
```

To limit the offloaded inventory objects to devices, use:

```
convert_from(convert_to("__fragments", 'JSON'), 'UTF8') LIKE '%"c8y_IsDevice"%'
```

QUERYING ADDITIONAL DATA WITH CUMULOCITY DATAHUB

Main use case of Cumulocity DataHub is to offload data from the internal Cumulocity database to a data lake and query the data lake contents afterwards. In some use cases, Cumulocity DataHub is required to query additional data which is not kept in the Cumulocity platform. For a cloud environment, the additional data must be provided as Parquet files and must be located in the data lake as configured in the initial configuration of Cumulocity DataHub. The Parquet files must not be stored in folders that are used as targets for offloadings as this could corrupt offloading pipelines of Cumulocity DataHub (if the schema doesn't match with the schema of the Parquet files created via offloading jobs). In addition, the Parquet files must be compliant with the [Dremio limitations for Parquet files](#).

For a dedicated environment, the additional data can be located somewhere else, provided it can be accessed via Dremio, for example, in a relational database. For performance and cost reasons, however, data and processing should always be co-located.

If you want to combine your offloaded IoT data with the new, additional data, you can define a join query in Dremio and store the query as a view. The view can then be queried like any other table in Dremio and provides the combined data.

MODIFYING DATA IN THE DATA LAKE

Cumulocity DataHub offloads IoT data into a data lake. Within this process, the original data is transformed into a relational format and finally stored in files, using the Apache Parquet format. Each offloading configuration has a unique folder in the data lake, which is referred to as target table. The files in that folder are organized in a folder hierarchy based on temporal information. In order to ensure an efficient and in particular correct querying of the data, the files in the data lake must not be modified.

In rare cases, however, the modification of the files is required. You might either want to drop old data or delete one or more columns. First step is to stop and delete the offloading configuration associated with this target table. Next, you can either externally rewrite the

data or use Dremio's [CTAS query](#) feature to query the data from this target table, filter obsolete data, and use projections to skip unwanted columns. The query results must be written to a new folder of the data lake. To make the folder accessible as a table in Dremio, promote the folder to a dataset. Then, you can delete the old folder from the data lake. Next, you must define a new offloading configuration which uses the new folder name as target table name. Define the same filter criteria and columns as in the CTAS query to ensure that the data has the same format. Save the offloading configuration, select the append mode for storing the data, and activate the offloading.

It must be emphasized that this use case is not officially supported by Cumulocity DataHub. While there is nothing wrong in particular with rewriting data (or importing legacy data this way), there is a high risk that the manually created files are not compliant to the required table schema and thus offloading additional data to the target table would not work, that means, the corresponding offloading pipeline would be broken. In particular, one must be careful to use the correct data type (`TIMESTAMPMILLIS`) for all timestamp columns (`time` , `timeWithOffset` , `creationTime` , `creationTimeWithOffset`).

OPERATING CUMULOCITY DATAHUB

This section describes how you can access system information, usage statistics, and audit logs.

CHECKING SYSTEM INFORMATION

✔ REQUIREMENTS

You need administration permissions to access system information. See [Defining Cumulocity DataHub permissions and roles](#) for details.

In the navigator, select **Administration** and then **System status** to get information about the system configuration and its status.

Under **Microservice** you will find the status of the microservice, which is either marked as green or red. This status reflects whether the microservice can be accessed from the web application. If the microservice is accessible, its current version is shown. If not, check the status of the microservice and its logs as described in [Managing applications](#).

Under **Web application** you will find the version of the web application.

Under **Dremio** you will find the status of Dremio, which is either marked as green or red. This status reflects whether Dremio can be accessed from the microservice. If Dremio is accessible, its current version is shown. If not, check the status of the microservice and its logs as described in [Managing applications](#).

Under **Management** you will find the setup of the system. If you expand that box by clicking on the arrow to the right, all relevant system properties and their values are listed. Note that these values cannot be modified for a running microservice. The tenant administrator must redeploy the microservice with corresponding new values.

TRACKING USAGE STATISTICS

If enabled, Cumulocity DataHub tracks usage statistics on the amount of data being processed. These statistics are collected for offloading queries and track the amount of data these queries read from the Operational Store of Cumulocity. The statistics are also collected for ad-hoc queries and track the amount of data these queries read from the data lake. The usage statistics can be utilized for a volume-based charging. They can also be utilized to pinpoint resource-intensive queries in terms of network load.

i INFO

The tracking of usage statistics is supported for the Cumulocity DataHub Cloud edition. It is not supported for the Cumulocity DataHub Edge edition.

In the navigator, select **Administration** and then **Usage statistics** to view the usage statistics.

In the action bar, a date control allows you to select the month for which you want to see the usage statistics.

The three top panels show overall summary statistics as well as statistics separated for offloading and ad-hoc queries. If data from the month before the selected month is available, a tendency arrow illustrates whether the data volume of the selected month has decreased, increased, or stayed flat. The panels with the offloading and the ad-hoc query statistics additionally list the days with minimum/maximum volume as well as the daily average volume.

The table below the summary statistics shows the details on a per-day basis for the selected month. For each day, the volume offloaded and the volume queried are shown as well as their sum, which constitutes the daily volume. In addition the percentage of the monthly volume is shown, that is, how much did the daily volume contribute to the overall monthly volume. The date of each entry links to the [Query log](#), which lists all queries for the respective day.

INFO

The statistics are refreshed once per hour. Therefore, the statistics for the current month may not include the latest data. The statistics are deleted after a retention period, so for older months statistics may no longer be available.

VIEWING AUDIT LOGS

Auditing shows in the query log the queries being executed and in the system log the operations that users have carried out.

QUERY LOG

In the navigator, select **Query log** to view the query log.

REQUIREMENTS

The Cumulocity DataHub feature for storing query profiles must be enabled. The profiles are deleted after a retention period, so for older months profiles may no longer be available.

At the top of the page you can select either offload or ad-hoc queries, define a text filter on the offloading task/ad-hoc query string, and select a time period. Use the pagination buttons at the bottom of the page to navigate through the result list.

For each offloading query, the following information is provided:

Column name	Description
Offloading task	The task name of the offloading pipeline, complemented by a status icon showing success or failure of the pipeline execution
Runtime	The execution runtime of the Dremio queries related to the offloading run
Data scanned (MB)	The amount of data the offloading query has read from the Operational Store of Cumulocity
Data billed (MB)	The amount of data being billed (depending also on your contract); amounts of data less than 10 MB in an offloading query will be billed as if they were 10 MB
Details	The internal task UUID in an expandable box

For each ad-hoc query, the following information is provided:

Column name	Description
User	The username of the Dremio user, which has been used to execute the query
Query	The SQL query, complemented by a status icon showing success or failure of the query execution
Runtime	The execution runtime of the query
Data scanned (MB)	The amount of data the ad-hoc query has read from the data lake
Data billed (MB)	The amount of data being billed (depending also on your contract); amounts of data less than 10 MB in an ad-hoc query will be billed as if they were 10 MB
Details	The query string as well as a link to the associated Dremio job in an expandable box

SYSTEM LOG

Switch to the Administration application and navigate to **Accounts > Audit logs** to view the system-related [audit log entries](#) for Cumulocity DataHub. The log entries comprise, for example, information on offloading configurations, Dremio users, and initial configuration. At the top of the page, you can filter for those entries by selecting "DataHub" under **Type**. For each operation, two entries are provided in the audit log, indicating start and end of the operation.

For each log entry, the following information is provided:

Column name	Description
Device time	The point in time the user has carried out the operation
User	The user that has carried out the operation
Event	The type of operation and whether it has started or completed
Description	The details of the operation

ENDPOINTS FOR MONITORING

ETL PIPELINE HEALTH

The Cumulocity DataHub microservice exposes an endpoint to automatically monitor the health of active offloading jobs as well as compaction and data collection jobs. The health status can be monitored with the endpoint `GET /service/datahub/scheduler/health`. The endpoint accepts two optional parameters, **format** and **check**.

The parameter **format** determines the format of the response body. It supports the following values:

Value	Definition
text	Send the response body as plain text.
json	Send the response body as JSON.

If **format** is not set, the text option is used by default.

The parameter **check** defines which jobs are reported. The parameter supports the following values:

Value	Definition
ALL	All jobs are reported.
OFFLOADING	Only offloading jobs are reported. In corresponding messages such a job is also denoted as CTAS job.
COMPACTION	Only compaction jobs are reported.
DremioJobDetailPersistence_OFFLOADING	Only the job for collecting and persisting offloading usage data is reported.
DremioJobDetailPersistence_QUERY	Only the job for collecting and persisting usage data for ad-hoc queries is reported.
C8Y_BILLING_METRICS	Only the job for submitting usage data is reported.

If **check** is not set, all jobs except C8Y_BILLING_METRICS are reported.

The endpoint examines the latest job executions of qualified jobs and classifies them:

- If the job has failed, it is reported as CRITICAL.
- If the job is still running, it is categorized as follows:
 - If it is running for up to one hour, its health is classified as STEADY.
 - If it is running for up to six hours, its health is classified as WARNING.
 - If it is running for more than six hours, its health is classified as CRITICAL.
- If the job has succeeded, it is checked whether it was the last job that should have been run for this configuration. If there should have been a new run of this job and the system is already 10 minutes behind the scheduled execution time, the job is classified as CRITICAL. Otherwise, the job is classified as STEADY.

If all jobs are classified as STEADY, the endpoint returns the HTTP status code 200 with the following message:

```
"HTTP 200 CDHCBEI0029 - Scheduler healthcheck succeeded."
```

Otherwise, the endpoint returns the HTTP status code 500 with the following message:

```
"HTTP 500 CDHCBEE0031 - Scheduler healthcheck failed: There were failed or suspended jobExecutions."
```

The response body indicates the jobs to be checked by an administrator:

```
"There were failed or suspended jobExecutions:
CRITICAL: Job should already have been executed at 14:08:03.705: uuid=34391b71-abaa-477e-b870-2c32aa6ea790, jobType=CTAS,
jobRunId=CDHScheduler_9cd4309c-99d7-43ae-92f7-4f1d267faff71713875003234"
```

The endpoint can be accessed by any logged in Cumulocity user who is authorized to access the Cumulocity DataHub microservice.

MANAGING THE DATA LAKE

Cumulocity DataHub uses a data lake to store data being offloaded from the Cumulocity operational database. The data is organized in hierarchical folders, following a temporal hierarchy. Within the folders the offloaded data is organized in Parquet files. During the offloading process Cumulocity DataHub creates temporary Parquet files, holding intermediate data, which are deleted afterwards. In order to prevent data being spread over multiple small files, a compaction process is executed regularly, producing fewer, larger files.

! IMPORTANT

The contents and hierarchy of the data lake must not be modified. There is a high risk that data gets lost and subsequent querying of the data lake produces incomplete results. In general, you must also have a backup strategy in place for your data lake in order to properly handle situations causing data inconsistencies, corruption, or even data loss.

FOLDER STRUCTURE

The data within the data lake is organized hierarchically. Each offloading pipeline is associated with one target table. Each target table corresponds to a folder in the data lake with the same name. Such a folder consists of three different types of subfolders:

- Monthly/daily folder: The folder name starts with **monthly** or **daily** followed by the timespan of data managed within that folder. For example, `monthly_2024_01` contains all data from January 2024, while `daily_2024_01_15` contains all data from the 15th of January 2024.
- Initial offloading folder: When an offloading pipeline for the measurements collection offloads for the first time, all folders with data from this initial offloading is located in folders starting with **chunk**. Within a chunk folder data is also organized hierarchically with respect to years, months, and days, as encoded in the folder names. The chunk folders are optional.
- Internal folders: Folders starting with **incremental** contain internal information and must not be deleted.

EMPTY PARQUET FILES

Cumulocity DataHub may produce empty Parquet files in certain constellations, like an execution node crashing during a write process. If such empty files exist in the data lake, the initial configuration as well as offloading runs will fail. This requires interaction with the data lake. Cumulocity DataHub does not delete those empty files automatically. You must delete them manually using the tooling of your data lake provider, like AWS S3 Console or Azure Storage Explorer.

If the initial configuration has failed due to an empty Parquet file, the error message shown during the failed configuration attempt provides the details on the file. This includes the folder containing the empty Parquet file, like `c8y_cdh_temp/connectionTest`. You must delete the folder with all its sub-folders, including potential other non-empty Parquet files, to avoid inconsistencies caused by incomplete, partially written data.

If an offloading has failed, the associated error is shown in the job history, providing details on the empty Parquet file causing the error. You have to browse to the associated collection folder in the data lake, like `events` or `alarms`. Within that folder a couple of sub-folders can exist, starting with `incremental_`, `daily_`, `monthly_`, or `chunk_`. The error message gives you the corresponding folder in which that empty Parquet file is located, like `events/incremental_1694787385`. You must delete the folder with all its sub-folders. With the next offloading run, the corresponding time frame of data within that folder will be offloaded again, so that no data is lost. However, data might be lost if the data already moved out of the retention window in the operational database before a corresponding offloading was successfully executed.

INTEGRATING CUMULOCITY DATAHUB WITH OTHER PRODUCTS

INTEGRATING CUMULOCITY DATAHUB WITH MICROSOFT POWER BI

Microsoft Power BI is a business intelligence tool which allows you to create and use interactive reports for data from various sources. These reports can also be built on your IoT data. Given your devices are connected with the Cumulocity platform, you can utilize Cumulocity DataHub to offload the data into a data lake of your choice. Then you can create a Microsoft Power BI report which is based on the data in the data lake. Cumulocity DataHub allows you to access and work with these reports from within the Cumulocity DataHub web frontend.

PREREQUISITES

Before setting up the connection to Microsoft Power BI in Cumulocity DataHub, conduct the following steps.

Accessing data lakes in Microsoft Power BI reports

Cumulocity DataHub leverages the native interaction between Microsoft Power BI and Dremio. Microsoft Power BI reports can consume data from data lakes using Dremio as query and data access layer. When creating a new report in Microsoft Power BI desktop, you can select [Dremio as a database](#) and establish a connection to the Dremio cluster, using the ODBC connection settings as provided on the homepage of the Cumulocity DataHub UI. With this connection you have access to the data lakes connected to Dremio.

INFO

The Microsoft Power BI datasets should use the DirectQuery mode, which prevents replicating and caching the data from the data lake.

In contrast to versions prior to 10.18, it is no longer required to deploy a Microsoft Power BI gateway. A native connector from Power BI Web to Dremio is available now.

Configuring access to Microsoft Power BI reports

To make reports available in its web frontend, Cumulocity DataHub embeds Microsoft Power BI content. Users neither must sign in to Microsoft Power BI nor need a Microsoft Power BI license to access the reports. For access authentication an Azure Active Directory service principal object with an application secret is used.

The following configuration steps are required, as discussed in detail in the corresponding [Microsoft documentation](#).

As prerequisite you need an Azure Active Directory tenant. If you do not have an Azure Active Directory tenant, follow the instructions in the [Microsoft documentation](#).

Next you must register an Azure Active Directory application, which serves as service principal. You must configure the service principal application to access the REST APIs of Microsoft Power BI, following the instructions on the [Microsoft Power BI website](#):

1. Select **Embed for your customers**.
2. Sign in to Microsoft Power BI.
3. Register an application with respective permissions.
4. Skip creating a workspace and importing content.
5. Grant permissions to the service principal.

INFO

An application created with the wizard can be used as a service principal.

Alternatively, you can create a service principal application following the section *Creating an Azure AD app in the Microsoft Azure portal* in the [Microsoft documentation](#).

Additionally, you must add a client secret for the service principal application. You can do that via the [Azure portal](#). Search for **App registrations**, select your application by its name under **All applications**, and click the link next to the **Client credentials** entry on the **Overview** page of the application.

Next you can define a workspace to organize your reports. By adding the service principal application as a member or admin to the workspace, it can access the reports of the workspace. Go to the [Microsoft Power BI website](#) and conduct the following steps to grant the permissions:

1. Sign in to Microsoft Power BI.
2. Click **Workspaces**.
3. Select the context menu of the workspace to share with the service principal.
4. Select **Workspace access**.
5. Enter the name of your recently created service principal application and grant the *Member* or *Admin* permission.

Only workspaces granting access to the service principal application can be browsed from within Cumulocity DataHub. Once the workspace is available, you can publish reports to it and access it in Cumulocity DataHub.

SETTING UP THE CONNECTION IN CUMULOCITY DATAHUB

In the navigator, select **Settings** and then **Microsoft Power BI** to define the connection settings.

Settings	Description
Azure Active Directory tenant ID	The ID of the Azure Active Directory tenant. Within the tenant, an Azure Active Directory application must exist with a service principal that is allowed to access corresponding resources of Microsoft Power BI.
Client ID	The ID of the Azure Active Directory application which has permissions to call the REST APIs of Microsoft Power BI.
Client secret	The client secret, which is configured for the Azure Active Directory application.

Once all settings are completed, click **Save** on the action bar to save the settings and establish the connection.

If you want to delete the settings, click **Delete** on the action bar. You cannot access reports afterwards.

WORKING WITH REPORTS

Once the settings are defined, you can access and work with the reports.

1. In the navigator, select **Microsoft Power BI**. The menu entry is only shown if the connection settings are defined.
2. On the **Reports** page, click **Add report** in the action bar. A dialog opens with two dropdown boxes. The first dropdown box lists all workspaces which grant member or admin access to the service principal. Select the workspace you are interested in. The second dropdown box provides all reports of the selected workspace. Select a report from the dropdown box.
3. Click **Select** to open the report or **Cancel** to close the dialog without selecting a report.

The selected report is shown and can be interacted with. You can open multiple reports. For each opened report, a tab entry shows up in the action bar. To close the currently selected report, click **Remove report** in the action bar.

INFO

The list of currently opened reports is not stored permanently. When closing the browser, the list will be flushed. It will also be flushed if the settings are deleted.

[🏠](#) > [Cumulocity DataHub](#) > Integrating Cumulocity DataHub with other products