IBM watsonx.data v 2.0.2

Fix-Pack Release

Notices

Trademarks

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Contents

Chapter 1. IBM watsonx.data

Overview

IBM® watsonx.data is a new open architecture lakehouse that combines the elements of the data warehouse and data lakes. The best-in-class features and optimizations available on the watsonx.data make it an optimal choice for next generation data analytics and automation.

watsonx.data Developer edition

watsonx.data on Red Hat® OpenShift®

watsonx.data is a unique solution that allows co-existence of open source technologies and proprietary products. It offers a single platform where you can store the data or attach data sources for managing and analyzing your enterprise data.

Use watsonx.data to store any type of data (structured, semi-structured, and unstructured) and make that data accessible directly for Artificial Intelligence (AI) and Business Intelligence (BI). You can also attach your data sources to watsonx.data, which helps to reduce data duplication and cost of storing data in multiple places. It uses open data formats with APIs and machine learning libraries, making it easier for data scientists and data engineers to use the data. watsonx.data architecture enforces schema and data integrity, making it easier to implement robust data security and governance mechanisms.

Key features

- An architecture that fully separates compute, metadata, and storage to offer ultimate flexibility.
- Multiple engines such as Presto (Java) and Spark that provide fast, reliable, and efficient processing of big data at scale.
- Open formats for analytic data sets, allowing different engines to access and share the data at the same time.
- Data sharing between watsonx.data, Db2® Warehouse, and Netezza Performance Server or any other data management solution through common Iceberg table format support, connectors, and a shareable metadata store.
- Built-in governance that is compatible with existing solutions, including IBM Knowledge Catalog.
- Cost-effective, simple object storage is available across hybrid-cloud and multicloud environments.
- Integration with a robust ecosystem of IBM’s best-in-class solutions and third-party services to enable easy development and deployment of key use cases.

Presto (Java) overview

Presto (Java) is a distributed SQL query engine, with the capability to query vast data sets located in different data sources, thus solving data problems at scale.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Presto (Java) provides the ANSI SQL interface, which can be used for all data analytics and IBM watsonx.data use cases. With this feature, you do not need to manage multiple query languages and interfaces to different databases and storage. Presto (Java) is designed for storage abstraction, which allows connections to any data source through its connectors.

IBM watsonx.data uses version **0.286** of Presto (Java).

Presto (Java) server types

A Presto (Java) installation includes three server types - Coordinator, Worker, and Resource manager. Following is a brief explanation of the server types. For more information about the server types, see Presto (Java) concepts in Presto (Java) documentation.

- **Coordinator** - A coordinator is a server type in a Presto (Java) installation, which is responsible for parsing statements, planning queries, and managing Presto (Java) worker nodes. It is the brain of a Presto (Java) installation and is also the node to which a client connects to submit statements for execution. It is also responsible for fetching results from the workers and returning the results to the client.
- **Worker** - A worker is a server type in a Presto (Java) installation, which is responsible for running tasks and processing data. Worker nodes fetch data from connectors and exchange intermediate data with each other.
- **Resource manager** - The resource manager is a server type in presto, which aggregates data from all coordinator and workers and creates a global view of the Presto (Java) cluster.

For more information on supported connectors, see Adding a database-catalog pair.

Presto (Java) SQL Language

For information about SQL language used in Presto (Java), see SQL Language in Presto (Java) documentation.

Data types

By default, Presto (Java) supports the following data types. More types can be provided by plug-ins:

- Boolean
- Integer
- Floating-Point
- Fixed-Precision
- String
- Date and Time
- Structural
- Network Address
- UUID
- HyperLogLog
- KHyperLogLog
- Quantile Digest
- T-Digest

For more information about the data types, see Data types in Presto (Java) documentation.

Reserved keywords

Presto (Java) has a set of reserved keywords for SQL queries. These keywords must be quoted in double quotation marks to be used as an identifier. For the list of reserved keywords, see Reserved keywords in Presto (Java) documentation.

SQL Syntax

For information about SQL syntax used in Presto (Java), see SQL Statement Syntax in Presto (Java) documentation.

For information about provisioning a Presto (Java) engine, see Provisioning a Presto (Java) engine.

For information about switching between Presto (Java) and Presto (C++) engines in developer edition, see Switching between Presto (Java) and Presto (C++) engines in developer edition.

For information about customizing properties, see:

- Configuration properties for Presto (Java) - coordinator and worker nodes
- JVM properties for Presto (Java) - Coordinator and worker nodes
- Catalog properties for Presto (Java)

Presto (C++) overview

Presto (C++) is a version of Presto workers that are implemented in C++ instead of Java by using the Velox library.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Presto (C++) aims to enhance performance for data lakes without requiring a JVM on worker nodes. It supports several connectors, including Hive and Iceberg, and focuses on improved integration with data warehousing systems.

IBM watsonx.data uses version **0.286** of Presto (C++).

Presto (C++) features

Task management
Presto C++ includes HTTP endpoints that allow users to monitor and manage tasks. This feature enhances operational oversight and makes it easier to track ongoing processes.

Remote function execution
Enables executing functions on remote nodes, which enhance scalability and distributed processing capabilities, making data processing more efficient across a network of nodes.

Authentication
Uses JSON Web Tokens (JWT) for secure internal communication between nodes, ensuring that data remains secure and tamper-proof during transmission.

Data caching
Implements asynchronous data caching with prefetching capabilities. This optimizes data retrieval and processing speed by anticipating data needs and caching it in advance.

Performance Tuning
Offers various session properties for performance tuning, including settings for spill thresholds and compression. This allows users to fine-tune performance parameters according to their specific needs, ensuring optimal performance of data processing tasks.

For more information about Presto (C++) features, see Presto C++ features.

For information about provisioning a Presto (C++) engine, see Provisioning a Presto (C++) engine.

For information about switching between Presto (Java) and Presto (C++) engines in developer edition, see Switching between Presto (Java) and Presto (C++) engines in developer edition.

For information about customizing properties, see:

- Configuration properties for Presto (C++) - worker nodes
- Configuration properties for Presto (C++) - coordinator nodes
- Catalog properties for Presto (C++)
- Velox properties for Presto (C++)

Hive Metastore overview

Hive Metastore (HMS) is a service that stores metadata that is related to Presto and other services in a backend Relational Database Management System (RDBMS) or Hadoop Distributed File System (HDFS).

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

When you create a new table, information that is related to the schema such as column names and data types is stored in the metastore relational database. A metastore enables the user to see the data files in the HDFS object storage as if they are stored in tables with HMS.

Metastore acts as a bridge between the schema of the table and the data files that are stored in object storages. HMS holds the definitions, schema, and other metadata for each table and maps the data files and directories to the table representation that is viewed by the user. Therefore, HMS is used as a storage location for the schema and tables. HMS is a metastore server that connects to the object storage to store data and keeps its related metadata on PostgreSQL.

Any database with a JDBC driver can be used as a metastore. Presto makes requests through thrift protocol to HMS. The Presto instance reads and writes data to HMS. HMS supports 5 backend databases as follows. In watsonx.data, PostgreSQL database is used.

- Derby
- MySQL
- MS SQL Server
- Oracle
- PostgreSQL

Currently HMS in watsonx.data supports the Iceberg table format.

The following three modes of deployment are supported for HMS. In watsonx.data the remote mode is used.

- Embedded Metastore - Derby with singe session.
- Local Metastore - MySQl with multiple sessions accessible locally.
- Remote Metastore - metastore runs on its own separate JVM and is accessible by using thrift network APIs.

For more information about exposing Hive metastore ports, see Exposing Hive metastore ports.

For more information about managing access, see Managing access to the Hive Metastore.

For more information about accessing HMS from outside of the OpenShift Container Platform cluster, see Accessing Hive Metastore (HMS) using NodePort.

Data Access Service (DAS) overview

Data Access Service (DAS) proxy in watsonx.data provides a unified way to access object storage, govern external engines, and audit data access. All of these are accomplished without exposing credentials or requiring complex modifications to engines, which are not controlled by watsonx.data.

watsonx.data on Red Hat OpenShift

Currently, DAS signature and DAS proxy features are available in watsonx.data.

Note: DAS signature is available only internally for DataStage and IBM Spark. IBM Spark by default connects to the watsonx.data object storage through DAS signature.

By using DAS proxy, you can access any S3 or S3 compatible object storage, such as IBM Cloud Object Storage, MinIO, Ceph, and ADLS and ABS compatible buckets. If you are using IBM Spark, additional file action capabilities are available.

Note: DAS is a technology preview function, official support will come soon. You can try this preview function and provide feedback.

The following topics provide more information about DAS proxy support for different storage types and file action capabilities:

- Using DAS proxy to access S3 and S3 compatible buckets.
- S3 REST API permissions (specific to IBM Spark and DAS).

- Using DAS proxy to access ADLS and ABS compatible buckets.

Milvus overview

Milvus is a vector database that stores, indexes, and manages embedding vectors used for similarity search and retrieval augmented generation. It is developed to empower embedding similarity search and AI applications. Milvus makes unstructured data search more accessible and consistent across various environments.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

IBM watsonx.data uses version **2.4.0** of Milvus.

With Milvus, you can perform a variety of tasks related to managing and searching vector data, which is crucial for many artificial intelligence (AI) and machine learning (ML) applications.

Here are some of the key things you can do with Milvus:

Vector similarity search

You can search for vectors similar to a query vector from millions of vectors in seconds. Vector similarity search in Milvus is a core feature that allows users to find vectors closest to a given query vector based on a specific metric of similarity. This capability is essential in many applications, such as recommendation systems, image and audio retrieval, natural language processing, and more.

Hybrid search

Conducting a hybrid search in Milvus allows you to combine vector similarity search with traditional relational database-style filtering based on scalar fields. This feature is particularly useful when you need to refine your search results further by filtering on attributes like categories, timestamps, or any other metadata associated with your vectors. Hybrid search leverages both the vector embeddings and the scalar fields to provide more precise and relevant search results.

Creating indexes

Indexing in Milvus involves organizing the data in a way that enables efficient query processing, especially for high-dimensional vector data. Milvus supports several types of indexes, each designed for specific scenarios or data characteristics.

Recommendation Systems

You can use Milvus to power recommendation systems by finding items similar to your customer’s preferences or previous interactions.

For more information about Milvus, see Working with Milvus.

Important: You cannot upgrade from the private-preview version to the GA version of Milvus. You must delete the private preview and add the GA version.

Query Optimizer overview

In the domain of database management systems, query optimizers play a pivotal role in ensuring efficient execution of database queries. **`Query Optimizer`**, a component of IBM watsonx.data, improves the performance of queries that are processed by Presto (C++) engine. If optimization is analyzed to be feasible, the query undergoes rewriting; otherwise, the native engine optimization takes precedence.

Note: **`Query Optimizer`** is not supported for Presto (Java).

Within watsonx.data, **Query Optimizer** operates as a component, tasked with optimizing queries. It accepts a standard SQL query as input and generates an optimized SQL equivalent, which is tailored for enhanced execution by Presto (C++). In instances where optimization is not feasible, the system reverts to using the original query.

Query Optimizer emerges as a valuable addition to the watsonx.data, empowering users to optimize their queries and achieve enhanced performance from their engines.

You can see Query Optimizer section for more information.

Advantages of Query Optimizer

The query optimization feature of Db2 is leveraged in watsonx.data and the key factors considered include:

- watsonx.data uses Db2 as the Decades-Honed Query Optimization for Peak Performance.

  Leveraging extensive development, query optimization feature of Db2 analyzes your SQL queries and generates optimal execution plans. Key factors considered include:

  - Accurate statistics: `RUNSTATS` gathers data distribution and cardinality estimates for informed decisions.
  - Well-designed indexes and constraints: These guide the optimizer towards efficient access paths and enforce data integrity.
  - Advanced techniques for complex queries: Cost-based optimization and cardinality estimation ensure efficient processing.
- Enhanced Query Performance: **Query Optimizer** effectively optimizes queries, leading to significant performance improvements.
- Seamless Integration: **Query Optimizer** seamlessly integrates with existing watsonx.data infrastructure, ensuring a smooth adoption process.
- Flexible Optimization: **Query Optimizer** operates flexibly as users can enable and disable the feature either at global or session level.

Limitation of **Query Optimizer**:

- Query Optimizer cannot be configured from the watsonx.data web console as the console does not use session variables.
- Query Optimizer cannot verify whether an original query or a rewritten query was run. Verify the Presto (Java) log to get this information.
- When metastores are synced, all schemas and tables are in the uppercase. For example, `"catalog.SCHEMA.TABLE"`.
- For optimal performance, you must define constraints like `NOT NULL`, `Primary key`, and `Foreign key` in Query Optimizer after the tables are synced.
- Query Optimizer do not support views.
- Decimal and float columns in the projection list might interchange and can cause mismatch in data type.
- Three-part name queries need quotation marks around the catalog name in lowercase (`“catalog”.schema.table`). Query returns an error otherwise.
- Certain queries (full outer join, anti join) do not return the correct result.
- Special characters in the identifier do not work properly.
- Interval is not supported. Use `date_add`.
- Query Optimizer supports only Hive tables and not Iceberg tables.

Presto (Java) mixed-case support overview

Case-sensitivity refers to the Presto (Java) engine’s capability to distinguish between uppercase and lowercase letters, treating them as distinct characters. This is important when querying databases, as the case of table and column names can affect the query results.

watsonx.data Developer edition

Case-insensitive behavior

Open-source Presto (Java) is case-insensitive. It does not distinguish between uppercase and lowercase characters for schema names, table names, and column names. Presto (Java) converts identifiers to lowercase. The following queries are considered as the same.

1. SELECT * FROM catalog1.schema1.**table**;
2. SELECT * FROM catalog1.schema1.**TABLE**;
3. SELECT * FROM catalog1.schema1.**TaBle**;

Case-sensitive behavior

Presto (Java) engine behavior in IBM watsonx.data was case-insensitive till version 1.0.3. Case sensitivity was introduced in version 1.1.0. All Presto (Java) engine versions from 1.1.0 and above are case-sensitive by default. The table names in the following examples are stored and fetched separately.

1. SELECT * FROM catalog1.schema1.**table**;
2. SELECT * FROM catalog1.schema1.**TABLE**;
3. SELECT * FROM catalog1.schema1.**TaBle**;

Mixed-case feature flag

From IBM watsonx.data version 2.0.0, a new feature is available to switch between both case-sensitive and case-insensitive behavior in Presto (Java) by using a mixed-case feature flag. The mixed-case feature flag is set to OFF in Presto (Java) by default. The flag can be set to ON or OFF as required during deployment of the Presto (Java) engine. It is advised not to toggle between ON and OFF configurations after the deployment, as it may result in inconsistent system behavior.

To configure the flag, you can either configure it by using the Customization API or reach out to the support team.

Note: You can use the following curl command to set the flag as true or false:

```
{
  "engine_properties": {
    "global": {
      "enable-mixed-case-support": "true"
    }
  },
  "engine_restart": "force"
}
```

The following are the two scenarios to illustrate mixed-case support behavior:

Scenario 1: Mixed-case feature flag ON

A. Create multiple tables in the mixed-case feature flag ON cluster for both lowercase and mixed case table names.

- The user can access all the tables.

B. Change the setting for mixed-case feature flag from ON to OFF.

- If there are multiple tables with same name but in mixed cases, then such tables may not be accessible or can cause data discrepancy depending on the connector used.

Scenario 2: Mixed-case feature flag OFF

A. Create multiple tables in the mixed-case feature flag OFF cluster.

- For duplicate table names, only the table that is created first will be fetched.
- For unique table names, all tables are created and fetched.

B. Change the setting for mixed-case feature flag from OFF to ON.

- The user can access all the tables.

For more information on mixed-case behavior, see Mixed-case behavior based on connectors.

API customization overview

API customization in watsonx.data provides a way for instance administrators to customize JVM, CONFIG, catalog, and Velox (Presto (C++)) properties for Presto (Java) and Presto (C++) engines through an API.

watsonx.data on Red Hat OpenShift

This customization method does not require you to add the parameters inside the pod, restart the pod (if there is CPD deployment), or reach out to support personnel (if there is SaaS deployment) for customization. It also does not require any special access and privileges to the backend system. API customization is a unified way to customize allowed properties in watsonx.data, regardless of the deployment. API customization is supported through the PATCH API. API endpoints and sample requests for Presto (Java) and Presto (C++) engines are as follows:

PATCH API (Presto (Java) engine)

Endpoint

```
/presto_engines/{engine_id}
```

Request body

```
{
  "description": "updated description for presto engine",
  "engine_display_name": "sampleEngine",
  "engine_properties": {
    "configuration": {
      "coordinator": {
        "property_1": "property_value",
         "property_2": "property_value"
      },
      "worker": {
        "property_1": "property_value",
         "property_2": "property_value"
      }
    },
    "jvm": {
      "coordinator": {
        "property_1": "property_value",
         "property_2": "property_value"
      },
      "worker": {
        "property_1": "property_value",
         "property_2": "property_value"
      }
    },
   "catalog":{
      "catalog_name":{
           "property_1": "property_value",
         "property_2": "property_value"
      },
},
"global":{

}
   },
  "engine_restart": "force",
  "remove_engine_properties": {
    "configuration": {
      "coordinator": [
        "property1","property2"
      ],
      "worker": [
         "property1","property2"
      ]
    },
    "jvm": {
      "coordinator": [
         "property1","property2"
      ],
      "worker": [
         "property1","property2"
      ]
    },
  "catalog":{
"catalog_name":["property1","property2"]
  },
"global":{

}
  },
  "tags": [
    "tag1",
    "tag2"
  ]
}
```

PATCH API (Presto (C++) engine)

Endpoint

```
/prestissimo_engines/{engine_id}
```

Request body

```
{
  "description": "updated description for prestissimo engine",
  "engine_display_name": "sampleEngine",
  "engine_properties": {
    "configuration": {
      "coordinator": {
        "property_1": "property_value",
         "property_2": "property_value"
      },
      "worker": {
        "property_1": "property_value",
         "property_2": "property_value"
      }
    },
   "catalog":{
      "catalog_name":{
           "property_1": "property_value",
         "property_2": "property_value"
      }},
    "velox": {
            "property_1": "property_value",
         "property_2": "property_value"
        }
  }
   },
  "engine_restart": "force",
  "remove_engine_properties": {
    "configuration": {
      "coordinator": [
        "property1","property2"
      ],
      "worker": [
         "property1","property2"
      ]
    },
  "catalog":{
"catalog_name":["property1","property2"]
  },
"velox":
[
  "property1","property2"
      ]

  },
  "tags": [
    "tag1",
    "tag2"
  ]
}
```

Note: The GET API also supports customization, but is available for internal use only.

You can find the curl example for API customization in Update presto engine.

For the list of properties that can be customized through an API for Presto (Java), see:

- Configuration properties for Presto (Java) - coordinator and worker nodes.
- JVM properties for Presto (Java) - Coordinator and worker nodes
- Catalog properties for Presto (Java)

For the list of properties that can be customized through an API for Presto (C++), see:

- Configuration properties for Presto (C++) - worker nodes
- Configuration properties for Presto (C++) - coordinator nodes
- Catalog properties for Presto (C++)
- Velox properties for Presto (C++)

For properties that must be customized under the guidance of the watsonx.data support team, see Properties to be customized under support guidance.

Note: With IBM watsonx.data 2.0.2 and later, you can get System Access Control (SAC) plug-in logs with DEBUG information. To enable DEBUG logs of SAC plug-in in Presto, trigger the customization API and add `"com.ibm.openlakehouse.prestodb": "DEBUG"` under `logConfig`.

```
"logConfig": {
            "coordinator": {
                "com.ibm.openlakehouse.prestodb": "DEBUG"
            },
            "worker": {}
        }
```

For more information, see Update presto engine.

Data Gate overview

Data Gate is a replication technology that synchronizes data from IBM Z to various hybrid-cloud targets.

watsonx.data on Red Hat OpenShift

Data Gate delivers data that is originated in databases like Db2 for z/OS, IMS, and VSAM to various targets including the Iceberg open data format in watsonx.data.

An IBM z Integrated Information Processor (zIIP)-enabled log data provider captures Db2 for z/OS log changes, consolidates, encrypts, and sends them to a log data processor, which prepares data for loading. Data Gate uses watsonx.data connectors to connect and replicate data.

Important: Data Gate supports only Iceberg catalog for watsonx.data with external storage. IBM-managed storages are not supported.

For more information about connecting and synchronizing Data Gate with watsonx.data, see Connecting Data Gate for watsonx with a watsonx.data target.

For more information about administering and working with Data Gate, see Administering Data Gate for watsonx.

Resource groups overview

Resource groups help to manage query execution and resource allocation. Using resource groups, administrators can control the allocation and utilization of resources on a Presto cluster. Resource groups limit resource usage and enforce policy queues on queries.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Note: The resource group feature is available with IBM watsonx.data 2.0.2 and later.

Key features

- Resource groups can set limits on resource usage such as CPU time, memory usage, or total number of queries (especially in multi-tenant environments to ensure that no single user or query monopolizes the system resources).
- Subgroups in resource groups allow hierarchical resource allocation. Each subgroup can have its own resource limits and policy queues.

For more information about resource groups, see Resource Groups.

For more information about Presto resource group configuration in watsonx.data, see Configuring Presto resource groups.

For more information about the resource group properties that you can define in the resource group `JSON` file, see Resource group properties.

Access management and governance in watsonx.data

This topic provides details about access management and governance in watsonx.data.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

watsonx.data on IBM Cloud®

Access management includes three levels of access control:

- User authentication (Level 1)
- User access to resources (Level 2)
- User access to data (Level 3)

This topic also covers the details of the Data Access Service (DAS) and integration with Apache Ranger and IBM Knowledge Catalog (IKC) for data governance.

User authentication (Level 1)

User authentication has two parts to it:

- **Platform level**: This level defines who is permitted to access the platform. The platform here can be any one of the watsonx.data deployment (CPD, IBM Cloud, AWS, or software).
- **Service level**: Role Based Access Control (RBAC) at a service level in the platform. This specifically applies to watsonx.data on CPD, IBM Cloud, and AWS.

Level 1 authentication in watsonx.data on CPD and software

In watsonx.data software and watsonx.data on CPD, the platform administrators are responsible to decide and implement the best approach for user access management. You can use the internal repository for user records or an enterprise-grade password management solution, such as SAML SSO or LDAP provider for password and access management. For more information, see Connecting to your identity provider.

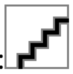
Roles and users - Administrator and User are two predefined roles at the platform level in watsonx.data. For more information about these roles and permissions that can be associated, see Predefined roles and permissions.

You can create users or user groups and assign them the required roles.

Note: If a user or user group has multiple roles, the user or group has all the permissions from all the roles that are assigned to them.

Level 1 authentication in watsonx.data on IBM Cloud

Level 1 authentication in watsonx.data on IBM Cloud is aligned with the IBM Cloud authentication framework. For more information, see IBM Cloud IAM roles and Actions and roles for account management services.

You can create access groups, or give access to a trusted profile, user, or service ID access to any of the target and specific permissions as depicted in the following illustration: 

In addition to the authentication, in IBM cloud, the IAM platform roles are assigned some privileges and permissions by default. The following table provides the details:

Table 1. IAM platform roles

IAM platform roles	**Actions**
IAM Platform Administrator	lakehouse.metastore.admin lakehouse.dashboard.view
IAM Platform Operator, Editor, Viewer	lakehouse.dashboard.view
Others	Depends on the actions that are assigned by the administrator

The following table provides the service role details that are specific to watsonx.data. MetastoreAdmin role is used for Db2, Netezza, and Spark. MetastoreAdmin has full access to HMS Thrift API. MetastoreViewer role has read access to HMS Rest API. The DataAccess role is used only for IKC integration on data profiling.

Table 2. Service roles

Service roles	**Actions**	**Permissions**
MetastoreAdmin	lakehouse.metastore.admin	Manage metastore data
MetastoreViewer	lakehouse.metastore.view	View metastore data
DataAccess (primarily used for service to service integration. For example, IKC integration with WXD)	lakehouse.data.access	Access data

Authentication options

Users can authenticate (log in to watsonx.data instance, invoke an API, or connect to the CLI) through one of the following options:

- **IBM Cloud –** IBM API key or IAM token
- **Software or watsonx.data on CPD –** watsonx.data platform API key or token, user’s instance scope API key or token, user’s username and password.

  Note:

  – To generate an API key, go to your CPD profile page and click **API key** > **Generate API key**.

  – To generate a token, you can use CPD API. For more information, see Using Authorization: Bearer token.
- **Developer edition –** Token that is acquired by calling `.../auth/authenticate` using default user `ibmlhaadmin/password` or your username and password.

Authentication options for `presto-cli`

Users can also use `presto-cli` or connect to Presto via JDBC with one of the following authentication options:

- **IBM Cloud –** IBM API key or IAM token
- **Software or watsonx.data on CPD –** User's platform API key or token (recommended), user's instance scope API key or token, user's username and password.

  Note:

  – To generate an API key, go to your CPD profile page and click **API key** > **Generate API key**.

  – To generate a token, you can use CPD API. For more information, see Using Authorization: Bearer token.
- **Developer edition –** hardcoded default user `ibmlhadmin/password`

For more information, see Connecting to a Presto server and Connecting to Presto server in watsonx.data on IBM Cloud.

User access to resources (Level 2)

With level 2 access control, you can assign roles for watsonx.data users to view, edit, and administer the resources, which include engines, catalogs, storage, and databases.

Controlling access to the engines and other components is a critical requirement for many enterprises. To ensure that the resource usage is under control, IBM watsonx.data provides the ability to manage access controls on these resources. A user with admin privileges on the resources can grant access to other users.

For more information on L2 access control in:

- watsonx.data software and on CPD, see Infrastructure access.
- watsonx.data on IBM Cloud, see Managing users and Managing roles and privileges.

Data access (Level 3)

At the data access level, you can define data access policies and grant or restrict access to schema, table, and columns in watsonx.data.

For more information about data access policies in:

- watsonx.data software and on CPD, see Data policy.
- watsonx.data on IBM Cloud, see Managing data policy rules.

Default username and password in different watsonx.data deployments

Different deployments of watsonx.data have different default username and password as follows:

watsonx.data Developer edition

- **Username –** `ibmlhadmin` is the default username. Based on your requirements, you can add users with roles, User and Admin. For more details, see user-mgmt.
- **Password –** `password` is the default password.

watsonx.data software

- **Username –** `admin` is the default username. If IAM is enabled, the default username is `cpadmin`.
- **Password –** `password` is the default password. To get the default password, run the following command:

  ```
  ibm-lakehouse-manage get-cpd-instance-details
  ```

watsonx.data on CPD

- **Username –** `admin` is the default username. If IAM is enabled, the default username is `cpadmin`.
- **Password –** `password` is the default password. To get the default password, run the following command:

```
cpd-cli manage get-cpd-instance-details \
        --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
        --get_admin_initial_credentials=true
```

watsonx.data on IBM Cloud

- **Username –** Username can either be `ibmlhapikey` or `ibmlhtoken`.
- **Password –** Password can either be IBM Cloud API key or IBM IAM access token. For more information see, Getting IBM API key and Getting IBM Access Management (IAM) token.

Data Access Service (DAS)

Data Access Service (DAS) proxy in watsonx.data provides a unified way to access object storage, govern external engines, and audit data access. All of these are accomplished without exposing credentials or requiring complex modifications to engines, which are not controlled by watsonx.data.

For more information, see Data Access Service overview.

IBM Knowledge Catalog integration for data governance and access control

Integrating watsonx.data with IBM Knowledge Catalog provides self-service access to data assets for knowledge workers who need to use those data assets to gain insights.

For more information, see Integrating with IBM Knowledge Catalog.

Apache Ranger integration for data governance and access control

IBM watsonx.data supports Apache Ranger policies to allow comprehensive data security on integrating with multiple governance tools and engines.

For more information, see Enabling Apache Ranger policy for resources.

Getting connection information

You can see the connection information of watsonx.data from the **Connect information** tile of the **Configurations** page and from the **Instance details** page. For more information about watsonx.data connections, see Getting connection information.

Licensing guidance for watsonx.data

This topic provides information about licensing and entitlements for watsonx.data.

watsonx.data on Red Hat OpenShift

Notice: This Licensing guidance is intended to provide only supplementary information to assist you in deploying one or more programs you have licensed from IBM within your purchased entitlement. Your license agreement such as the IBM International Program License Agreement (IPLA) or equivalent and its transaction documents, including the license information for watsonx.data is the sole and complete agreement between you and IBM regarding use of the program.

If you are not familiar with the concepts of how licensing works for IBM software, see the Licensing Basics, Essential Elements, and Licensing & Compliance pages.

This topic covers the following information:

- Listing of licenses by type
- What do you get with your purchase of watsonx.data and what is your entitlement?

- Options when purchasing watsonx.data
- License ratio topics
- Differences in license terms
- Offering-specific licenses

Listing of licenses by type

The following licenses are used when creating instances of watsonx.data in the `spec.license.license` field of each custom resource:

- Full licenses
- Reserved licenses

For license versions, see the table of license versions.

Full licenses

Full licenses include 1 virtual processor core (VPC) of watsonx.data. To review the license agreements for any of the following full watsonx.data licenses, click the link for that license:

- IBM watsonx.data 2.0.2 (5900-AXD)
- IBM watsonx.data Non-Production 2.0.2 (5900-AXD)
- IBM watsonx.data Perpetual 2.0.2 (5900-AYS)
- IBM watsonx.data Perpetual Non-Production 2.0.2 (5900-AYS)
- IBM watsonx.data 2.0.1 (5900-AXD)
- IBM watsonx.data Non-Production 2.0.1 (5900-AXD)
- IBM watsonx.data Perpetual 2.0.1 (5900-AYS)
- IBM watsonx.data Perpetual Non-Production 2.0.1 (5900-AYS)

For more information, see IBM Terms.

Reserved licenses

Reserved licenses does not include Red Hat OpenShift entitlement. They are intended for organizations that either have an existing Red Hat OpenShift entitlement or commit to using watsonx.data on public cloud environments with managed Red Hat OpenShift. Prior IBM approval is required.

The reserved watsonx.data licenses can be found here:

- IBM watsonx.data Reserved 2.0.2 (5900-AXD)
- IBM watsonx.data Non-Production Reserved 2.0.2 (5900-AXD)
- IBM watsonx.data Perpetual Reserved 2.0.2 (5900-AYS)
- IBM watsonx.data Perpetual Non-Production Reserved 2.0.2 (5900-AYS)
- IBM watsonx.data Reserved 2.0.1 (5900-AXD)
- IBM watsonx.data Non-Production Reserved 2.0.1 (5900-AXD)
- IBM watsonx.data Perpetual Reserved 2.0.1 (5900-AYS)
- IBM watsonx.data Perpetual Non-Production Reserved 2.0.1 (5900-AYS)

For more information, see IBM Terms.

Table of license versions

Table 3. Table of license versions

License	Use	Description
5900-AXD	Production or non-production	IBM watsonx.data 2.0.2

Table 3. Table of license versions (continued)

License	Use	Description
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 2.0.2
5900-AXD	Production or non-production	IBM watsonx.data 2.0.1
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 2.0.1
5900-AXD	Production or non-production	IBM watsonx.data 2.0
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 2.0
5900-AXD	Production or non-production	IBM watsonx.data 1.1.4
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 1.1.4
5900-AXD	Production or non-production	IBM watsonx.data 1.1.3
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 1.1.3
5900-AXD	Production or non-production	IBM watsonx.data 1.1.2
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 1.1.2
5900-AXD	Production or non-production	IBM watsonx.data 1.1.1
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 1.1.1
5900-AXD	Production or non-production	IBM watsonx.data 1.1.0
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 1.1.0
5900-AXD	Production or non-production	IBM watsonx.data 1.0.3
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 1.0.3
5900-AXD	Production or non-production	IBM watsonx.data 1.0.2
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 1.0.2
5900-AXD	Production or non-production	IBM watsonx.data 1.0.1
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 1.0.1
5900-AXD	Production or non-production	IBM watsonx.data 1.0.0
5900-AYS	Production or non-production	IBM watsonx.data Perpetual 1.0.0

What do you get with your purchase of watsonx.data, and what is your entitlement?

IBM watsonx.data offers collect, store, query, and analyze all your enterprise data with a single unified data platform. You can purchase entitlement to watsonx.data 2.0 for different term lengths:

- Perpetual license

- Monthly license
- Subscription license

Your purchase of watsonx.data includes entitlement to:

- watsonx.data 2.0
- Supporting offerings
- Non-charged entitlements for watsonx.data

Supporting offerings

Supporting offerings can be used only to support functionality that is included in watsonx.data. If there is a conflict of license terms, the license terms for watsonx.data supersede the license terms of the supporting offerings.

watsonx.data includes the following supported offerings:

Table 4. Supported offerings

Supporting program	**Permitted component**
IBM Cloud Pak for Data Enterprise Edition	IBM Cloud Pak for Data Control Plane
Analytics Engine Powered by Apache Spark	
IBM Storage Fusion Advanced - Foundation	• Internal deployment mode only. • Excludes disaster recovery, backup components, data cataloging, and advanced encryption with KMS.
IBM Storage Ceph Pro Edition	
IBM Data Virtualization Manager for z/OS	Data Virtualization JDBC Driver
IBM Cloud Pak Foundational Services	• Identity and Access Management (IAM) • Certificate Management • License Service • Platform UI • Install

Note: Supporting components except Analytics Engine Powered by Apache Spark are non-chargeable components. That is, they are not used to determine the number of entitlements required for the program.

Following are the supporting programs with deployment in containers:

- IBM Cloud Pak for Data Control Plane
- Analytics Engine Powered by Apache Spark
- IBM Cloud Pak Foundational Services
- IBM Data Virtualization Manager for z/OS - Data Virtualization JDBC Driver
- Fusion Data Foundation - internal deployment mode only

The licensing for watsonx.data is by the VPC metric; either perpetual, monthly, or a subscription license.

For more information about IBM perpetual, monthly, and subscription licenses, see Passport Advantage Licensing Overview.

For more information about different chargeable metrics such as VPCs and PIUs, see Reported License Metrics.

The parts associated with the perpetual, monthly, and subscription licenses are published in the announcement letter for each release. See the most recent announcement letter for watsonx.data 2.0.

Monthly and subscription licenses for watsonx.data include both the entitlement to use the program and the associated subscription and the support entitlement. The perpetual license entitlement allows use of the program in perpetuity, but for subscription and support after the first 12 months, you must purchase yearly subscription and support renewals.

If customers with perpetual license entitlement do not renew their subscription and support, the support access key expires, and they will no longer be able to download product images from the IBM entitled registry (`cp.icr.io`). Therefore, they lose access to the product images unless they mirror the product images from the IBM entitled registry to a customer-owned registry (before subscription and support lapses) and configure their system to pull from this registry.

Support availability for each watsonx.data release follows the published support model as described in the announcement letter. For an updated view of which releases are supported and whether fixes are available for each release, see the Support Lifecycle page.

Options for purchasing watsonx.data

Currently, there is only **net new licenses** option to purchase watsonx.data. Customers can buy their chosen quantity of watsonx.data licenses as either perpetual, monthly, or subscription.

License ratios

Licensing for watsonx.data is measured by using virtual processor core (VPC). A single watsonx.data license is equal to 1 VPC. All watsonx.data components require watsonx.data - Core.

Table 5. Core details

Component	Entitlement	Compute	Memory	Description
Core	20 VPCs	20 CPUs	17 GB	Core services needed by all watsonx.data engines.

Note: Virtual processor core (VPC) is a unit of measurement that is used to determine the licensing cost of IBM products. When you license a virtual machine (VM), VPC is based on the number of virtual cores that are assigned to the VM. One VPC license is required for every core that is available to the IBM program.

Converting licensing metrics

The following table covers the conversion ratio for VPC to VPC. The entitlements can be used multiple times with different combinations.

watsonx.data includes both chargeable and non-chargeable components. Only chargeable components consume watsonx.data entitlements when deployed. Other components will not consume watsonx.data entitlements but will use OpenShift entitlement if deployed. Examples of the components that are chargeable and non-chargeable are listed in the following table.

Table 6. ***Production and non- production license ratios***

Engine	VPC ratio (capability: watsonx.data)
Presto (Java)	1 : 1
Presto (C++)	1 : 2
Spark	1 : 1
Milvus	3 : 1

What qualifies as non-production workload?

If a program is designated as non-production, it can only be used as part of internal development and test environment for internal non-production activities.

What consumes watsonx.data license entitlements according to the ratio?

- watsonx.data Core

- watsonx.data Presto
- watsonx.data Spark
- watsonx.data Milvus

Additional non-charged entitlements for watsonx.data

Additional flat entitlement means that regardless of the number of entitlements that are obtained for the program, use of the program is limited to the number of entitlements listed.

- IBM Storage Fusion Advanced (check the permitted components section) - 100 usable terabytes.
- IBM Storage Ceph Pro Edition (excluding Red Hat Enterprise Linux) - 768 terabytes raw storage.

Differences in license terms

The license terms for watsonx.data supersede the license terms of the bundled offerings. However, this policy applies only when there is a conflict of terms. Terms that apply to the bundled programs still apply if not superseded.

Note: All deployments of watsonx.data that are deployed on Red Hat OpenShift Container Platform must have sufficient entitlement for the Red Hat OpenShift Container Platform cores that are used.

Offering-specific licenses

Red Hat OpenShift Container Platform entitlements (software subscription and support for the Red Hat OpenShift Container Platform)

The entitlements for Red Hat OpenShift that are included in the watsonx.data entitlement are restricted license entitlements. Restricted license entitlement means that software subscription and support for the Red Hat OpenShift Container Platform that is acquired under your watsonx.data license is only provided for use of the Red Hat OpenShift Container Platform (specifically for watsonx.data and not non-watsonx.data workloads).

The entitlements can be used only for deployments of watsonx.data instances. You cannot use them for third-party deployments or custom code. If you deploy other code or components (such as agents used for monitoring watsonx.data capabilities), you must purchase separate Red Hat OpenShift entitlements to make them available to the cluster. Without that, the deployment of the non-watsonx.data workload on the Red Hat OpenShift licenses result in the OpenShift cores and workload to be unsupported. These additional OpenShift entitlements for running non-watsonx.data workload must be procured separately from the OpenShift entitlements that are granted through IBM watsonx.data.

Note: Organizations deploying watsonx.data on managed OpenShift environments in public clouds such as AWS ROSA, IBM ROKS, or Azure ARO might get discounts on the cost of OpenShift on worker nodes where watsonx.data is deployed. This is based on the OpenShift entitlements that are included in the watsonx.data entitlements. Customers must verify with their public cloud service provider to establish if a discount is available. If no discount is available, check with your IBM representative whether they should buy watsonx.data reserved, which does not include OpenShift entitlement and is sold if deployment will only be in environments with separately entitled OpenShift.

What's new in watsonx.data

Read about the new features and enhancements in the current and previous releases of IBM watsonx.data.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

IBM watsonx.data is a new open architecture lakehouse that combines the elements of the data warehouse and data lakes. The best-in-class features and optimizations available on the watsonx.data make it an optimal choice for next generation data analytics and automation. It is released in three versions:

- Software
- Developer
- Cloud (IBM Cloud and AWS Cloud)

IBM watsonx.data Version 2.0.2

A new version of watsonx.data was released in August 2024.

This release of watsonx.data includes the following features and updates:

Data sources and storage enhancements

- Content Aware Storage (CAS) is now called Data Access Service (DAS).
- A new storage option called IBM Storage Scale is available for Presto (Java), Presto (C++), and Spark engines. For more information, see IBM Storage Scale.
- Apache Hive is upgraded to version 4.0.0.
- You can now use Azure Data Lake Storage (ADLS) with Data Access Service (DAS) to store your data when you submit Spark applications. For more information, see “Submitting Spark application by using native Spark engine” on page 500.
- You can now view the DAS endpoint from the **Storage details** page. For more information, see Exploring the storage objects.

Integration enhancements

- You can now use the governance capabilities of IBM Knowledge Catalog (IKC) for SQL views within the watsonx.data platform. For more information, see Integrating with IBM Knowledge Catalog.
- IBM watsonx.data now supports Apache Ranger policies to govern data with Presto (C++) engines. For more information, see Enabling Apache Ranger policy for resources.

Engine and service enhancements

- Instance administrators can now configure resource groups in Presto. For more information, see Resource groups overview.
- You can now use an API to execute queries and retrieve results. For more information, see API.
- You can now configure or change the log level of Presto (Java) through API customization. For more information, see API.
- You can now use the custom data source option to connect to Black Hole and Local File connectors for the Presto (Java) engine. For more information, see Custom data source.
- Test connection feature is now available for Teradata, Snowflake, and IBM Informix data sources.
- The default value of the `task.max-drivers-per-task` property for Presto (Java) and Presto (C++) workers is now set based on the number of vCPUs.
- The **Starter** size for Milvus provisioning is now removed.

Access management enhancements

- You can now control access to Presto (C++) engines. For more information, see “Infrastructure access” on page 407.
- You can now grant component access to users and user groups in batch. For more information, see Managing user access.
- You can now have System Access Control (SAC) plug-in logs with DEBUG information in Presto. For more information, see “API customization overview” on page 8.

Ingestion enhancements

- You can run queries on all tables under a schema through the SQL query workspace without specifying the path by selecting the required catalogs and schemas from the new drop down list. For more information, Running SQL queries.

- New environment variables are available for Spark ingestion through `ibm-lh tool` command line. For more information, see Spark ingestion through `ibm-lh` tool command line.
- New parameter for certificate verification while ingesting data using Spark REST API. For more information, see Spark REST API ingestion through `ibm-lh` tool command line.

Deprecating the support of co-located and self-managed Spark engines

Co-located and self-managed Spark engines are deprecated in the 2.0.2 release and will not be available from the 2.0.3 release onwards. Use native Spark engine for Spark use cases. To start using native Spark engine, see "Native Spark engine" on page 490.

IBM watsonx.data Version 2.0.1

A new version of watsonx.data was released in July 2024.

This release of watsonx.data includes the following features and updates:

Data sources

- You can now connect watsonx.data to Db2 data sources by using IBM API key as the authentication mechanism. For more information, see IBM Db2.
- From watsonx.data, you can now run a Spark job that accesses data available in a remote Hadoop cluster. For more information, see Submitting Spark jobs to access components in a remote Hadoop cluster.
- Presto (C++) engine can now be associated with Arrow Flight service data sources. Read only operations are supported. The following Arrow Flight service data sources are supported:
  - Salesforce
  - MariaDB
  - Greenplum
  - Apache Derby

  For more information, see Arrow Flight service.
- The following new databases are available for Presto (Java) engine:
  - Redis
  - Apache Druid

  For more information, see Redis and Apache Druid.
- Test connection feature is now added for the following data sources:
  - Oracle
  - Amazon Redshift
  - Elasticsearch

Integrations

- When integrating IBM Knowledge Catalog with watsonx.data, you can configure data protection rules for individual rows in a table, allowing users to access a subset of rows in a table. For more information, see Filtering rows.
- You can now apply the following Apache Ranger policies for Presto (Java) engines:
  - Row-level filtering: Users can access a subset of rows in a table. For more information, see Row-level filtering.
  - Column masking: Restrict users to seeing masked values instead of displaying sensitive data. For more information, see Column masking.
- You can now integrate IBM watsonx.data with on-premises IBM DataStage. You can use DataStage service to load and to read data from IBM watsonx.data. For more information, see DataStage Integration.

- You can integrate IBM® watsonx.data with **Data Virtualization** on Cloud Pak for Data to easily join data from different sources in one unified view, without manual changes, data movement, or replication. For more information, see Integrating with Data Virtualization.

Authentication and authorization

- The Spark access control extension allows additional authorization, enhancing security at the time of application submission. If you enable the extension in the spark configuration, only authorized users are allowed to access and operate watsonx.data catalogs through Spark jobs. For more information, see Spark access control extension.
- IBM watsonx.data now supports object storage proxy and signature for Azure Data Lake Storage and Azure Blob Storage. For more information, see Using DAS proxy to access ADLS and ABS compatible buckets.
- Lightweight Directory Access Protocol (LDAP) is now provided for Teradata and Db2 data sources. The user needs to set up this configuration at the server level. For Teradata, explicitly choose the authentication mechanism type as LDAP in the UI. For more information, see Teradata.

  Note: DAS proxy to access ADLS and ABS buckets and LDAP enhancements are Tech preview in version 2.0.1.
- Milvus now supports partition-level isolation for users. Administrators can authorize specific user actions on partitions. For more information, see Infrastructure Access.

Storage

- You can now use Google Cloud Storage to store and access your data while submitting Spark applications. For more information, see Submitting Spark application.
- You can now add the following storage to Presto (Java) engine in watsonx.data:
  - Azure Data Lake Storage Gen2
  - Azure Data Lake Storage Gen1 Blob

  For more information, see Adding storage-catalog pair.
- You can modify the access key and secret key of a user-registered bucket for a storage. This feature is not applicable to default buckets, ADLS, or Google Cloud Storage. This feature can only be used if the new credentials successfully pass the test connection.

Engines

- You can now use the `ALTER TABLE ADD`, `DROP`, and `RENAME` column statements for MongoDB data source.
- You can now configure how Presto handles unsupported data types. For more information, see Catalog properties for Presto (Java).

Catalogs

- You can now associate and disassociate catalogs to an engine in bulk through UI under **Manage associations** in the **Infrastructure manager** page.

API Customization and properties

- The following customization parameters are added for Presto (C++) workers.
  - `system-mem-limit-gb`
  - `system-mem-shrink-gb`
  - `system-mem-pushback-enabled`

  For more information, see Configuration properties for Presto (C++) - worker nodes.
- Enhanced API customization to support data cache and fragment result cache for performance improvement. For more information, see Configuration properties for Presto (Java) - coordinator and worker nodes and Catalog properties for Presto (Java).

- The configuration property `optimizer.size-based-join-flipping-enabled` is added for Presto (C++) coordinator nodes. For more information, see Configuration properties for Presto (C++) - coordinator nodes.

Ingestion

- You can now cancel an ingestion job from the user interface. For more information, see Ingesting data by using Spark.

Infrastructure manager

- You can use search feature for the following values on the **Infrastructure manager** page:
  - database name
  - registered hostname
  - created by username
- You can now use the 'Do Not Disturb' toggle switch in the Notifications section under the bell icon to enable or disable pop-up notifications.
- You can find the connectivity information under the **Connect information** tile in the **Configurations** page. This information can be copied and downloaded to a JSON snippet.

Query Optimizer

- You can Activate and Deactivate Query Optimizer Manager through the web console. For more information, see Activating Query Optimizer.

IBM watsonx.data Version 2.0.0

A new version of watsonx.data was released in June 2024.

This release of watsonx.data includes the following features and updates:

Azure Data Lake Storage Gen2 (ADLS), Azure Blob and Google Cloud Storage

You can now use the following storage types:

- You can now add Azure Blob, Azure Data Lake Storage Gen2 (ADLS), and Google Cloud Storage to watsonx.data.
- You can now use Azure Data Lake Storage (ADLS) Gen1 and Gen2 to store your data while submitting Spark applications.

For more information, see Adding a storage-catalog pair.

New Arrow Flight service-based data sources

You can now use the following data sources with Arrow Flight service:

- Greenplum
- Salesforce
- MariaDB
- Apache Derby

For more information, see Arrow Flight service.

New data sources

You can now use the following data sources:

- Cassandra
- BigQuery
- ClickHouse
- Apache Pinot

For more information, see Adding a database-catalog pair.

New page for Bring Your Own JAR (BYOJ) process for SAP HANA data source

Users can now use a new dedicated section **Driver manager** under the new **Configurations** page to manage drivers for SAP HANA data source. Each of these drivers undergoes a series of validation.

For more information, see SAP HANA.

Apache Ranger policies

IBM watsonx.data now supports Apache Ranger policies to allow integration with Presto engines.

For more information, see Apache Ranger policy.

Provision Spark as a native engine

In addition to registering external Spark engines, you can now provision a native Spark engine in watsonx.data. With the native Spark engine, you can manage Spark engine configuration, manage access to Spark engines, and view applications by using REST API endpoints from watsonx.data.

For more information, see Native Spark engine.

Query Optimizer to improve query performance

You can now use Query Optimizer, to improve the performance of queries that are processed by the Presto (C++) engine. If Query Optimizer determines that optimization is feasible, the query undergoes rewriting; otherwise, the native engine optimization takes precedence.

For more information, see “Query Optimizer overview” on page 5.

New name for Presto engine in watsonx.data

Presto is renamed to Presto (Java).

New engine (Presto C++) in watsonx.data

You can provision a Presto (C++) engine (version 0.286) in watsonx.data to run SQL queries on your data source and fetch the queried data.

For more information, see Presto (C++) overview.

API Customization feature

You can now use catalog and engine API Customization for Presto (Java) and Presto (C++) engines in watsonx.data.

For more information, see IBM API docs.

Mixed case feature flag for Presto (Java) engine

The mixed case feature flag, which allows to switch between case sensitive and case insensitive behavior in Presto (Java), is available. The flag is set to OFF by default and can be set to ON during the deployment of watsonx.data.

For more information, see “Presto (Java) mixed-case support overview” on page 6.

Using proxy to access S3 and S3 compatible buckets

External applications and query engines can access the S3 and S3 compatible buckets managed by watsonx.data through an S3 proxy.

For more information, see Using DAS proxy to access S3 and S3 compatible buckets.

Semantic automation for data enrichment

Semantic automation for data enrichment uses generative AI with IBM Knowledge Catalog to understand your data on a deeper level and enhance data with automated enrichment to make it valuable for analysis.

For more information, see Semantic automation for data enrichment in watsonx.data.

Hive Metastore (HMS) access in watsonx.data

You can now fetch metadata information for Hive Metastore by using REST APIs instead of getting the information from the engine details. HMS details are used by external entities to integrate with watsonx.data. You must have an Admin, Metastore Admin, or Metastore Viewer role to run the API.

Manage resource quota limits for your Spark engine

You can now manage the resource usage quota for the Spark engine in Cloud Pak for Data by using the REST API or from the Spark engine details page.

For more information, see Managing resource quota.

Version upgrade

- Presto (Java) engine is now upgraded to version 0.286.
- Milvus service is now upgraded to version to 2.4.0. Important features include:
  - Better Performance (Low Memory Utilisation)
  - Support Sparse Data
  - Inbuilt SPLADE Engine for Sparse Vector Embedding
  - BGE M3 Hybrid (Dense+Sparse) Search

Command to retrieve ingestion history

You can now retrieve the status of all ingestion jobs that are submitted by using the `ibm-lh get-status --all-jobs` CLI command. You can retrieve the status of all ingestion jobs that are submitted. You get the history records that you have access to.

For more information, see Options and parameters supported in ibm-lh tool.

New operations for Db2 data source

You can perform the following operations for `BLOB` and `CLOB` data types for Db2 data source:

- `INSERT`
- `CREATE`
- `CTAS`
- `ALTER`
- `DROP`

New data types for data sources

The following new data types are now available for some data sources. You can access these data types on the **Data manager** page under the **Add column** option.

`BLOB`

- Db2
- Teradata
- Oracle
- MySQL
- SingleStore

`CLOB`

- Db2
- Teradata
- Oracle

`BINARY`

- SQL Server
- MySQL

Because the `numeric` data type is not supported in watsonx.data, you can use the `decimal` data type as an equivalent alternative to the `numeric` data type for Netezza data source.

You can now use the `BLOB` and `CLOB` data types with the `SELECT` statement in the **Query workspace** to build and run queries against your data for Oracle and SingleStore data sources.

You can now use the BLOB and CLOB data types for MySQL and PostgreSQL data sources as equivalents to LONGTEXT, BYTEA, and TEXT because these data types are not compatible with Presto (Java). These data types are mapped to CLOB and BLOB in Presto (Java) if data sources have existing tables with LONGTEXT, TEXT, and BYTEA data types.

- MySQL (CLOB as equivalent to LONGTEXT)
- PostgreSQL (CLOB as equivalent to TEXT)
- PostgreSQL (BLOB as equivalent to BYTEA)
- Netezza (`decimal` as equivalent to `numeric`)
- Oracle (BLOB and CLOB with the SELECT statement)
- SingleStore (BLOB and CLOB with the SELECT statement)

Known issues and limitations

The following known issues and limitations apply to IBM watsonx.data.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Known issues: Access

- Presto workloads and zen core authentication failure
- Inconsistent rule display and performance in **Access Control** page
- Data access denied due to credential mismatch
- User is still visible in the **Access control** page of an engine after removing the user from the CPD platform
- User cannot access tables created in the COS bucket due to a missing metadata error
- RBAC management for buckets with non-compliant names
- Access is denied when querying an external database
- Assigning Grant or Revoke privilege
- Only table creator has DROP access in Apache Hive (API)
- User-provided certificates are not supported by watsonx.data

Presto workloads and zen core authentication failure

Applies to: 2.0.0

When you execute heavy workloads with presto engine, the zen core fail to authenticate user password and you get the following error :

```
CallAMS API failed. {"errors":null,"exception":"Failed to get zen platform token:
Unauthorized","message_code":"Unauthorized","status_code":401}
```

Workaround: Use zen API key and scale up the Common Services pods to reduce the occurrence of this error.

Inconsistent rule display and performance in Access Control page

Applies to: 2.0.0

Fixed in: 2.0.1

An intermittent issue has been identified within the Access Control page. Users might encounter the following problems:

- After updating rules in an existing policy, the rule table might not immediately reflect the latest changes. The displayed data could be outdated or incorrect.

- Adding rules continuously to a policy might lead to performance issues, causing the page to become unresponsive.

Workaround: If you encounter either of these issues, simply close the policy window and reopen it. This should refresh the rule table data and restore normal page responsiveness.

Data access denied due to credential mismatch

Applies to: 1.1.0 and later

If a CPD user connects to watsonx.data using credentials other than their own, the data access authorization is evaluated based on these connection credentials. This means that the access may be different than the CPD user's access. Specifically, creating and profiling a new asset in the catalog will fail because an unprofiled asset can only be accessed by the asset owner. Though the current user is the creator and owner, the connection to watsonx.data is through a different user. In such situations, watsonx.data does not identify the asset creator as the asset owner.

Workaround:

Use the credentials of the CPD user to access watsonx.data for consistent user identification and data authorization.

The user is still visible in the Access control page of an engine after removing the user from the CPD platform

Applies to: 1.1.3 and later

User cannot access tables that are created in the COS bucket due to a missing metadata error

Applies to: 1.1.2 and later

Sometimes, the data that is created from the COS data source by using the iceberg catalog becomes inaccessible due to a missing metadata error.

```
(Unable to execute HTTP request: No X509TrustManager implementation
available)
```

Users cannot create any additional schemas and tables in COS.

Workaround: To recover all missing tables from your existing COS iceberg catalog, complete the following steps.

1. Deactivate the COS bucket.
2. Dissociate the COS catalog from Presto (Java) engine.
3. Associate the COS catalog with Presto (Java) engine.
4. Select **Sync all objects** from the sync catalog dialog box and click **Sync**.

RBAC management for buckets with non-compliant names

Applies to: 2.0.0 and later

Users attempting to manage role-based access control (RBAC) for buckets with names that deviate from the following regular expression (`^[a-zA-Z0-9-_.]+$`) encounters an error. Bucket names must consist of only lowercase and uppercase letters, numbers, underscores, and hyphens. Any characters apart from this list prevents RBAC management functions from working as expected.

Access is denied when querying an external database

Applies to: 1.1.0 and later

When a user with the User role and Create access (the user only has Create access), is added to an external database, they cannot run the select query from the table they have created. Though the user can connect to the Presto (Java) engine and create tables and schemas, they cannot query from the table. The system displays a `Access Denied` message.

```
Query 20230608_132213_00042_wpmk2 failed: Access Denied: Cannot select from columns [id] in
table or view tab_appiduser_01
```

Workaround: Provide select privilege for the table the user created.

Assigning Grant or Revoke privilege

Applies to: 1.1.0 and later

Assigning **Grant** or **Revoke** privilege to a user through access policy does not work as expected in the following scenarios:

- User_A adds a bucket and a Hive catalog (for example, `useracat02`).
- User_A creates a schema and a table.
- User_B and User_C are assigned **User** roles to the catalog.
- User_A adds allow grant policy to User_B
- User_B connects to the catalog and runs `grant select` to User_C.

```
presto:default> grant select on useracat02.schema_test_01.tab_1 to "6ff74bf7-
b71b-42f2-88d9-a98fdbaed304";
```

- When the User_C connects to the catalog and runs `select` command on the table, the command fails with access denied message.

```
presto:default> select * from useracat02.schema_test_01.tab_1;
Query 20230612_073938_00132_hthnz failed: Access Denied: Cannot select from columns
[name, id, salary, age] in table or view tab_1
```

Only table creator has DROP access in Apache Hive (API)

Applies to: 1.1.0 and later

Only the creator of a table can drop the table that is created in the Apache Hive catalog. Other users cannot drop the table even if they have an explicit DROP access to the table. They get the `Access Denied` message.

User-provided certificates are not supported by watsonx.data

Applies to: 1.1.0 and later

User-provided certificates are not supported in watsonx.data when adding database connections, object store buckets, or by using **`ibm-lh`** utility.

Known issues: Catalog, schema, and tables

- Time data type support in Hive and Iceberg
- Presto (Java) does not support creation of schema and table in the Delta Lake
- Special characters and mixed case impacting data synchronization
- Accessing Hive and Iceberg tables in the same glue metastore catalog
- Table names with multiple dots
- Timeout error when creating schema in AFM bucket
- Creating a schema without a location
- Creating schema location with path
- Enabling Amazon S3 bucket makes catalogs inactive for 15 minutes
- Unable to view created schema

- Unique names for schema and bucket

Time data type support in Hive and Iceberg

Applies to: 2.0.0 and later

Hive: The Hive catalog does not natively support the time data type.

Iceberg: Iceberg does support the time data type.

Workaround: To enable correct handling of time data in Iceberg tables, the `hive.parquet-batch-read-optimization-enabled` property must be set to `false`.

Presto (Java) does not support creation of schema and table in the Delta Lake

Applies to: 2.0.0 and later

Presto (Java) does not support creation of schema and table in the Delta Lake. The Delta Lake connector relies on the Hive metastore to find the location of Delta Lake tables.

Workaround: You can create table in the Delta Lake by using external location.

Example:

```
CREATE TABLE sales.apac.sales_data_new (dummyColumn INT)
WITH (external_location = 's3://db-sa-datasets/presto/sales_data_new');
```

Special characters and mixed cases impacting data synchronization

Applies to: 2.0.0 and later

When synchronizing data between buckets containing tables or schemas with special characters or mixed case letters in their names, you might encounter with the following unexpected behaviors:

- Tables or schemas with certain special characters `%`, `,`, `{`, `)`, `(`, `@`, `$`, `[`, `:` has their data entirely skipped during synchronization.
- Tables or schemas with mixed case or uppercase letters are converted to lowercase before synchronization.

Workaround: Avoid using special characters and mixed cases in table and schema names. Rename existing tables and schemas to use only the supported characters.

Accessing Hive and Iceberg tables in the same glue metastore catalog

Applies to: 1.1.0 and later

When using the AWS Glue Data Catalog to manage a bucket or storage location containing both Iceberg and Hive tables, attempting to access Iceberg tables from the Hive catalog gives, `Not a Hive table` error and attempting to access Hive tables from the Iceberg catalog gives, `Not an Iceberg table` error.

Table names with multiple dots

Applies to: 1.1.0 and later

Presto (Java) does not support creating or querying table names that contain three or more consecutive dots in its name. Attempts to reference such tables in queries may result in errors.

Timeout error when creating schema in AFM bucket

Applies to: 1.1.1 and later

When you try to create a schema in an Active file management (AFM) bucket present in the Spectrum Scale system, you might encounter a timeout error.

Creating a schema without a location

Applies to: 1.1.0 and later

When you create a schema without a location, it is not listed in the schema list of any catalog.

For example, if you create a schema without specifying the location of the bucket, the schema is created in HMS and not in the bucket. When you try to create a new schema with the same name, it fails and responds that the schema already exists.

Workaround: Specify the location of the bucket when creating a schema.

Creating schema location with path

Applies to: 1.1.0 and later

Use one of the following location options when creating a schema:

- Location pointing to a `bucket/subpath` without a trailing `/`.
- Location pointing to a `bucket/subpath` with a trailing `/` – Recommended for better structuring.

Note: Though you can use a location pointing to a bucket only with or without a trailing `/`, it might lead to failure. Therefore, it is recommended to use a subpath. The subpath must be the same as the schema name. For example, to create a schema `test1`, specify the location as `location='s3a://iceberg-bucket/test1'`

Enabling Amazon S3 bucket makes catalogs inactive for 15 minutes

Applies to: 1.1.0 and later

After enabling Amazon S3 bucket type, you need to wait for 15 minutes to use the catalogs. You might get the following error:

```
Failed to create external path _s3a://bucket/schema_ name for database _database name_.
This may result in access not being allowed if the StorageBasedAuthorizationProvider is
enabled: null
```

Workaround: If the error persists, restart the HMS pod.

```
oc get po -n cpd-instance |grep ibm-lh-lakehouse-hive-metastore |awk '{print $1}' |xargs oc
delete po -n cpd-instance
```

Unable to view created schema

Applies to: 1.1.0 and later

When a user with the User role and the Create access (the user only has the Create access) is added to an external database, they cannot see the schemas that they created. Though the user can create schemas, they cannot view them. The following is the system response:

```

presto:default> show schemas;
 Schema
--------
(0 rows)
```

Workaround: Provide select privilege for the schema the user created.

Unique names for schema and storage

Applies to: 1.1.0 and later

A schema and a storage cannot be created with the same name.

For example, if you create a schema that is named “sales” in one catalog, the same name cannot be used for another schema in another catalog. Similarly, if you register a storage with the name “salesbucket”, another storage with the same cannot be registered, even if the storage is located in a different object store.

Workaround: Use unique names when creating schemas and storages.

Known issues: Database and Storage

- `Server concurrency limit reached` error in flight server
- Bring your own bucket (BYOB) for Query history monitoring management (QHMM) feature does not support Amazon S3 bucket.
- Using `ID` as a column name in Cassandra `CREATE TABLE`
- Custom database crashes due to invalid parameters or properties

`Server concurrency limit reached` error in flight server

Applies to: 2.0.0 and later

You might encounter a `Server concurrency limit reached` error when using the flight server to run queries. This occurs when the server experiences high memory usage due to a large number of concurrent requests.

Workaround: Simplify the query and try after few minutes.

Bring your own bucket (BYOB) for Query history monitoring management (QHMM) feature does not support Amazon S3 bucket.

Applies to: 2.0.1

Fixed in: 2.0.2

Using `ID` as a column name in Cassandra `CREATE TABLE`

Applies to: 2.0.0 and later

In Cassandra, you cannot create a table with a column named `ID` while using a Cassandra connector through Presto. This is because `ID` is a reserved keyword for the Cassandra driver that is used by Presto, which automatically generates a UUID for each row. Attempting to create a table with a column name `ID` results in an error message indicating a duplicate column declaration as follows:

```
Duplicate column 'id' declaration for table 'tm_lakehouse_engine_ks.testtable12'
```

Workaround: Avoid using `ID` as a column name when creating Cassandra tables through Presto.

Custom database crashes due to invalid parameters or properties

Applies to: 1.1.4 and later

Creating a custom database with invalid parameters or properties can lead to the database crashing due to memory issues or other internal errors. The custom database becomes unavailable or unusable and other functions within the platform might also be impacted.

Workaround: You can directly modify the database record to remove or correct the problematic parameters or properties. In IBM Cloud Pak for Data (CPD), using the Patch API to update the database configuration can trigger a restart of the engine process to resolve the issue once the property or parameter is removed. The problematic parameters or properties could be removed using the patch API for customization by passing the property name in `remove_engine_properties`.

1. Login to Postgres database by using the command:

   ```
   oc exec -it ibm-lh-postgres-edb-1 -- bash
   psql -U postgres -l
   psql -U postgres ibm_lh_repo
   \dt
   ```

2. Remove the database record that contains wrong parameter from `metadata_properties`.

   Example:

   ```
   delete from metadata_properties where type = 'jvm-worker' and service_id='presto-01';
   ```

3. Restart engine using the Patch API.

Known issues: Engine

- Invalid file associations in Pesto resource group through UI and engine restart issues
- Invalid connection information for co-located Spark engine
- Presto pod restarts with invalid API patch
- Associating SAP HANA with Presto (Java) engine gives an error
- Columns of type numeric array are left out while starting `DatabaseMetadata.getColumns()`
- The deployment type is singlenode
- Issue with prestodb module in ibm-lh-client
- Presto (Java) do not recognize the path as a directory
- Presto (Java) JDBC driver returns incorrect values of date and timestamp

Invalid file associations in Pesto resource group through UI and engine restart issues

Applies to: 2.0.2 and later

CPD: When an invalid file is associated for an engine in Pesto resource group through watsonx.data UI in Cloud Pak for Data (CPD), the engine will experience a restart. However, the user interface may incorrectly display that the engine is using the newly assigned file.

Developer Package: When an invalid file is associated for an engine in Pesto resource group through watsonx.data UI in Developer package, the engine becomes stuck. In this situation, the engine cannot be deleted, and the resource group file cannot be unassigned.

Workaround: If you find that the new file is not associated in CPD environment, reach out to IBM support for further assistance.

Invalid connection information for co-located Spark engine

Applies to: 2.0.0

Fixed in: 2.0.1

Users creating a new Analytics Engine on IBM Cloud Pak for Data (CPD) and attempting to add the co-located Spark engine on the watsonx.data might encounter with the following error.

```
Invalid connection info received for instance: <zen_instance_id>
```

Workaround: If you encounter this error, wait for a minute or two for the Spark engine to be in `Running` status and try adding the co-located Spark engine. The specific waiting time may vary.

Presto pod restarts with invalid API patch

Applies to: 2.0.0

When using the API patch method to configure a Presto engine with an invalid parameter (example, `task.concurrency=15` but the valid value is 16 it can lead to unnecessary pod restarts.

While the engine does not start with an invalid parameter, subsequent patches to correct the issue will not take effect and will need to wait for operator to restart the pod to make the correction to take effect.

Associating SAP HANA with Presto (Java) engine gives an error

Applies to: 1.1.1 and later

You might get an error message when associating SAP HANA with the Presto (Java) engine. However, the association is successful and queries can be run. There is no workaround for this error message currently.

Columns of type numeric array are left out while starting `DatabaseMetadata.getColumns()`

Applies to: 1.1.0 and later

When you start `DatabaseMetadata.getColumns()` using the Presto (Java) JDBC driver, columns of type numeric array are left out.

The default deployment type is singlenode

Applies to: 1.1.0 and later

When an engine is created, regardless of the configuration mode that is selected, the deployed engine is always singlenode and small size.

Workaround: Customize the engine to multinode and different resource configurations, refer to Specifying additional customization for watsonx.data.

Issue with `prestodb` module in `ibm-lh-client`

Applies to: 1.1.0 and later

Due to an issue with the prestodb module in ibm-lh-client, you must complete the following steps to connect to a Python client when using ibm-lh-client:

1. Start the sandbox container for the registered Presto engine.

   ```
   ibm-lh-client/bin/dev-sandbox --engine=demo-b
   ```

2. In the bash prompt, install the prestodb module.

   ```
   export HOME=/tmp
   pip3 install SQLAlchemy 'pyhive[presto]' presto-python-client
   ```

Presto (Java) do not recognize the path as a directory

Applies to: 1.1.0 and later

When you create a new table with a Presto (Java) Hive connector that uses an S3 folder from an external location, Presto (Java) does not recognize the path as a directory and an error might occur.

For example: When creating a customer table in the target directory `DBCERT/tbint` in a bucket that is called dqmdbcertpq by using the **`IBM Cloud`** UX and **`Aspera`** S3 console, the following error is encountered: `External location must be a directory`.

```
CREATE TABLE "hive-beta"."dbcert"."tbint" (
RNUM int , CBINT bigint
) WITH (
format='PARQUET', external_location = 's3a://dqmdbcertpq/DBCERT/tbint'
);
 Query 20230509_113537_00355_cn58z failed: External location must be a directory
```

Objects in a file system are stored as objects and their path. The object and path must have an associated metadata. If the path is not associated with the metadata, Presto (Java) fails to recognize the object and responds that the path is not a directory.

Presto (Java) JDBC driver returns incorrect values of date and timestamp

Applies to: 1.1.0 and later

Date and timestamp string literals before 1900-01-01 return incorrect values.

Known issues: Ingestion

- Inconsistent CSV and Parquet file ingestion behavior
- Ingestion is not possible in non-interactive mode using Presto
- Delayed UI update after successful ingestion jobs
- Incorrect recognition of Gregorian dates in Presto with Hive Parquet tables
- Scala jobs shall fail in co-located, self-managed, and fully managed Spark engines
- Discrepancy in access policy enforcement between user interface (UI) and `ibm-lh data-copy` utility

- Token expired error using Spark ingestion through web console
- Ingestion not supported using external Spark
- Ingestion fails if CSV file contains bad record
- `No columns to parse from file` error
- Spark ingestion through UI is not possible without Presto (Java) engine permission
- Special characters in target table names can cause ingestion failures
- The command `CREATE TABLE AS SELECT` (CTAS) fails in Db2 when attempted on ingested data tables in Iceberg

Inconsistent CSV and Parquet file ingestion behavior

Applies to: 2.0.0 and later

Despite the design specifications stating that CSV files should only be ingested into tables created from CSV files, and parquet files should only be ingested into tables created from parquet files, there is a discrepancy in the actual behavior where users are able to ingest CSV files into parquet tables. This might result in unexpected results like columns with null values, data quality issues, or performance problems if the schema or formatting of the CSV or parquet file does not align with the expected structure of the target table.

Ingestion is not possible in non-interactive mode using Presto

Applies to: 2.0.0 and later

Fixed in: 2.0.2

Ingestion is not possible in non-interactive mode when using the Presto mode of ingestion due to an issue.

Workaround: You can set the environment variable in interactive mode to run the ingestion job using Presto.

Delayed UI update after successful ingestion jobs

Applies to: 2.0.0 and later

After a successful ingestion job, the schema or table may not be immediately visible in the user interface (UI). This is due to the background execution of the ingestion process.

Workaround: Refresh your browser or refresh the catalogs or schemas from the **Data manager** page after an ingestion job status is changed to **Finished** to make sure that the UI is updated with the newly created schema or table. Once the UI is refreshed, you can proceed to run another ingestion job on the same schema or tables that were just created.

Incorrect recognition of Gregorian dates in Presto with Hive Parquet tables

Applies to: 2.0.0 and later

Presto exhibits issues when processing historical dates prior to `0200-01-01`, specifically when they are stored in Hive tables formatted as Parquet. This issue occurs due to the conversion between the Gregorian and Julian calendars, which were implemented in `1582-10-15`. Dates before this cutoff date are misinterpreted by Presto.

Scala jobs shall fail in co-located, self-managed, and fully managed Spark engines

Applies to: 2.0.0 and later

Fixed in: 2.0.2

Discrepancy in access policy enforcement between user interface (UI) and `ibm-lh data-copy` utility

Applies to: 1.1.1 and later

Fixed in: 2.0.2

It is noticed that user defined access policies are not enforced when unauthorized users perform data ingestion by using `ibm-lh data-copy` utility. Whereas, the same policy restricts unauthorized users from performing data ingestion in the UI. This inconsistency in access policy enforcement can lead to unintended data ingestion by unauthorized users.

Token expired error while using Spark ingestion through web console

Applies to: 1.1.2 and later

When you ingest by using Spark through the web console, the ingestion job shall continue running and finish successfully even if you encounter a false error message as follows:

```
Spark ingestion error. Job status command authorization token expired or does not have
sufficient permission.
```

Ingestion is not supported by using external Spark

Though an external Spark engine (**Fully** or **Self managed**) can be added to watsonx.data, ingestion job is not possible using the Spark engine. Ingestion is supported only by a **Colocated** Spark engine.

Applies to: 1.1.2 and later

Ingestion fails if CSV file contains bad record

Applies to: 1.1.0 and later

`ibm-lh` tool does not support skipping maximum bad records for CSV files if the mismatch field is greater than the table definition.

`No columns to parse from file` error

Applies to: 1.1.0 and later

When you try to ingest folder from AWS S3 using the **`ibm-lh`** tool, the following error may be encountered if there are '0' sized empty files in the folder:

```
No columns to parse from file
```

Workaround: First list the folders inside the bucket by using `aws s3 ls` command. If '0' sized empty files are listed, copy all the files to another folder by using `aws s3 cp` command.

Spark ingestion through UI is not possible without Presto (Java) engine permission

Applies to: 1.1.0 and later

You cannot ingest the data through UI by using Spark engine without an admin role for the default Presto (Java) engine.

Special characters in target table names can cause ingestion failures

Ingestion fails if a target table name contains special characters in it when ingesting through the web console.

Applies to: 1.1.0 and later

Workaround: You can ingest data by using ingestion through Spark CLI.

The command `CREATE TABLE AS SELECT` (CTAS) fails in Db2 when attempted on ingested data tables in Iceberg

Applies to: 1.1.0 and later

Some versions of Db2 require an explicit length specification for VARCHAR columns. This requirement causes failure of the command `CREATE TABLE AS SELECT` (CTAS) in Db2 when attempted on ingested data tables in Iceberg.

Workaround: Change the SQL statement from VARCHAR to `VARCHAR(20)`.

Example:

```
create table "db2"."testgaissue"."testga" as (
  select
    cast(checkingstatus as varchar(100)) as checkingstatus,
    loanduration,
    cast(credithistory as varchar(100)) as credithistory,
    cast(loanpurpose as varchar(100)) as loanpurpose,
    loanamount,
    cast(existingsavings as varchar(100)) as existingsavings,
    cast(employmentduration as varchar(100)) as employmentduration,
    installmentpercent,
    cast(sex as varchar(100)) as sex,
    cast(othersonloan as varchar(100)) as othersonloan,
    currentresidenceduration,
    cast(ownsproperty as varchar(100)) as ownsproperty,
    age,
    cast(installmentplans as varchar(100)) as installmentplans,
    cast(housing as varchar(100)) as housing,
    existingcreditscount,
    cast(job as varchar(100)) as job,
    dependents,
    cast(telephone as varchar(100)) as telephone,
    cast(foreignworker as varchar(100)) as foreignworker,
    cast(risk as varchar(100)) as risk
  from
    "iceberg_data"."project"."testga"
);
```

Known issues: Installation and upgrade

- Milvus API fails when upgrading from 2.0.0. to 2.0.1 or higher versions
- Installation of watsonx.data is stuck in `In Progress` state in the CPD 4.8.5 version
- Mixed case functionality might fail when upgrading from 1.1.4 to 2.0.0 or 2.0.1
- Upgrading might encounter `Catalog does not exist` error
- Upgrading standalone watsonx.data might fail
- Installation path directory with space

Milvus API fails when upgrading from 2.0.0 to 2.0.1 or higher versions

Applies to: 2.0.0 only

When upgrading watsonx.data 2.0.0 version to 2.0.1 or higher versions, Milvus API fails with the following error:

```
“exception”: “Authmode values not exist in the table”
```

or

```
“exception”: “ADLS Authmode doesn’t match or not a supported auth method”
```

Workaround: To resolve the issue, you must remove and re-add the existing ADLS storage configuration in the watsonx.data instance.

Installation of watsonx.data is stuck in `In Progress` state in the CPD 4.8.5 version

Applies to: 1.1.4 only

When attempting to install watsonx.data (`wxdaddon`) on 4.8.5 deployment, the installation process stalls and the `wxdaddon` Custom Resource (CR) remains in `In Progress` state with a `MODULE FAILURE` error message.

Workaround: You must contact IBM support if encountered with the `MODULE FAILURE` error.

Mixed case functionality might fail when upgrading from 1.1.4 to 2.0.0 or 2.0.1

Applies to: 1.1.4

Upgrading from Cloud Pak for Data (CPD) version 1.1.4 to 2.0.0 or the developer package version 1.1.4 to 2.0.0 or 2.0.1 does not update the parameter `enable-mix-case-support` to `true` which might affect the schemas and tables in mixed case.

Workaround: To be able to access all the schemas and tables including the mixed case names, you must follow the instructions:

1. For CPD upgrades: Run the customization API patch.

   After the upgrade, you must execute a customization API patch with the global parameter `"enable-mixed-case-support": "true"`.
2. For Developer Package Upgrades:
   a. Log in to the `ibm-lh-presto` pod within your environment.
   b. Locate and edit the `/opt/presto/etc/custom-config.properties` file.
   c. Update the property `enable-mixed-case-support` in the file to set as `true`.

   ```
   enable-mixed-case-support=true
   ```

   d. Run the following command to restart the Presto service.

   ```
   /opt/presto/bin/launcher_restart_handler.sh restart
   ```

Upgrading might encounter `Catalog does not exist` error

Applies to: 1.1.1, 1.1.2, 1.1.3, and 1.1.4 only

Fixed in: 2.0.0

Upgrading watsonx.data version 1.1.x to any higher version might give `Catalog does not exist` error in your development (DEV) and Cloud Pak for Data (CPD) environments.

Workaround: To access the missing catalogs after the upgrade, you must follow the instruction:

1. Log in to your machine where the development package or CPD environment is installed.
2. Run the following command to create `UpgradeFix.sh` file in the machine:

```
cat <<EOF > UpgradeFix.sh
#!/bin/bash

usage(){
cat << EOM

Usage:

    Syntax
    ------
    ./UpgradeFix.sh --platform [dev|cpd] --container [docker|podman] --namespace
[operand_namespace]

EOM
}

while true; do
    case "\$1" in
        --container ) LH_DOCKER_EXE="\$2"; shift ;;
        --platform ) PLATFORM="\$2"; shift ;;
        --namespace ) NAMESPACE="\$2"; shift ;;
        -h|--help ) usage; exit 0; shift ;;
        (*) break ;;
    esac
    shift
done

if [[ \$PLATFORM == "dev" ]]; then
    dockerexe=\${LH_DOCKER_EXE:-docker}
    \${dockerexe} exec ibm-lh-postgres psql -U admin -d ibm_lh_repo -c "INSERT INTO
metadata_properties (type, service_id, properties) SELECT 'database_properties' AS type,
database_id AS service_id, convert_to('connector.name=dvm' || chr(10) || 'connection-
url=jdbc:rs:dv://' || host || ':' || port || ';DatabaseType=dvm' || chr(10) ||
'connection-user=' || username || chr(10) || 'connection-password=' || password, 'UTF8')
AS properties FROM database WHERE database_type = 'dvm';"
```

```
    sleep 2
    \${dockerexe} exec ibm-lh-postgres psql -U admin -d ibm_lh_repo -c "INSERT INTO
metadata_properties (type, service_id, properties) SELECT 'database_properties' AS
type, database_id AS service_id, convert_to('connector.name=postgresql' || chr(10) ||
'connection-url=jdbc:postgresql://' || host || ':' || port || '/' || database_name
|| chr(10) || 'connection-user=' || username || chr(10) || 'connection-password=' ||
password || chr(10) || 'allow-drop-table=true', 'UTF8') AS properties FROM database
WHERE database_type = 'postgresql';"

    sleep 2
    \${dockerexe} exec ibm-lh-postgres psql -U admin -d ibm_lh_repo -c "INSERT
INTO metadata_properties (type, service_id, properties) SELECT 'database_properties'
AS type, database_id AS service_id, convert_to('connector.name=teradata' || chr(10)
|| 'connection-url=jdbc:teradata://' || host || '/DATABASE=' || database_name ||
',DBS_PORT=' || port || CASE WHEN ssl THEN ',SSLMODE=VERIFY-CA,ENCRYPTDATA=ON' ELSE
',SSLMODE=DISABLE,ENCRYPTDATA=OFF' END || chr(10) || 'connection-user=' || username ||
chr(10) || 'connection-password=' || password || chr(10) || 'allow-drop-table= true',
'UTF8') AS properties FROM database WHERE database_type = 'teradata';"

    sleep 2
    \${dockerexe} exec ibm-lh-postgres psql -U admin -d ibm_lh_repo -c "INSERT INTO
metadata_properties (type, service_id, properties) SELECT 'database_properties' AS type,
database_id AS service_id, convert_to('connector.name=db2' || chr(10) || 'connection-
url=jdbc:db2://' || host || ':' || port || '/' || database_name || CASE WHEN ssl
THEN ':sslConnection=true;' ELSE '' END || chr(10) || 'connection-user=' || username
|| chr(10) || 'connection-password=' || password || chr(10) || 'allow-drop-table=true',
'UTF8') AS properties FROM database WHERE database_type = 'db2';"

    sleep 2
    \${dockerexe} exec ibm-lh-postgres psql -U admin -d ibm_lh_repo -c "INSERT
INTO metadata_properties (type, service_id, properties) SELECT 'database_properties'
AS type, database_id AS service_id, convert_to('connector.name=snowflake' || chr(10)
|| 'connection-url=jdbc:snowflake://' || host || '.snowflakecomputing.com/?db=' ||
database_name || chr(10) || 'connection-user=' || username || chr(10) || 'connection-
password=' || password || chr(10) || 'allow-drop-table=true', 'UTF8') AS properties FROM
database WHERE database_type = 'snowflake';"
elif [[ \$PLATFORM == "cpd" ]]; then
    oc project \${NAMESPACE}
    pod_name=\$(oc get cluster | tr ' ' '\n' |grep ibm-lh-postgres-edb-)
    oc exec \${pod_name} -- psql -U postgres -d ibm_lh_repo -c "INSERT INTO
metadata_properties (type, service_id, properties) SELECT 'database_properties' AS type,
database_id AS service_id, convert_to('connector.name=dvm' || chr(10) || 'connection-
url=jdbc:rs:dv://' || host || ':' || port || ';DatabaseType=dvm' || chr(10) ||
'connection-user=' || username || chr(10) || 'connection-password=' || password, 'UTF8')
AS properties FROM database WHERE database_type = 'dvm';"

    sleep 2
    oc exec \${pod_name} -- psql -U postgres -d ibm_lh_repo -c "INSERT INTO
metadata_properties (type, service_id, properties) SELECT 'database_properties' AS
type, database_id AS service_id, convert_to('connector.name=postgresql' || chr(10) ||
'connection-url=jdbc:postgresql://' || host || ':' || port || '/' || database_name
|| chr(10) || 'connection-user=' || username || chr(10) || 'connection-password=' ||
password || chr(10) || 'allow-drop-table=true', 'UTF8') AS properties FROM database
WHERE database_type = 'postgresql';"

    sleep 2
    oc exec \${pod_name} -- psql -U postgres -d ibm_lh_repo -c "INSERT INTO
metadata_properties (type, service_id, properties) SELECT 'database_properties' AS
type, database_id AS service_id, convert_to('connector.name=teradata' || chr(10)
|| 'connection-url=jdbc:teradata://' || host || '/DATABASE=' || database_name ||
',DBS_PORT=' || port || CASE WHEN ssl THEN ',SSLMODE=VERIFY-CA,ENCRYPTDATA=ON' ELSE
',SSLMODE=DISABLE,ENCRYPTDATA=OFF' END || chr(10) || 'connection-user=' || username ||
chr(10) || 'connection-password=' || password || chr(10) || 'allow-drop-table= true',
'UTF8') AS properties FROM database WHERE database_type = 'teradata';"

    sleep 2
    oc exec \${pod_name} -- psql -U postgres -d ibm_lh_repo -c "INSERT INTO
metadata_properties (type, service_id, properties) SELECT 'database_properties' AS type,
database_id AS service_id, convert_to('connector.name=db2' || chr(10) || 'connection-
url=jdbc:db2://' || host || ':' || port || '/' || database_name || CASE WHEN ssl
THEN ':sslConnection=true;' ELSE '' END || chr(10) || 'connection-user=' || username
|| chr(10) || 'connection-password=' || password || chr(10) || 'allow-drop-table=true',
'UTF8') AS properties FROM database WHERE database_type = 'db2';"

    sleep 2
    oc exec \${pod_name} -- psql -U postgres -d ibm_lh_repo -c "INSERT INTO
metadata_properties (type, service_id, properties) SELECT 'database_properties' AS
type, database_id AS service_id, convert_to('connector.name=snowflake' || chr(10)
|| 'connection-url=jdbc:snowflake://' || host || '.snowflakecomputing.com/?db=' ||
database_name || chr(10) || 'connection-user=' || username || chr(10) || 'connection-
password=' || password || chr(10) || 'allow-drop-table=true', 'UTF8') AS properties FROM
```

```
database WHERE database_type = 'snowflake';"
else
    echo
    echo "error : Platform name is not correct"
    usage
    exit 1
fi
EOF
```

3. Run the following command to grant permission to run the file `UpgradeFix.sh`:

```
chmod 777  UpgradeFix.sh
```

4. To fix the issue in different environments, run the following commands:

   - Development environment:

   ```
   ./UpgradeFix.sh --platform dev --container [docker|podman]
   ```

   - CPD environment:

   ```
   ./UpgradeFix.sh --platform cpd --namespace [operand_namespace]
   ```

5. Go to watsonx.data user interface, navigate to **Infrastructure manager** and re-associate the missing catalog with engine.

Upgrading standalone watsonx.data might fail

Applies to: Standalone versions 1.0.0, 1.0.1, 1.0.2, 1.0.3

When you try to upgrade a standalone watsonx.data to a latest version 5.0.x/2.0.x, the upgrading might fail.

Workaround: To upgrade without failure, you must do the following:

1. Upgrade the standalone 1.0.x version of watsonx.data to 1.1.0/4.8.0 version of watsonx.data. See Upgrading watsonx.data from version 1.0.x to 1.1.x.
2. Upgrade the watsonx.data 1.1.0/4.8.0 version to any latest version of watsonx.data by following Upgrading watsonx.data from version 1.0.x or 1.1.x to 2.0.x.

Installation path directory with space

Applies to: 1.1.0 and later

When you run the `setup.sh` script for the watsonx.data Developer version, if the installation path has a directory that contains spaces, an error might occur.

For example, if the installation path is:

```
/Users/john/documents/userdata/Hybrid Data Management/Lakehouse/ibm/lh-dev/bin
```

the error might be similar to the following message:

```
./setup.sh: line 19: /Users/john/documents/userdata/Hybrid: no such file or directory
```

Workaround: Install the watsonx.data developer version into a directory that contains no space.

Known issues: Milvus

- Users are able to search data without specifying a partition in Milvus service
- `CreateIndex` permission denied due to policy sync delay of collection creator
- User role with `CreateCollection` L3 policy fails to create collection in Milvus
- Milvus unresponsive to queries
- Inaccurate row count after deletions in Milvus
- Potential data loss during batch insert of large data collection in Milvus

- Milvus service cannot be deleted using the delete icon from the user interface (UI) of watsonx.data developer edition in 1.1.3 version

Users are able to search data without specifying a partition in Milvus service

Applies to: 2.0.1

Fixed in: 2.0.2

Users can search data without specifying a partition in the `partition_names` field, bypassing intended access controls and potentially exposing sensitive data.

Workaround: You must explicitly specify the partitions that you are authorized to access in the `partition_names` field.

`CreateIndex` permission denied due to policy sync delay of collection creator

Applies to: 2.0.0 and later

When attempting to create an index immediately following the creation of a collection in Milvus, users with the `Viewer` or `User` role might encounter a `CreateIndex` permission denied error. Apart from the `CreateIndex` operation, this error can happen to other Milvus operations within a Collection like `Insert`, `CreatePartition`, etc.

Workaround: Wait for 5–10 seconds after collection creation to create index.

User role with `CreateCollection` L3 policy fails to create collection in Milvus

Applies to: 2.0.1 and later

Users with `User role` while creating collections in Milvus with pymilvus can fail when using the `ORM Connection` and `MilvusClient Connection` methods.

Workaround: You must follow the instructions:

- `ORM Connection`: The user requires both `DescribeCollection` and `CreateCollection` privileges granted in the L3 policy page. You must select all collections in a database while granting `DescribeCollection` privilege in the L3 policy through web console.
- `MilvusClient Connection`: Only `CreateCollection` privilege is necessary in the L3 policy page. However, the first attempt to create a collection will fail.
  1. Run the `create_collection` function once.
  2. Re-run the `create_collection` function again. This allows the policies to synchronise and the collection creation will succeed.

Milvus unresponsive to queries

Applies to: 1.1.3 and later

Milvus may not respond to queries when attempting to load collections or partitions that exceed available memory capacity. This occurs because all search and query operations within Milvus are executed in memory, requiring the entire collection or partition to be loaded before querying.

Workaround:

- Consider the memory limitations of your Milvus deployment and avoid loading excessively large collections or partitions.
- If Milvus becomes unresponsive to queries, employ the appropriate Milvus API to unload or release some collections from memory. An example using Python SDK: `collection.release()`.

Inaccurate row count after deletions in Milvus

Applies to: 1.1.3 and later

The `collection.num_entities` property might not reflect the actual number of rows in a Milvus collection after deletion operations. This property provides an estimate and may not account for deleted entities.

To get an accurate count of rows, execute a `count(*)` query on the collection. This provides an accurate count even after deletions.

Pymilvus syntax:

```
collection = pymilvus.Collection(...)
collection.query(expr='', fields=['count(*)'])
```

Potential data loss during batch insert of large data collection in Milvus

Applies to: 1.1.3 and later

Fixed in: 2.0.1

Potential data loss may occur when inserting large dataset (5 million vectors) through the Milvus batch insert API with a single final flush. A subset of rows might be missing from the ingested data.

Workaround:

- Flush the collection manually every 500,000 rows.
- Use the bulk insert API for data ingestion, see Insert Entities from Files. This is the recommended way to ingest large data sets.

Milvus service cannot be deleted using the delete icon from the user interface (UI) of watsonx.data developer edition in 1.1.3 version

Applies to: 1.1.3

Known issues: Presto (C++)

- Attempting to query Query History and Monitoring Management (QHMM) related tables using Presto (C++) engines might encounter errors
- Attempting to read Parquet v2 tables through Presto (C++) results in an error
- Presto (C++) does not support `NULLIF()` SQL function
- Presto (C++) fails to query an external partitioned table

Attempting to query Query History and Monitoring Management (QHMM) related tables using Presto (C++) engines might encounter errors

Applies to: 2.0.0

When you attempt to query QHMM related tables using Presto (C++) engines, you might encounter errors due to unsupported file formats. Presto (C++) supports only DWRF and Parquet v1 formats. You can not use Presto (C++) to query data or tables in other formats.

Workaround: You can switch to use Presto (Java) engines to query QHMM related tables.

Attempting to read Parquet v2 tables through Presto (C++) results in an error

Applies to: 2.0.0

When you attempt to read Parquet v2 tables through Presto (C++) that were created via **Data manager** in watsonx.data, it gives the following error:

```
Error in ZlibDecompressionStream::Next
```

Workaround: Presto (C++) currently does not support reading Parquet v2 tables. You must copy the data to a new table in v1 format to be compatible for reading using Presto (C++).

1. Set the session property to `PARQUET_1_0`:

   ```
   set session <catalog_name>.parquet_writer_version = 'PARQUET_1_0';
   ```

2. Run the following command to copy the data to a new table:

```
create table <catalog name>.<schema name>.<table name> as (select * from
<originaltablename>;
```

Presto (C++) does not support `NULLIF()` SQL function

Applies to: 2.0.0

Fixed in: 2.0.1

Presto (C++) does not appear to support the `NULLIF()` function, which is used to return `NULL` if two arguments are equal, otherwise returning the second argument. While Presto (Java) itself has support for `NULLIF()`, Presto (C++) seems to be missing this functionality.

Presto (C++) fails to query an external partitioned table

Applies to: 2.0.0 and later

When you query an external table with `CHAR` data type columns, the query fails to run. This issue occurs due to the limitation that Presto (C++) does not support `CHAR` data types.

Workaround: Change the `CHAR` data type column to `VARCHAR` data type.

Known issues: Query Optimizer

- Calculation error for `OPT_SORTHEAP` in Query Optimizer

Calculation error for `OPT_SORTHEAP` in Query Optimizer

Due to a calculation error in the configuration setting of Query Optimizer for the value of `OPT_SORTHEAP`, the performance of Query Optimizer might be affected.

Applies to: 2.0.0 and later

Workaround:

1. To resolve the calculation error for `OPT_SORTHEAP` in Query Optimizer, complete the following steps to update the configuration as `OPT_SORTHEAP= <initial_value>` to `OPT_SORTHEAP <initial_value>/20`.

   a. Set up the `PROJECT_CPD_INSTANCE` environment variable pointing to the namespace where watsonx.data is installed.

   ```
   export PROJECT_CPD_INSTANCE=<wxd_namespace>
   ```

   b. Edit the value of `OPT_SORTHEAP` to `OPT_SORTHEAP <initial_value>/20` by running the following command.

   ```
   oc edit db2uinstance lakehouse-oaas -n $PROJECT_CPD_INSTANCE
   ```

   c. Wait for sometime for the STATE to change to `Ready for lakehouse-oaas` and run the following command.

   ```
   watch "oc get db2uinstance  -n $PROJECT_CPD_INSTANCE"
   ```

Known issues: Semantic automation for data enrichment

- Schema not visible in the data enrichment list

Schema not visible in the data enrichment list

Applies to: 2.0.2 and later

In watsonx.data cluster with semantic automation registered, creating a new schema and then ingesting a table may result in the newly created schema not being immediately visible in the **Enrich data** tab.

Workaround: Refresh the **Enrich data** tab in watsonx.data. This will trigger a re-sync with IKC, and the newly created schema should become visible.

Known issues: Spark

- Spark application submission fails when Data Access Service (DAS) is enabled
- `ALTER TABLE` operation fails in Spark job submission
- Spark history UI shows temporary 502 error on startup

Spark application submission fails when DAS (Data Access Service) is enabled

Applies to: 2.0.0 and later

DAS does not currently support buckets or object storage that use HTTP endpoints.

Workaround: You can disable DAS or make sure that your buckets or object storage are configured with HTTPS endpoints.

`ALTER TABLE` operation fails in Spark job submission

Applies to: 2.0.1 and later

Spark jobs that creates a schema, table, and then attempt an `ALTER TABLE` operation may encounter an `authz.AccessControlException` due to insufficient permissions.

This occurs because, even though the schema and table creation are successful, the job tries to execute the `ALTER TABLE` operation before the metastore data is updated with the newly created schema and table details.

Workaround: To prevent access denied errors, you must provide a delay in time between each operations that involves creation of new schemas or tables within the same Python script.

Spark history UI shows temporary 502 error on startup

Applies to: 1.1.4 and later

Fixed in: 2.0.1

When attempting to access the Spark History UI immediately after starting the Spark History Server, users might encounter a temporary `502 Bad Gateway` error in their web browser.

Workaround: If you encounter the 502 error, reload the Spark history UI page after waiting 1-5 seconds. This should allow enough time for the server to become operational.

Known issues: SQL queries

- Skipping header lines during table creation
- String literal interpretation in Presto
- Presto (Java) queries with many columns and size exceeding default limit
- An unexpected error in parquet metadata reading occurs when running the queries on partitioned data
- Unrestricted access to SQL statements in worksheets
- `DROP TABLE` command on an Iceberg table does not remove folder and files from object storage

Skipping header lines during table creation

Applies to: 1.1.0 and later

Fixed in: 2.0.1

`skip.header.line.count` property is not supported by default in Presto (Java) and cannot be used in the `CREATE TABLE` statement to skip header lines when defining a table based on external data. However, the property can be used to skip a specific number of header lines when creating table from Hive, as the property is supported in Hive.

String literal interpretation in Presto

Applies to: 1.1.0 and later

Presto, by default interprets string literals as VARCHAR, unlike many other database systems that treat them as CHAR.

In Presto, string comparisons are performed on the actual characters present in the string, excluding trailing spaces. This can cause queries to return incorrect results when working with strings that may contain trailing spaces, as these spaces are not considered during comparison.

Presto (Java) queries with many columns and size exceeding default limit

Applies to: 1.1.0 and later

Presto (Java) queries involving multiple tables with a large number of columns (for example, 1000 columns per table or more) in the SELECT clause might encounter performance issues across all deployment environments.

The iterative optimizer times out when `max_reorder_joins` is set to 5 or higher (the default timeout is 3 minutes) and gives the following error:

```
The optimizer exhausted the time limit of 180000 ms
```

For queries exceeding the default `max-task-update-size` limit (16MB in Presto), you might observe a `TaskUpdate size exceeding this limit` error (the specific value of limit depends on the actual query).

Workaround:

- You can improve query performance by temporarily disabling the `reorder_joins` rule using the following session property:

  ```
  set session reorder_joins = false;
  ```

- Increase the `max-task-update-size` value in the config.properties file if the issue involves a `TaskUpdate size exceeding the limit` error and restart Presto.

  Example:

  ```
  experimental.internal-communication.max-task-update-size=64MB
  ```

For more information on how to configure properties, see API customization overview.

An unexpected error in parquet metadata reading occurs when running the queries on partitioned data

Applies to: 1.1.1

When you try to run the queries on partitioned data, you get the error `Unexpected error in parquet metadata reading after cache miss`.

Workaround: Disable the metastore versioned caching and header and footer caching. See Enhancing the query performance through caching for more information.

Unrestricted access to SQL statements in worksheets

Applies to: 1.1.0 and later

Fixed in: 2.0.1

SQL statements within worksheets can be shared with all users who have access to the instance. These statements could be viewed, edited, or deleted by any of these users.

`DROP TABLE` command on an Iceberg table does not remove folder and files from object storage

Applies to: 1.1.0 and later

`DROP TABLE` command on an Iceberg table removes the table metadata from the metastore, but the data is not removed from the object store.

Known issues: Web console

- Disabled **Next** button when selecting source files during data ingestion
- Logout option is unavailable when IAM is enabled
- Assigning user role access with Japanese browser language
- Missing data validation for Amazon S3 storage endpoints

Disabled Next button when selecting source files during data ingestion

Applies to: 2.0.0 and later

During data ingestion, when you select data sources from either bucket or system, the **Next** button might appear disabled. However, clicking these options will directly take you to the next page.

Workaround: Clicking option either from bucket or system in the **Select file(s)** page will directly take you to the next page. If the wrong option is chosen, click the **Cancel** button to return to the **Select file(s)** page. Review the options, and then click the desired option that will take you to the next page.

Logout option is unavailable when IAM is enabled

Applies to: 2.0.0 only

Fixed in: 2.0.1

When Identity and Access Management (IAM) is enabled for watsonx.data, the logout option directly from the watsonx.data dashboard might become unavailable.

Workaround: To securely log out when IAM is enabled, close the watsonx.data window to terminate your current session and log out from the IBM Cloud Pak for Data (CPD) platform by navigating to the CPD login page.

Assigning user role access with Japanese browser language

Applies to: 1.1.3 and later

Fixed in: 2.0.1

Users with browser language set to Japanese may encounter difficulties assigning access to components for **User** roles within the watsonx.data.

Workaround: Users can switch the browser language to English and assign **User** access to different components.

Missing data validation for Amazon S3 storage endpoints

Applies to: 1.1.0 and later

Currently, the user interface (UI) does not perform data validation for endpoints associated with the Amazon S3 storage type.

Known issues: Others

- Cache performance impact due to default size
- Missing `LhInstanceId` header in watsonx.data API calls
- Default MinIO bucket access through DAS is unavailable
- Concurrent Custom Resource (CR) reconciliation resulting in CPU/memory saturation
- Data that is imported from watsonx.data bucket fails

Cache performance impact due to default size

Applies to: 2.0.1 and later

Users are not able to patch the property `cache_alluxio_max_cache_size` using oc patch command. Hence, the default value for the maximum cache size in the data cache is not automatically adjusted to match the user environment persistent storage size when cache is enabled.

Workaround: You can utilize the customization API to manually update the maximum cache size value to a more appropriate value that aligns with their environment's persistent storage capacity. For more information, see "API customization overview" on page 8.

Missing `LhInstanceId` header in watsonx.data API calls

Applies to: 2.0.0

Fixed in:: 2.0.1

Direct API calls made to watsonx.data without including the `LhInstanceId` header can lead to API pod crashes. This issue only affects users integrating with watsonx.data APIs directly on Cloud Pak for Data (CPD), as the watsonx.data user interface automatically includes this header.

Workaround: Ensure all API calls to watsonx.data user interface to include the `LhInstanceId` header. This header should be set to the unique identifier of your watsonx.data instance on CPD.

Default MinIO bucket access through S3 proxy is unavailable

Applies to: 2.0.0 and later

Currently, it is not possible to access buckets stored in the default MinIO object storages created during instance provisioning using an S3 proxy functionality.

Concurrent Custom Resource (CR) reconciliation resulting in CPU/memory saturation

Applies to: 2.0.0 and later

Fixed in: 2.0.2

When all watsonx.data nodes reboot simultaneously, a large number of Custom Resource (CR) reconciliations for **`wxd`**, **`wxdaddon`**, or **`wxdengine`** components (like Presto and Milvus) might occur at the same time. This concurrent reconciliation can lead to CPU and memory saturation on the watsonx.data operator pod.

watsonx.data operator pod might become overloaded, impacting the overall functionality and performance.

Workaround: Lower the `ANSIBLE_FORKS` environment variable within the watsonx.data operator pod configuration. Reducing the value from 5 to 1 ensures that only one CR reconciliation happens at a time, preventing resource saturation.

Run the following command to modify the `ANSIBLE_FORKS` environment variable.

```
oc patch clusterserviceversion.operators.coreos.com/ibm-lakehouse-operator.v3.0.0  \
       --type json \
       -n <operator_ns> \
       -p '[
         {
           "op": "replace",
       "path": "/spec/install/spec/deployments/0/spec/template/spec/containers/0/env/5/
value",
           "value": "1"
         }
      ]'
```

Data that is imported from watsonx.data bucket fails

Applies to: 1.1.0 and later

When you run the import script (`import-bucket-data.sh`) to restore the watsonx.data bucket data, the system displays an error that the container `ibm-lh-lakehouse-minio` is not found.

```
+ oc exec -t ibm-lh-lakehouse-minio-7db7c6788f-g4r8 -n cpd-instance -- bash -c 'mc alias
set ibm-lh http://ibm-lh-lakehouse-minio-svc.cpd-instance.svc.cluster.local:9000 <access-
key> <secret_key> \
--config-dir=/tmp/.mc \
--insecure && mc alias set ibm-lh_backup https://s3.us-west-2.amazonaws.com/ <access-key>
<secret_key> \
--config-dir=/tmp/.mc --insecure'
error: unable to upgrade connection: container not found ("ibm-lh-lakehouse-minio")
```

Workaround: Delete the running MinIo pod by using the following command and rerun the import script.

```
oc delete -n $CPD_NAMESPACE $(oc get rs -o name -n $CPD_NAMESPACE | grep "ibm-lh-lakehouse-
minio")
```

Limitations: Access

- User access control is not supported for fully managed and self managed Spark engines
- Add external MinIO bucket to allowlist to establish connection from air-gapped watsonx.data cluster.

User access control is not supported for fully managed and self managed Spark engines

Applies to: 1.1.2 and later

The **Access control** tab is not supported for fully-managed or self-managed Spark engine. Administrators cannot carry out the access control operations for fully managed or self managed Spark engines.

Add MinIO bucket to allowlist to establish connection with watsonx.data

Applies to: 1.1.2 and later

To establish connection with watsonx.data, you must add the MinIO bucket URL to allowlist on your air-gapped cluster.

Limitations: Catalog, schema, and tables

- Unsupported special characters in schema and table creation
- Cross catalog schema creation anomaly in Presto
- Creating schemas in the root path of Ceph Object Storage gives an error
- Hive does not support `json` data that starts with array
- Hive catalog table creation by using `external_location` fails due to wrong placement of file
- Table creation fails if the column names differ only by spaces
- Using special characters in schema, table, or column names

Unsupported special characters in schema and table creation

Applies to: 1.1.0 and later

The following special characters are not supported while creating schemas and tables:

Schemas (Hive and Iceberg): `{`, `[`, `(`, and `)`.

Tables (Hive): `{`, `(`, `[`, and `)`. (Creation of tables within a schema name that starts with the special character `` `@` `` shall result in an error).

Tables (Iceberg): `$`, `@`, `{`, `[`, `)`, and `(`.

Cross catalog schema creation anomaly in Presto

Applies to: 1.1.0 and later

An anomaly exists in schema creation for Hive and Iceberg catalogs managed by Presto. When using a common Hive Metastore Service for multiple catalogs (Example, an Iceberg catalog and a Hive catalog, or two Iceberg or Hive catalogs), creating a schema in one catalog might create it in a wrong catalog. This occurs if the location specified during schema creation belongs to a different catalog than intended.

Workaround: You must always explicitly provide the correct storage path associated with the target catalog when using `CREATE SCHEMA` statements in Presto. This ensures the schema is created in the desired location.

Creating schemas in the root path of Ceph Object Storage gives an error

Applies to: 1.1.0 and later

Due to a bug in IBM Storage Ceph 5/6 and Red Hat Ceph Storage 4/5/6, if you are creating schema in the root path of one of the Ceph Object Storage in the watsonx.data, it gives you the following error message.

```
Executing query failed with error: com.facebook.presto.spi.PrestoException: Failed to
create schema. Check the credentials, permissions and storage path for the bucket. Make
sure that the bucket is registered with wxd and retry.
```

Solution: You can upgrade your IBM Storage Ceph and Red Hat Ceph Storage to 7.0z1 and 7.1 versions respectively.

Workaround: If you are still using the older IBM Storage Ceph 5/6 and Red Hat Ceph Storage 4/5/6 versions, you must do the following:

When you create a schema in Ceph Object Storage, a pseudo-directory must be created prior to creating schema in watsonx.data.

Run the following command to use the `s5cmd` S3 client to create a pseudo-directory and insert an empty file into it:

```
touch a
s5cmd --endpoint-url s3.ceph.example.com cp a s3://watsonx/mycatalog/myschema/
```

The copy command puts an empty file in the `/mycatalog/myschema` pseudo-directory.

Use the newly created pseudo-directory as the path for creating schema in watsonx.data web console.

Hive does not support `json` data that starts with array

Applies to: 1.1.0 and later

Hive does not support `json` data that starts with array.

Hive catalog table creation by using `external_location` fails due to wrong placement of file

Applies to: 1.1.0 and later

Hive catalog table creation by using `external_location` fails when the file is placed in the root of the bucket.

Table creation fails if the column names differ only by spaces

Applies to: 1.1.0 and later

When you create a table from a data file by using the watsonx.data web console, the column names must be unique. Due to this limitation, if a CSV data file has column names that differ only by "spaces" for example, `Cash Flow per Share` and `CashFlowPerShare`, then these columns are considered to have the same names and table creation fails.

Using special characters in schema, table, or column names

Applies to: 1.1.0 and later

It is recommended to not use special characters such as question mark (?), hyphen (-), or asterisk (*) in table, column names and schema names. Though these special characters are supported and tables, columns, and schemas can be created, using them might cause issues when running the `INSERT` command. In db2, the special characters listed are not supported.

Limitations: Database and storage

- DB2 connector cannot access views created using external tools
- Transactions not supported in unlogged Informix databases
- LDAP authentication is not supported for Teradata connector
- Netezza® Performance Server INSERT statement limitation
- Unsupported Db2 operations
- Handling Null Values in **`Elasticsearch`**
- Loading Nested JSON with **`Elasticsearch`**
- Db2 does not support `CREATE VIEW` statement for a table from another catalog
- Netezza Performance Server does not support `CREATE VIEW` statement for a table from another catalog
- Incorrect table name syntax in DB2 `CREATE VIEW` statements

DB2 connector cannot access views created using external tools

Applies to: 2.0.1 and later

The DB2 connector in watsonx.data currently allows access to the views created through the watsonx.data instance or DBeaver. However, accessing views created with other tools such as Data Manager Console (DMC), Db2 command line (DB2 cmd), or any other third-party tools is not supported.

Transactions not supported in unlogged Informix databases

Applies to: 1.1.4 and later

In watsonx.data, when attempting to execute queries with transactional implications on unlogged Informix databases, queries will fail. This is because unlogged Informix databases, by design, do not support transactions.

LDAP authentication is not supported for Teradata connector

Applies to: 1.1.0 and later

The watsonx.data Teradata connector does not currently support LDAP (Lightweight Directory Access Protocol) for user authentication.

Netezza Performance Server INSERT statement limitation

Applies to: 1.1.0 and later

Netezza Performance Server currently does not support inserting multiple rows directly into a table using `VALUES` clause. This functionality is limited to single-row insertions. Refer to the official Netezza Performance Server documentation for details on the `INSERT` statement.

The following example using `VALUES` for multiple rows is not supported:

```
INSERT INTO EMPLOYEE VALUES (3,'Roy',45,'IT','CityB'),(2,'Joe',45,'IT','CityC');
```

Workaround: Use a subquery with `SELECT` and `UNION ALL` to construct a temporary result set and insert it into the target table.

```
INSERT INTO EMPLOYEE SELECT * FROM(SELECT 4,'Steve',35,'FIN','CityC' UNION ALL SELECT
5,'Paul',37,'OP','CityA') As temp;
```

Unsupported Db2 operations

Applies to: 1.1.0 and later

watsonx.data currently does not support the `ALTER TABLE DROP COLUMN` operation for Db2 column-organized tables.

Note: By default, Db2 instances create tables in column-organized format.

watsonx.data does not support creating row-organized tables in Db2.

Handling Null Values in `Elasticsearch`

Applies to: 1.1.0 and later

`Elasticsearch` connector requires explicit definition of index mappings for fields to handle null values when loading data.

Loading Nested JSON with `Elasticsearch`

Applies to: 1.1.0 and later

`Elasticsearch` connector requires users to explicitly specify nested JSON structures as arrays of type ROW for proper loading and querying. To process such structures, use the UNNEST operation.

Db2 does not support `CREATE VIEW` statement for a table from another catalog

Applies to: 1.1.0 and later

For Db2, you can create the view for a table only if that table is in the same catalog and the same schema.

Netezza Performance Server does not support `CREATE VIEW` statement for a table from another catalog

Applies to: 1.1.0 and later

For Netezza Performance Server, you can create the view for a table only if that table is in the same catalog and the same schema.

Incorrect table name syntax in DB2 `CREATE VIEW` statements

Applies to: 2.0.0

Fixed in: 2.0.1

When creating views in DB2, using specific table name formats within the `CREATE VIEW` statement may encounter the following errors:

- Fully qualified table name: Specifying the catalog, schema, and table name as `create view my_view as select * from my_catalog.my_schema.my_table;` might result in error.
- Unqualified table name: Using only the table name without any qualifiers as `create view my_view as select * from my_table;` might also result in error.

Workaround:

To ensure successful view creation, use a schema-qualified table name within the `CREATE VIEW` statement.

1. Use the USE statement to set the active schema for the session.

```
USE my_catalog.my_schema;
```

2. Reference the table name qualified by the schema in the `CREATE VIEW` statement.

```
create view my_view as select * from my_schema.my_table;
```

Limitations: Engine

- Back up your data to prevent data loss
- Presto (Java) needs precision of the DECIMAL column to be within a valid range
- Unable to create views in Presto
- HMS and Presto (Java) log level from Default Error level to Debug level

Back up your data to prevent data loss while working with VS Code development environment - Spark Labs

Applies to: 2.0.0 and later

As Spark labs are ephemeral in nature, you must back up the data stored periodically to prevent potential data loss during upgrades or a Spark master crash.

Presto (Java) needs precision of the DECIMAL column to be within a valid range

Applies to: 1.1.0 and later

Presto (Java) needs precision of the DECIMAL column in the PostgreSQL table creation statement to be within a valid range.

Unable to create views in Presto

Applies to: 1.1.0 and later

Presto (Java) describes a view in a mapped database as a **TABLE** rather than a **VIEW**. This is apparent to JDBC program connecting to the Presto (Java) engine.

HMS and Presto (Java) log level from Default Error level to Debug level

Applies to: 1.1.0 and later

watsonx.data console does not support changing HMS and Presto (Java) log level from Default Error level to Debug level.

Workaround:

- Run the following curl command to change the log level inside the HMS pods:

```
curl -k -X POST 'https://localhost:8281/v1/hms/loglevel' -H 'Content-Type: application/
json' -d '{"log-level": "DEBUG"}'
```

- Run the following curl command to change the log level inside the Presto pods:

```
curl --location 'https://<host>:8481/v1/lh_engine/change_configuration' \
-k --header 'secret: $LH_INSTANCE_SECRET' \
--header 'Content-Type: application/json' \
--data '{
    "type":"loglevel",
    "value":"info",
    "restart":true
}'
```

Limitations: Presto (C++)

- Limitations - Presto (C++)

Limitations - Presto (C++)

Applies to: 2.0.0 and later

- Presto (C++) engine currently does not support database catalogs.
- Parquet and DWRF are the only file formats supported.
- Hive connector is supported.
- Default Iceberg table has read only support for Parquet v1 format.
- TPC-H/TPC-DS queries are supported.
- `DELETE FROM` and `CALL` SQL statements are not supported.
- `START`, `COMMIT`, and `ROLLBACK` transactions are not supported.
- Data types `CHAR`, `TIME`, and `TIME WITH TIMEZONE` are not supported. These data types are subsumed by `VARCHAR`, `TIMESTAMP`, and `TIMESTAMP WITH TIMEZONE`.
  - `IPADDRESS`, `IPPREFIX`, `UUID`, `kHYPERLOGLOG`, `P4HYPERLOGLOG`, `QDIGEST`, and `TDIGEST` are not supported.

   - VARCHAR supports only a limited length. `Varchar(n)` with a maximum length bound is not supported.
   - TIME and TIME WITH TIMEZONE is supported in community development.
   - TIMESTAMP columns in Parquet files cannot be read
- Scalar functions:
   - `IPFunctions`, `QDigest`, `HyperLogLog`, and Geospatial internationalization are not supported.
- Aggregate functions:
   - QDigest, Classification metrics, and Differential entropy are not supported
- S3 and S3 compatible file systems (both read and write) are supported

Limitations: SQL queries

- Incomplete information on column length in SHOW COLUMNS output
- Timestamp with timezone handling limitation in CREATE/ALTER TABLE
- Presto SQL operations with Spark 3.3 and Iceberg timestamp data
- Alter column is not supported for Hive and Iceberg catalogs

Incomplete information on column length in SHOW COLUMNS output

Applies to: 1.1.0 and later

The SHOW COLUMNS query in Presto currently provides information about columns including name, data type, additional details (extra), and comments. This issue highlights that the existing functionality lacks details about the length of character-based data types (CHAR and VARCHAR). While some connectors return the actual length defined during table creation, others might provide a default value or no information at all.

To address this limitation, three new columns have been added to the SHOW COLUMNS output:

- Scale: Applicable to DECIMAL data type, indicating the number of digits after the decimal point.
- Precision: Applicable to numerical data types, specifying the total number of digits. (Default: 10)
- Length: Intended for CHAR and VARCHAR data types, representing the maximum number of characters allowed.

Current Limitations:

- The reported length in the "Length" column might not always reflect the actual size defined in the table schema due to connector limitations.
- Connectors that don't provide length information will display a default value or null depending upon connector.

Timestamp with timezone handling limitation in CREATE/ALTER TABLE

Applies to: 1.1.4 and later

Presto (Java) previously had a limitation where CREATE TABLE and ALTER TABLE statements incorrectly treated timestamps with timezones as simple timestamps. Since these are distinct data types, this could lead to errors. To address this issue, the functionality of mapping timestamps with timezones has been disabled.

Workaround: You need to modify CREATE TABLE and ALTER TABLE statements to use plain timestamps (without timezone information).

Presto SQL operations with Spark 3.3 and Iceberg timestamp data

Applies to: 2.0.0 and later

When data containing `timestampz` is ingested using Spark, Presto queries on these tables fail with the following error `Iceberg column type timestamptz is not supported`.

Workaround: To ensure interoperability between Spark and Presto for datasets containing timestampz, you must use Spark 3.4 applications with the configuration `spark.sql.timestampType` set to `TIMESTAMP_NTZ`.

Alter column is not supported for Hive and Iceberg catalogs

Applies to: 1.1.0 and later

`ALTER TABLE` operations that change a column's type to an incompatible type (for example, from STRING to MAP) are not supported for Hive and Iceberg catalogs.

Limitations: Others

- Spark ingestion currently does not support special characters like quotation marks, back ticks, and parentheses for partitioned table column names.
- IBM Knowledge Catalog integration does not support row-level filtering
- Cut or Copy icons are still enabled even after the action is performed
- `No space left on device` error occurs
- Using the S3 Select Pushdown option

Spark ingestion currently does not support special characters like quotation marks, back ticks, and parentheses for partitioned table column names.

Applies to: 2.0.1 only

IBM Knowledge Catalog integration does not support row-level filtering

Applies to: 1.1.2 and later

After IBM Knowledge Catalog integration, data-masking rules are enforced in watsonx.data. But row-filtering rules are not applied, which can cause the rows to be visible and accessible.

Cut or Copy icons are still enabled even after the action is performed

Applies to: 1.1.0 and later

When you select text in the Query workspace, Cut and Copy icons are enabled. The Cut and Copy icons remain enabled after performing the actions. The Cut and Copy icons must be disabled when no text is selected after the action is completed.

Workaround: Clipboard settings in the Advanced preferences option of the Firefox browser must be set to `true`. Following are the list of Clipboard settings:

- `dom.event.clipboardevents.enabled`
- `dom.event.asyncClipboard.clipboardItem`
- `dom.event.asyncClipboard.readText`
- `dom.event.testing.asyncClipboard`

`No space left on device` error occurs

Applies to: 1.1.0 and later

When you run queries on Presto (Java) while caching is enabled, a `No space left on device` error is displayed.

Workaround: To resolve this error, log in to the cache directory and delete all entries in it.

Using the S3 Select Pushdown option

Applies to: 1.1.0 and later

The S3 Select Pushdown option allows you to filter the data at the source and retrieve just the subset of data that you need. In watsonx.data, this option is disabled by default. You can enable the S3 Select Pushdown option (`s3_select_pushdown_enabled`) by using the API. Currently, the S3 Select Pushdown option is supported only on IBM Storage Ceph and Amazon Web Services (AWS).

You can select only non-database catalogs

Applies to: 1.1.2 and later

When integrating with IBM Knowledge Catalog, you can select only non-database catalogs. Database catalogs are not supported in watsonx.data.

For more information, see Integrating with IBM Knowledge Catalog.

Chapter 2. Setting up watsonx.data developer edition

You can do the following to provision and manage watsonx.data developer edition.

watsonx.data Developer edition

Planning

Use the following information to plan your watsonx.data developer edition.

watsonx.data Developer edition

Prerequisites for watsonx.data installation on Mac with Apple silicon

To install IBM watsonx.data on Mac machine with Apple silicon, first install the prerequisites by either using docker with Colima or by using docker with Docker Desktop.

watsonx.data Developer edition

Note: Docker Desktop requires a paid license, while Colima uses an open source license.

Option 1: Installing the prerequisites by using Colima with Rosetta

Open the Mac terminal and complete the following procedure:

1. Install Brew: https://brew.sh/.
2. Install Docker: https://formulae.brew.sh/formula/docker.

   ```
   brew install docker
   ```
3. Install Colima: https://formulae.brew.sh/formula/colima.

   ```
   brew install colima
   ```
4. Install Rosetta:

   ```
   softwareupdate --install-rosetta
   ```

Use the following commands to set up the Colima machine:

1. Run the following command:

   ```
   colima start machine-1 -a aarch64 --cpu 4 --memory 8 --disk 100 --vm-type=vz --vz-rosetta
   ```

   Note: If you experience problems to start the Colima machine (just after Rosetta installation), restart your computer and run the start command again.
2. Run the following command to check whether the machine is created and is running:

   ```
   colima list
   ```
3. Run the following command to shut off the machines that are not in use:

   ```
   colima stop <machine name>
   ```

   Note: Only machine-1 must be running.
4. Run the following command to start the machine if it is not running:

   ```
   colima start machine-1
   ```

   Note: Creating a Colima machine must be done only one time.

Option 2: Installing prerequisites by using Docker and enabling Rosetta support with Docker Desktop

Complete the following procedure:

1. Install Rosetta by the following command:

```
softwareupdate --install-rosetta
```

2. Install Docker Desktop. For more information, see Install Docker Desktop on Mac.
3. Enable Rosetta in Docker Desktop by the following steps:
   a. Open Docker Desktop.
   b. Go to 'Settings'.
   c. Go to 'Features in Development'.
   d. Check 'Use Rosetta for x86/amd64 emulation on Apple Silicon'.

Prerequisites for watsonx.data installation on Mac with Intel chip

Complete the prerequisites to install IBM watsonx.data on Mac machines with Intel chip.

watsonx.data Developer edition

1. Run the following command to install Brew. For more information, see Homebrew page.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/
install.sh)"
```

2. Run the following command to install Docker. For more information, see Homebrew Formula for Docker.

```
brew install docker
```

3. Run the following command to install Colima. For more information, see Homebrew Formula for Colima.

```
brew install colima
```

4. Run the following command to start Colima.

```
colima start
```

5. Run the following command to ensure that Colima is running.

```
colima status
```

Installing

Use the following information to install your watsonx.data developer edition.

watsonx.data Developer edition

Installing the watsonx.data developer version

IBM watsonx.data developer version is an entry-level version for the developer and partner community. The developer version offers a set of containers on a suitable host machine at the same release level as the Enterprise version, with restricted features.

watsonx.data Developer edition

Before you begin

- Ensure that you have your entitlement key to access the IBM Entitled Registry. After you get the entitlement key from the container software library, you can log in to the registry with the key and pull the runtime images to your local machine.

  Note: You must purchase IBM watsonx.data to get the entitlement key.

- For optimal security and stability, it is highly recommended to install the watsonx.data developer version as a non-root user. Also, ensure to establish connections to the remote systems by using SSH instead of using the `su` command.
- Ensure to meet the following system requirements and install the most recent version of `Docker` or `Podman` on your system.

Operating system	x86-64	Docker and Podman installation instructions
Linux®	✓	Docker Podman
Windows	✓	Docker Podman
Mac OS x86	✓	Docker Podman

  Note:

  - Ensure to meet the prerequisites for installing watsonx.data developer version on Mac with Apple silicon.
  - Ensure to meet the prerequisites for installing watsonx.data developer version on Mac with Intel chip.

- For SUSE Linux, `podman` is only available for version SLES 15.4. Upgrade the system first (`zypper dist-upgrade`) before installing the dependencies that are not provided but are needed for the installation of `docker` or `podman`.
- For Docker:
  - `sysuser-shadow` in SLES 12.5
  - `catatonit` in SLES 15.5
- For Podman:
  - `fuse-overlayfs` in SLES 15.4

  Note: Packages can be downloaded from the official site.

  Important: Ensure to add the `podman-plugins` for the DNS server of Podman network to work properly.

```
yum install -y podman-plugins
```

  For SUSE, install cni-plugin-dnsname instead.

  If you add the `podman-plugins` after you start the watsonx.data service, delete the Podman network `ibm-lh-network` and restart the watsonx.data service.

  Tip: Podman provides a Docker-compatible command-line front end. You can alias the Docker CLI with the `alias docker=podman` shell command.

Procedure

1. Set up the installation directory and environment variables.

a) Set up the work directory.

```
mkdir <install_directory>
cd <install_directory>
```

b) Set the environment variables.

```
export LH_ROOT_DIR=<install_directory>
export LH_RELEASE_TAG=latest
export IBM_LH_TOOLBOX=cp.icr.io/cpopen/watsonx-data/ibm-lakehouse-toolbox:$LH_RELEASE_TAG
export LH_REGISTRY=cp.icr.io/cp/watsonx-data
export PROD_USER=cp
export IBM_ENTITLEMENT_KEY=<your_IBM_entitlement_API_key>
export IBM_ICR_IO=cp.icr.io
```

Note: Use the following table to identify the `LH_RELEASE_TAG` value:

Table 7. Release tags with Cloud Pak for Data versions

Cloud Pak for Data version	Service instance version
5.0.2	`v2.0.2` (`latest`)
5.0.1	`v2.0.1`
5.0.0	`v2.0.0`
4.8.5	`v1.1.4`
4.8.4	`v1.1.3`
4.8.3	`v1.1.2`
4.8.1	`v1.1.1`
4.8.0	`v1.1.0`

The `latest` tag is set to `v2.0.2`.

Note: For air gap installation, ensure to mirror the images to a private registry before you set the environment variables:

```
export IBM_LH_TOOLBOX=<Private_registry>/cpopen/watsonx-data/ibm-lakehouse-
toolbox:$LH_IMAGE_TAG
export LH_REGISTRY=<Private_registry>/cp/watsonx-data
export PRIVATE_REGISTRY_USER=<User login to Private registry>
export PRIVATE_REGISTRY_PASSWORD=<User password to login private registry>
```

If you are using Docker, run the following command:

```
export DOCKER_EXE=docker
```

If you are using Podman, run the following command:

```
export DOCKER_EXE=podman
```

2. Pull the watsonx.data developer package and copy it to the host system:

```
$DOCKER_EXE pull $IBM_LH_TOOLBOX
id=$($DOCKER_EXE create $IBM_LH_TOOLBOX)
$DOCKER_EXE cp $id:/opt - > /tmp/pkg.tar
$DOCKER_EXE rm $id
id=
```

3. Extract the watsonx.data developer version `pkg.tar` file in to the `/tmp` directory. Verify that the checksum is correct by comparing the checksum in `bom.txt` and the `cksum` command output. For example:

```
tar -xf /tmp/pkg.tar -C /tmp
cat /tmp/opt/bom.txt
```

```
cksum /tmp/opt/*/*
tar -xf /tmp/opt/dev/ibm-lh-dev-*.tgz -C $LH_ROOT_DIR
```

4. Authenticate with the registry:

   ```
   $DOCKER_EXE login ${IBM_ICR_IO} \
   --username=${PROD_USER} \
   --password=${IBM_ENTITLEMENT_KEY}
   ```

   Note: For air gap installation, run the following command to authenticate to the private registry:

   ```
   $DOCKER_EXE login ${LH_REGISTRY} \
   --username=${PRIVATE_REGISTRY_USER} \
   --password=${PRIVATE_REGISTRY_PASSWORD}
   ```

5. Run the setup script.

   ```
   $LH_ROOT_DIR/ibm-lh-dev/bin/setup --license_acceptance=y --runtime=$DOCKER_EXE
   ```

   Use the `--password` command line argument to set the password of the default `ibmlhadmin` user for access to the UI interface, API interface and `presto-cli` interface. If the password is not set, then the default password is 'password'. You can add more users and change the password post setup. For more information, see “Developer package utilities” on page 66

6. Start the containers.

   ```
   $LH_ROOT_DIR/ibm-lh-dev/bin/start
   ```

7. To open the watsonx.data console, go to `https://localhost:<https_port>`, where `<https_port>` is the port number that you entered during setup. For remote access, replace `localhost` with the machine hostname. For example,`https://lh-demo-01:9443/`

   You need to enter the username `ibmlhadmin` and the related password. If the password was not chosen at the time of setup, the default password is 'password'. For more information, see “Developer package utilities” on page 66

8. To check the status of all the containers, run the following command:

   ```
   $LH_ROOT_DIR/ibm-lh-dev/bin/status --all
   ```

9. To stop all the containers, run the following command:

   ```
   $LH_ROOT_DIR/ibm-lh-dev/bin/stop
   ```

   Note: To stop or start a specific container, run the following commands:

   ```
   $LH_ROOT_DIR/ibm-lh-dev/bin/stop_service <container_name>
   ```

   ```
   $LH_ROOT_DIR/ibm-lh-dev/bin/start_service <container_name>
   ```

   where `<container_name>` is the service name. It is the string that is displayed under the `NAMES` column when you run the `docker ps` command.

   For example:

   ```
   $LH_ROOT_DIR/ibm-lh-dev/bin/stop_service lh-hive-metastore
   ```

   ```
   $LH_ROOT_DIR/ibm-lh-dev/bin/start_service lh-hive-metastore
   ```

Upgrading

Use the following information to upgrade your existing watsonx.data developer edition.

IBM watsonx.data developer edition

Upgrading watsonx.data in developer edition

An instance administrator can upgrade IBM watsonx.data developer edition from version 1.0.x or 1.1.x to 2.0.x and 2.0.x to a later 2.0.x refresh.

watsonx.data Developer edition

Procedure

Complete the following steps to upgrade watsonx.data developer edition:

1. Set up the environment variables.

```
export LH_ROOT_DIR=<install_directory>
export LH_RELEASE_TAG=latest
export IBM_LH_TOOLBOX=cp.icr.io/cpopen/watsonx-data/ibm-lakehouse-toolbox:$LH_RELEASE_TAG
export LH_REGISTRY=cp.icr.io/cp/watsonx-data
export PROD_USER=cp
export IBM_ENTITLEMENT_KEY=<your_IBM_entitlement_API_key>
export IBM_ICR_IO=cp.icr.io
```

Note: Use the following table to identify the `LH_RELEASE_TAG` value:

Table 8. Release tags with Cloud Pak for Data versions

Cloud Pak for Data version	Service instance version
5.0.2	`v2.0.2` (latest)
5.0.1	`v2.0.1`
5.0.0	`v2.0.0`
4.8.5	`v1.1.4`
4.8.4	`v1.1.3`
4.8.3	`v1.1.2`
4.8.1	`v1.1.1`
4.8.0	`v1.1.0`
4.7.4	`v1.0.3`
4.7.3	`v1.0.3`

The `latest` tag is set to `v2.0.2`.

Note: For air gap installation, ensure to mirror the images to a private registry before you set the environment variables:

```
export IBM_LH_TOOLBOX=<Private_registry>/cpopen/watsonx-data/ibm-lakehouse-
toolbox:$LH_IMAGE_TAG
export LH_REGISTRY=<Private_registry>/cp/watsonx-data
export PRIVATE_REGISTRY_USER=<User login to Private registry>
export PRIVATE_REGISTRY_PASSWORD=<User password to login private registry>
```

If you are using Docker, run the following command:

```
export DOCKER_EXE=docker
```

If you are using Podman, run the following command:

```
export DOCKER_EXE=podman
```

2. Run the following command to stop all containers:

```
./stop.sh
```

3. Pull the latest bundle for upgrade.

```
rm -rf /tmp/opt/dev/ibm-lh-dev-*
$DOCKER_EXE pull $IBM_LH_TOOLBOX
id=$($DOCKER_EXE create $IBM_LH_TOOLBOX)
$DOCKER_EXE cp $id:/opt - > /tmp/pkg.tar
$DOCKER_EXE rm $id
id=
```

4. Run the following command to extract the watsonx.data developer version `pkg.tar` file into the `/tmp` directory.

```
tar -xf /tmp/pkg.tar -C /tmp
cat /tmp/opt/bom.txt
cksum /tmp/opt/*/*
tar -xvf /tmp/opt/dev/ibm-lh-dev-*.tgz -C $LH_ROOT_DIR
```

5. Authenticate with the registry:

```
$DOCKER_EXE login ${IBM_ICR_IO} \
--username=${PROD_USER} \
--password=${IBM_ENTITLEMENT_KEY}
```

6. Run the setup script:

```
$LH_ROOT_DIR/ibm-lh-dev/bin/setup --license_acceptance=y --runtime=$DOCKER_EXE
```

7. Run the following command to restart the containers:

```
$LH_ROOT_DIR/ibm-lh-dev/bin/start
```

Note: If you cannot save the queries or cannot see the saved queries, run the following commands in developer-installed VM terminal and then refresh the page.

```
podman exec -it ibm-lh-postgres sh
export PGPASSWORD=$POSTGRES_PASSWORD
psql -h ibm-lh-postgres-svc -p 5432 -U $POSTGRES_USER -w -d ibm_lh_repo
ALTER TABLE stored_queries ALTER COLUMN created_on drop default;
ALTER TABLE stored_queries ALTER COLUMN created_on TYPE bigint USING EXTRACT(EPOCH FROM
created_on);
ALTER TABLE stored_queries ALTER COLUMN created_on set DEFAULT EXTRACT(EPOCH FROM now());
\q
exit
```

Administering

Use the following information to manage and maintain your watsonx.data developer edition.

watsonx.data Developer edition

Connecting to external object stores over `https`: Developer edition

IBM watsonx.data needs a valid signer certificate to establish a connection with the object stores secured with `https`.

watsonx.data Developer edition

Procedure

1. Use the `cert-mgmt` command to import the certificates from the object store server.

```
bin/cert-mgmt --op=add --host=<endpoint> --port=443
```

2. Restart the developer edition to reload the containers with the updated truststore.

```
bin/stop
bin/start
```

What to do next

For adding storages, see Adding a storage-catalog pair.

Exposing the MinIO ports and generating credentials

To demonstrate or test the functioning of IBM watsonx.data, use the embedded MinIO object store to load or store data. To access the embedded MinIO object store, you must expose the ports and generate credentials.

watsonx.data Developer edition

About this task

The embedded MinIO Object Store ports are not exposed by default. To access the embedded MinIO Object Store, you can run a script to expose the MinIO ports. Ports and MinIO credentials are generated when the script is run. Use the generated credentials to connect to the Object Store MinIO.

Note: The out-of-the-box MinIO object storage is provided for exploratory purposes only. It does not have all security features and is not configured to provide high-speed data access. You should register your own s3 storage that meet your security and performance requirements.

Use the `expose-minio` command to expose the embedded object store for use during development. For example, you can import data files into the storages directly by using the S3 API or compatible utilities outside the watsonx.data Developer edition's host.

The script is available in the `<install dir>/ibm-lh-dev/bin` path.

Procedure

1. Run the following command to expose the MinIO ports and generate credentials:

   ```
   ibm-lh-dev/bin/expose-minio
   ```

   The `ibm-lh-dev/bin/expose-minio` script starts the embedded MinIO Object Store container to expose the port details. Thus, the client programs outside the developer edition's host can access these client details. It also generates the credentials to connect to that Object Store that the client programs access outside the developer edition's host. It also generates the credentials to connect to that Object Store.

   ```
   ibm-lh-dev/bin/expose-minio
   FYI: LH_RUN_MODE is set to diag
   867c0daf36900cd05f0d06af3723b275ccdf933418cc78fb3a9a9b10c72f579a

   Minio credentials:
       username: 8518db917ed5260803e00685
       password: d0622ed45c07a8d125e4fe40
   Ports:
       S3 endpoint port: 9000
       Minio console port: 9001
   ```

   Note: Use the MinIO S3 endpoint port with any S3 compatible tools or use the console port on a web browser to navigate to the MinIO web console. The default URL is `http://<hostname>:<MinIO_console_port>`

2. Run the following command to reset the credentials and stop exposing the ports.

   ```
   bin/start_service ibm-lh-minio
   ```

Exposing Hive metastore ports

To facilitate connection from external applications (services outside of docker or Podman), such as the integration with Db2, Netezza, and Spark to watsonx.data, you must expose the Hive metastore port details.

watsonx.data Developer edition

About this task

The Hive metastore port details are not exposed by default outside the watsonx.data Developer edition's host. Run a script to expose Hive metastore server port details to outside network. This enables existing Db2, Netezza users to experiment with IBM watsonx.data integration feature by using the developer edition.

The `expose-hms` command helps to expose the Hive metastore port details outside the watsonx.data Developer edition's host.

The script is available in the `<install dir>/ibm-lh-dev/bin` path.

Procedure

1. Run the following command to go to the `bin` folder:

   ```
   cd <install dir>/ibm-lh-dev/bin
   ```
2. Start all the containers present in the `bin` folder by using the following command:

   ```
   ./start
   ```
3. Run the following command to expose the Hive metastore port details:

   ```
   ./expose-hms
   ```
4. The Hive metastore console port gets listed. To view the port details, run the following command:

   ```
   docker ps -a
   ```

   Note: If you are using Podman, use `podman ps -a` to view the port details.

Using a custom TLS certificate

When you install IBM watsonx.datadeveloper version, a self-signed key and certificate are generated. All HTTPS clients, including web browsers might not trust the self-signed certificates. It is recommended to replace the self-signed certificate with your own certificate..

watsonx.data Developer edition

Before you begin

Install IBM watsonx.data developer version before adding your certificates.

About this task

Complete the following steps to add your TLS key and certificate:

Procedure

1. Create a subdirectory folder `external-tls` in the `ibm-lh-dev/localstorage/volumes/infra` location.
2. Name your certificate file as **`cert.crt`** and key file as **`cert.key`**.
3. Place the `.crt` and `.key` files under the `ibm-lh-dev/localstorage/volumes/infra/external-tls` directory.

4. Run the following command to restart the **`lhconsole-ui`** service:

   ```
   run ./start_service lhconsole-ui
   ```

   This command makes sure that all exposed ports, including the watsonx.data console serves the custom TLS certificate.

Customizing `jvm.config` and `config.properties`

Customize the Java virtual machine (JVM) parameters and configuration properties for IBM watsonx.data developer version.

watsonx.data Developer edition

About this task

Perform the following steps to customize `jvm.config` and `config.properties`.

Procedure

1. Run the following command to create a directory inside the host location where IBM watsonx.data developer version is installed.

   ```
   mkdir -p /root/ibm-lh-dev/localstorage/volumes/infra/ibm-lh-presto-config
   ```

2. Get the `jvm.config` and `config.properties` files from the `/opt/presto/etc` directory inside the `ibm-lh-presto` container. Run the following command to copy the files from the container to the host location.

   ```
   docker cp ibm-lh-presto:/opt/presto/etc/jvm.config /root/ibm-lh-dev/localstorage/volumes/
   infra/ibm-lh-presto-config/jvm.config
   docker cp ibm-lh-presto:/opt/presto/etc/config.properties /root/ibm-lh-dev/localstorage/
   volumes/infra/ibm-lh-presto-config/config.properties
   ```

3. You can add, remove, or update parameters present in the `config.properties` and `jvm.config` files.

   - The following example shows how to update the `jvm.config` file to increase the `max jvm heap` by changing from `-Xmx16G` to `-Xmx32G`.

     Example:

     ```
     vi jvm.config

     -server
     -Xmx32G
     -XX:+UseG1GC
     -XX:G1HeapRegionSize=32M
     -XX:+UseGCOverheadLimit
     -XX:+ExplicitGCInvokesConcurrent
     -XX:+HeapDumpOnOutOfMemoryError
     -XX:+ExitOnOutOfMemoryError
     -XX:-UseBiasedLocking
     -XX:ReservedCodeCacheSize=256M
     -Djdk.nio.maxCachedBufferSize=2000000
     -Djdk.attach.allowAttachSelf=true
     -Duser.timezone=Etc/GMT
     -javaagent:/opt/presto/lib/jmx_prometheus_javaagent-0.18.0.jar=9090:/opt/presto/etc/jmx-
     exporter-config.yaml
     -javaagent:/opt/presto/etc/setvendor.zip
     ```

   - The following example shows how to update the `config.properties` file to include `query.max-memory=32GB, query.max-stage-count=200, query.stage-count-warning-threshold=150`.

     Example :

     ```
     vi config.properties
     ```

```
coordinator=true
discovery-server.enabled=true
discovery.uri=https://ibm-lh-presto-svc:8443
heap_dump_on_exceeded_memory_limit.enabled=true
heap_dump_on_exceeded_memory_limit.file_directory=/var/presto/data/var/heapdump
heap_dump_on_exceeded_memory_limit.max.number=5
heap_dump_on_exceeded_memory_limit.max.size=300
http-server.authentication.type=PASSWORD, CERTIFICATE
http-server.http.enabled=false
http-server.https.enabled=true
http-server.https.excluded-
cipher=TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,TLS_DHE_RSA_
WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,TLS_DHE_RSA_WITH_AES_256_CBC_SH
A,TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_256_CBC
_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_128_C
BC_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_
WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_S
HA
http-server.https.keystore.key=bf89575c532a381059fcedf8
http-server.https.keystore.path=/opt/presto/presto-keystore.jks
http-server.https.port=8443
http-server.https.truststore.key=bf89575c532a381059fcedf8
http-server.https.truststore.path=/mnt/infra/tls/lh-ssl-ts.jks
internal-communication.https.keystore.key=bf89575c532a381059fcedf8
internal-communication.https.keystore.path=/opt/presto/presto-keystore.jks
internal-communication.https.required=true
internal-communication.https.trust-store-password=bf89575c532a381059fcedf8
internal-communication.https.trust-store-path=/mnt/infra/tls/lh-ssl-ts.jks
max-prefix-count=10
node-scheduler.include-coordinator=false
task.concurrency=16
query.max-memory=32GB
query.max-stage-count=200
query.stage-count-warning-threshold=150
```

4. Restart the `ibm-lh-presto` container by running the following command.

```
/root/ibm-lh-dev/bin/stop_service ibm-lh-presto
/root/ibm-lh-dev/bin/start_service ibm-lh-presto
```

5. Verify the `jvm.config` settings by running the following command:

```
root@aaa1:~# docker exec -it ibm-lh-Presto (Java) cat /opt/presto/etc/jvm.config
-server
-Xmx32G
-XX:+UseG1GC
-XX:G1HeapRegionSize=32M
-XX:+UseGCOverheadLimit
-XX:+ExplicitGCInvokesConcurrent
-XX:+HeapDumpOnOutOfMemoryError
-XX:+ExitOnOutOfMemoryError
-XX:-UseBiasedLocking
-XX:ReservedCodeCacheSize=256M
-Djdk.nio.maxCachedBufferSize=2000000
-Djdk.attach.allowAttachSelf=true
-Duser.timezone=Etc/GMT
-javaagent:/opt/presto/lib/jmx_prometheus_javaagent-0.18.0.jar=9090:/opt/presto/etc/jmx-
exporter-config.yaml
-javaagent:/opt/presto/etc/setvendor.zip
```

6. Verify the `config.properties` settings by running the following command:

```
root@aaa1:~# docker exec -it ibm-lh-Presto (Java) cat /opt/presto/etc/config.properties
coordinator=true
discovery-server.enabled=true
discovery.uri=https://ibm-lh-presto-svc:8443
heap_dump_on_exceeded_memory_limit.enabled=true
heap_dump_on_exceeded_memory_limit.file_directory=/var/presto/data/var/heapdump
heap_dump_on_exceeded_memory_limit.max.number=5
heap_dump_on_exceeded_memory_limit.max.size=300
http-server.authentication.type=PASSWORD, CERTIFICATE
http-server.http.enabled=false
http-server.https.enabled=true
http-server.https.excluded-
cipher=TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,TLS_DHE_RSA_WI
TH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TL
S_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_256_CBC_SHA25
6,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA25
6,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_WITH_AES_2
```

```
56_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
http-server.https.keystore.key=bf89575c532a381059fcedf8
http-server.https.keystore.path=/opt/presto/presto-keystore.jks
http-server.https.port=8443
http-server.https.truststore.key=bf89575c532a381059fcedf8
http-server.https.truststore.path=/mnt/infra/tls/lh-ssl-ts.jks
internal-communication.https.keystore.key=bf89575c532a381059fcedf8
internal-communication.https.keystore.path=/opt/presto/presto-keystore.jks
internal-communication.https.required=true
internal-communication.https.trust-store-password=bf89575c532a381059fcedf8
internal-communication.https.trust-store-path=/mnt/infra/tls/lh-ssl-ts.jks
max-prefix-count=10
node-scheduler.include-coordinator=false
task.concurrency=16
query.max-memory=32GB
query.max-stage-count=200
query.stage-count-warning-threshold=150
```

Working with developer utilities

The watsonx.data developer edition package includes utilities for SSL certificate management and engine management. These utilities are automatically configured to work with the installed Presto engine.

watsonx.data Developer edition

Developer package utilities

The developer package includes utilities for SSL certificate management and engine management.

watsonx.data Developer edition

These developer package utilities are automatically configured to work with the installed Presto engine.

For more information, see Installing the IBM watsonx.data developer version.

The developer version supports the following utilities:

- `presto-cli`
- `presto-run`
- `python-run`
- `dev-sandbox`
- `user-mgmt`

`presto-cli`

Start an interactive session with the Presto engine.

Syntax

Select a specific catalog context for the interactive Presto CLI session. For example:

```
bin/presto-cli --catalog=tpch
```

For more information, see "Using built-in presto-cli in the developer edition" on page 450

`presto-run`

Run SQL in a non-interactive manner.

Syntax

Use the bin/presto-run utility to run SQL in a noninteractive manner. For example:

```
bin/presto-run --catalog=tpch <<< "select * from tiny.customer limit 10;"
```

Run SQL from a file from your LH_SANDBOX_DIR mount. For more information, see “Sandbox directory” on page 68

```
bin/presto-run -f <path of sql file directory>
```

For more information, see “Running SQL queries by using developer utilities” on page 68.

`python-run`

It provides a way to run python scripts.

Syntax

Start an interactive python session.

```
bin/python-run
```

Note: The engine details are available as environment variables (ENG_HOST, ENG_PORT, ENG_USERNAME, ENG_PASSWORD). For example,`os.environ['ENG_PASSWORD']` provides the saved password for the selected engine.

Run a python script from your LH_SANDBOX_DIR mount. For more information, see “Sandbox directory” on page 68

```
bin/python-run <python-script-in-sandbox>
```

For more information, see “Running Python scripts by using developer utilities” on page 69.

`dev-sandbox`

Provides a containerized environment with useful utilities and python modules to help explore the watsonx.data.

Syntax

Set the sandbox that starts an interactive bash shell from which you can run other commands:

```
bin/dev-sandbox
```

For more information, see “Running Python scripts in sandbox container by using developer utilities” on page 69

`user-mgmt`

It is used to add or remove users to the developer edition and to change passwords for such users. This can be helpful to exercise user access authorizations in watsonx.data Developer edition.

As part of the installation procedure, the `ibmlhadmin`user is created. During setup, the `--password=<ibmlhadmin password>` argument can be used to select a password. If a password is not selected, then the default password is `password`.

Syntax

Use the `add-user` command to introduce a new user ID. For example:

```
user-mgmt add-user <username> <role> [password]
```

Note: 'Role' must either be 'Admin' or 'User'.

If a password for this user is not set on the command line (recommended), then you are prompted to enter the new password and confirm. For example: `user-mgmt add-user tom User password`

Use the `delete-user` command to delete an existing user ID. For example, `user-mgmt delete-user tom`

```
user-mgmt delete-user <username>
```

Use the `change-password` command to set a new password for a user ID or default user. For example:

```
user-mgmt change-password <username>
```

You are prompted to enter the new password and confirm. For example:

```
user-mgmt change-password ibmlhadmin
```

Sandbox directory

watsonx.data Developer edition

After you install the developer version, you can introduce a directory of your own for use with the containerized utilities. Such a 'sandbox' directory can include your SQL files, python scripts and other files that can be referenced when you use the utilities.

To mount your own directory, set the `LH_SANDBOX_DIR` environment variable to the sandbox directory path and create a sample SQL file or python script to be run in that directory. The following steps describe the setup of sandbox directory.

- Create the sandbox directory. For example:

  ```
  mkdir /home/jmouse/mysandbox
  ```

- Set the environment variable to the sandbox path created in the previous step. For example:

  ```
  export LH_SANDBOX_DIR=/home/jmouse/mysandbox
  ```

- You can then run the python script or SQL file in that directory.

  For more information, see the following links:

  – "Running SQL queries by using developer utilities" on page 68
  – "Running Python scripts by using developer utilities" on page 69
  – "Running Python scripts in sandbox container by using developer utilities" on page 69

Note: It is advised that you create a new directory that is solely meant for sharing files inside these containers. Avoid selecting directories such as your HOME as a sandbox directory. Mounting directories such as your HOME can cause file permission changes on existing files on some operating systems or with specific container runtime.

Running SQL queries by using developer utilities

You can use Presto engine to run the SQL queries by using developer utilities.

watsonx.data Developer edition

About this task

Complete the following steps to work with the Presto engine host:

Procedure

1. Establish the connection with Presto engine by using the `developer utilities`. For more information about installation, see "Installing the watsonx.data developer version" on page 56
2. Use any of the following methods to run the SQL query:

Using an interactive session

Use the argument `--catalog` to select a catalog as a default for the query. For example, to select the `tpch` catalog, run the following command:

```
/bin/presto-cli --catalog=tpch
```

Using a non-interactive session

Use the argument `--catalog` to select a catalog as a default for the query and pass in a query string on the command line as input. For example, to run a query against the `tpch` catalog, run the following command:

```
/bin/presto-cli --catalog=tpch <<< "select * from tiny.customer limit 10;"
```

Using an SQL file

Set the `LH_SANDBOX_DIR` mount. For more information, see "Sandbox directory" on page 68.

Use the following command to run an SQL file from your `LH_SANDBOX_DIR` mount:

```
bin/presto-run -f PATH_SANDBOX_DIR_SQL_FILE
```

Parameter value:

- `PATH_SANDBOX_DIR_SQL_FILE`: Specify the SQL query file location available in your sandbox directory.

  For example: `/home/jmouse/mysandbox/sample.sql`

Running Python scripts by using `developer utilities`

The `developer version` supports the execution of Python scripts.

watsonx.data Developer edition

About this task

Perform the following steps to work with the Presto engine host:

Procedure

You can run Python scripts by using the `python-run` command.

Start an interactive Python session

Use the following command to start a session to run Python scripts:

```
bin/python-run
```

Run a Python script

Set the LH_SANDBOX_DIR mount by referring "Sandbox directory" on page 68

Use the following command to run python file from your LH_SANDBOX_DIR mount:

```
bin/python-run <PATH_SANDBOX_DIR_PYTHON_FILE>
```

Parameter value:

- PATH_SANDBOX_DIR_PYTHON_FILE: Specify the Python query location available in your sandbox directory. For example: /home/jmouse/mysandbox/sample.py

Running Python scripts in sandbox container by using developer utilities

You can use the sandbox container to develop various functions by using python and connecting to a Presto (Java) engine.

watsonx.data Developer edition

About this task

To work with the sandbox container, complete the following steps:

Procedure

1. Set the variable "LH_SANDBOX_DIR" with the folder path. For example: `export LH_SANDBOX_DIR=/home/jmouse/mysandbox`. For more information, see on how to setup sandbox directory, see "Sandbox directory" on page 68

   ```
   export LH_SANDBOX_DIR=<folder_path>
   ```

2. Run the following command to start the sandbox environment.

   ```
    bin/dev-sandbox
   ```

   Note: The environment variables are set inside the sandbox container to help you with scripting and to avoid hardcoding.
3. Run the following commands to get the details of the engine host and engine password:
   - In the python environment, import os and check for the engine host and engine password from the following commands:

     ```
     import os
     os.environ['ENG_HOST']
     os.environ['ENG_PASSWORD']
     ```

4. Use sandbox to run python file from the sandbox directory by using the command `python3 <path of python file to be run>` as shown in the following example where 'sample.py' is the python file:

   ```
    python3 /home/jmouse/mysandbox/sample.py
   ```

   Note: Type 'exit' to exit from the sandbox container.

Switching between Presto (Java) and Presto (C++) engines in developer edition

Presto (Java) and Presto (C++) engines are mutually exclusive in the developer edition. You can have only one at a time.

watsonx.data Developer edition

Procedure

1. Run the following commands in the appropriate directory (parent directory of `ibm-lh-dev`).
   a.
   ```
   echo 'ENGINE_TYPE=prestissimo' >> ibm-lh-dev/etc/local_config.env
   ```
   b.
   ```
   ibm-lh-dev/bin/stop
   ```
   c.
   ```
   ibm-lh-dev/bin/start prestissimo
   ```
2. The **Infrastructure manager** in the developer edition is not integrated with Presto (C++). The system assumes that you have a Presto (Java) engine. Verify whether your engine is Presto (C++) by running the following SQL:

   ```
   SELECT * FROM system.runtime.nodes;
   ```

   If you have successfully enabled Presto (C++), you can see two rows—`ibm-lh-presto` and `ibm-lh-prestissimo`.

```
presto> select http_uri, coordinator, state from system.runtime.nodes;

              http_uri               | coordinator | state
-------------------------------------+-------------+--------
 https://ibm-lh-prestissimo-svc:7443 | false       | active
 https://ibm-lh-presto-svc:8443      | true        | active
(2 rows)
```

Switching Presto (C++) to Presto (Java)

Procedure

1. To change the developer edition back to Presto (Java) mode, remove the `ENGINE_TYPE=prestissimo` line that you added to `ibm-lh-dev/etc/local_config.env`.
2. Restart the developer edition in Presto (Java) mode by using the following commands:

   a.
   ```
   ibm-lh-dev/bin/stop
   ```
   b.
   ```
   ibm-lh-dev/bin/start
   ```
3. Run the following command to verify that the developer edition is in Presto (Java) mode.

   ```
   SELECT * FROM system.runtime.nodes
   ```

   The output shows a single row for `ibm-lh-presto`.

```
presto> select http_uri, coordinator, state from system.runtime.nodes;

              http_uri               | coordinator | state
-------------------------------------+-------------+--------
 https://ibm-lh-presto-svc:8443      | true        | active
(1 row)
```

Chapter 3. Setting up watsonx.data on Red Hat OpenShift

You can do the following to provision and manage watsonx.data developer edition.

watsonx.data on Red Hat OpenShift

Planning

Use the following information to plan your IBM watsonx.data installation.

watsonx.data on Red Hat OpenShift

Architecture for watsonx.data

watsonx.data on Red Hat OpenShift

IBM watsonx.data operators

IBM watsonx.data is installed by using operators.

watsonx.data on Red Hat OpenShift

Overview of operators

An *operator* is a custom Kubernetes controller. A controller implements a control loop that continuously watches the state of specific objects on your cluster and makes needed adjustments to ensure that the objects are running as expected.

Tip: The Red Hat OpenShift: Operators Framework video from Red Hat provides a high-level explanation of operators. (This video is also available on YouTube: https://www.youtube.com/watch?v=LymzLHRbQdk.)

IBM watsonx.data includes an operator that is responsible for managing the control plane and the instance.

Operator Lifecycle Manager

IBM watsonx.data uses Operator Lifecycle Manager (OLM) to install, update, and manage operators. OLM defines the several custom resource definitions (CRDs), including but not limited to:

- Catalog sources
- Subscriptions
- Install plans
- Cluster service versions
- Operator groups

The `cpd-cli manage apply-olm` command abstracts the complexity of creating and maintaining operators. Most users do not need to be familiar with OLM concepts to successfully install and manage watsonx.data because the `cpd-cli manage apply-olm` command creates and updates the operators for them.

Catalog source

Resource name
`CatalogSource` or `catsrc`

Description

A catalog source is a way to introduce new software or new versions of software to the cluster. A catalog source is a repository of operator versions (as specified by a cluster service version), custom resource definitions (CRDs), and packages that comprise an application. The information that is provided by the catalog source enables OLM to discover and install operators and their dependencies.

The watsonx.data perspective

The `cpd-cli manage apply-olm` command automatically creates the catalog source for watsonx.data.

Subscription

Resource name

`Subscription` or `sub`

Description

An operator subscription provides the following information to OLM:

- The name of the operator
- The location to install the operator
- The channel to subscribe to
- The install plan approval mechanism to use
- The catalog source to use

The watsonx.data perspective

The `cpd-cli manage apply-olm` command automatically creates the subscriptions for the operators.

Install plan

Resource name

`InstallPlan` or `ip`

Description

When an operator subscription is created, OLM creates an install plan that describes the set of resources. These resources include the cluster service version and the operator deployment that must be created to install or upgrade the operator.

When you create an operator, you specify approval strategy for the install plan:

Automatic
When a subscription includes `installPlanApproval: Automatic` and a newer Cluster Service Version (CSV) is detected, OLM automatically approves the install plan and installs or upgrades the operator.

The watsonx.data perspective

To simplify the installation experience, all watsonx.data operators are created with `installPlanApproval: Automatic`. This feature ensures that the `cpd-cli manage apply-olm` can create or update all required operators.

Cluster service version (CSV)

Resource name

`ClusterServiceVersion` or `csv`

Description

A cluster service version (CSV) represents a specific version of an operator. The information that is provided by the CSV enables OLM to:

- Understand the custom resources that the operator manages or uses.
- Introduce the custom resource definition (CRD), if it does not exist.
- Set up the operator's service accounts.
- Start the operator deployment.
- Keep the operator running safely on the cluster.
- Understand how updates must be applied when new versions of the operator are available on the cluster.

The watsonx.data perspective

The `cpd-cli manage apply-olm` automatically creates the appropriate CSVs for the watsonx.data.

Supported project (namespace) configurations

The projects (namespaces) that you must create on your cluster depend on several factors. Review the following information to determine which projects you must create.

watsonx.data on Red Hat OpenShift

Projects for shared cluster components

The shared cluster components are installed only one time on the cluster. All instances of IBM watsonx.data on the cluster share these components. Each component is installed in its own project. Do not install any other software in the project.

The following table provides the default project name and the environment variable that is used to identify the project in commands throughout the documentation. You are not required to use the default project names. The default names are used only if you don't specify a project name when you install the components.

Component	Default project name	Environment variable
IBM® Certificate Manager	`ibm-cert-manager`	`${PROJECT_CERT_MANAGER}`
License Service	`ibm-licensing`	`${PROJECT_CS_CONTROL}`
Scheduling Service	No default	`$ {PROJECT_SCHEDULING_SERVICE}`

For more information about these components, see Shared cluster components.

Projects required to install an instance of IBM watsonx.data

At a minimum, an instance of watsonx.data has two projects (namespaces): the operators project and the operands project.

Only the specified software should be installed in the operators project and the operands project. The projects do not have default names.

Operators project

The project where the operators are installed.

In installation and administration commands, the `${PROJECT_CPD_INST_OPERANDS}` environment variable refers to the operators project.

- **What are Operators?**

An operator is a custom Kubernetes controller. A controller implements a control loop that continually watches the state of specific objects on your cluster and makes adjustments as needed to ensure that the objects are running as expected.

Each component of watsonx.data includes an operator that is responsible for managing that component. For example, the control plane has a dedicated operator, and each service has a dedicated operator.

Install the operators for the services that you plan to install in the operands project. Each operator monitors the appropriate operands in the operands project.

Operands project

The project where the watsonx.data control plane and services are installed.

In installation and administration commands, the `${PROJECT_CPD_INST_OPERANDS}` environment variable refers to the operands project.

The operators monitor the operands to ensure that they are running as expected.

Multitenancy support

IBM watsonx.data supports different installation and deployment mechanisms for achieving multitenancy.

watsonx.data on Red Hat OpenShift

According to Gartner, multitenancy is:

“Multitenancy is a reference to the mode of operation of software where multiple independent instances of one or multiple applications operate in a shared environment. The instances (tenants) are logically isolated, but physically integrated. The degree of logical isolation must be complete, but the degree of physical integration will vary."

https://www.gartner.com/it-glossary/multitenancy

Achieving multitenancy with multiple instances of watsonx.data

You can install the watsonx.data multiple times on the same cluster by installing each instance of the control plane and data plane in a separate project (Kubernetes namespace).

Note: The watsonx.data operator is installed only one time on the cluster and shared by any instances of watsonx.data on the cluster.

For more information, see Installing the stand-alone watsonx.data.

This installation architecture offers complete logical isolation of each instance of watsonx.data with limited physical integration between the instances. When you set up your cluster, a Red Hat OpenShift Container Platform cluster administrator can create multiple projects to partition your cluster. Within each project, you can assign resource quotas. Each project acts as a virtual cluster with its own security and network policies. In addition to being logically separated, you can use different authentication mechanisms for each watsonx.data deployment.

This tenancy model addresses the following use cases:

- Partitioning your nonproduction environment from your production environment in a continuous integration, continuous delivery (CICD) pipeline. In this model, tenants work in discrete, isolated units with a clear separation of duties.
- Creating instances for different departments or business units that have distinct roles and responsibilities within your enterprise. In this model, each tenant has their own authentication mechanism, resource quotas, and assets.

This tenancy model also offers several advantages:

- You can minimize your costs by deploying multiple instances on the same cluster.
- The cluster administrator can establish tenant-specific quality of service characteristics in each instance.

- The cluster administrator can assign project administrators to manage an instance of watsonx.data.

The project administrator can control which services are deployed in the project and can manage the resources that are associated with the project. However, the project administrator does not have access to cluster-level settings and cannot change the resource quotas for their project.

For more information about projects, see "Supported project (namespace) configurations" on page 74.

Installation roles and personas

Only a Red Hat OpenShift cluster administrator can complete certain planning and installation tasks. A project administrator can complete other tasks. Learn which role is set to complete each task, based on the installation method that you prefer.

watsonx.data on Red Hat OpenShift

Administrative roles

IBM watsonx.data supports a separation of roles and duties that enables the installation workflow to proceed with as few restrictions as possible.

A Red Hat OpenShiftContainer Platform can complete all of the installation tasks. However, if you want to enable users with fewer permissions to complete some of the installation tasks, use the roles defined in the following sections.

The installation and upgrade tasks use the following tags to help you identify which users are involved in a task:

- Cluster administrator
- Registry administrator

Cluster administrator

A cluster administrator is responsible for setting up and preparing the cluster for watsonx.data. To complete these tasks, the user must have the `cluster-admin` role.

A cluster administrator must complete the following tasks:

- Setting up a cluster, which includes:
  - Installing Red Hat OpenShiftContainer Platform, if it is not already installed.
  - Installing persistent storage, if it is not already installed.
  - Installing Multicloud Object Gateway, if needed.
  - Setting up a private container registry, if needed.
  - Ensuring the cluster is security hardened.
  - Adding, expanding, or replacing nodes, as needed
- Preparing the cluster for the watsonx.data, which includes:
  - Updating the global image pull secret.
  - (Optional) Manually creating the projects (namespaces) where the shared cluster components will be installed.
  - Installing the shared cluster components.
  - Configuring persistent storage for watsonx.data.
  - Creating custom security context constraints (SCCs) for services, if needed.
  - Adjusting node settings for services, if needed.
- Preparing the cluster for an instance of watsonx.data, which includes:
  - (Optional) Manually creating the projects (namespaces) for the instance of watsonx.data.

- (Optional) Setting namespace quotas and limit ranges on the projects that are associated the instance.
- Applying the required permissions to the instance to ensure that the operators project for the instance can watch the operands project where the watsonx.data control plane and services will be installed.

  If the instance will include tethered projects, ensure that you apply the required permissions to the tethered projects.
- Assigning the required roles to the user or users who will administer the instance.

Each instance of watsonx.data is logically isolated from any other instances of Cloud Pak for Data on the cluster. For more information about the private topology, see Supported project (namespace) configurations.

Registry administrator

If you use a private container registry, you must have a user who can push images to the private container registry, such as a registry administrator.

The registry administrator is responsible for mirroring the watsonx.data software images from the IBM Entitled Registry to the private container registry.

The registry administrator does not need access to the Red Hat OpenShift Container Platform cluster.

Licenses and entitlements

Your IBM watsonx.data license determines the components and programs that you are entitled to use.

watsonx.data on Red Hat OpenShift

License information

Detailed information about the watsonx.data licenses is available on the IBM Terms site.

What's included with your watsonx.data license?

Your purchase of watsonx.data includes entitlements to the watsonx.data software and the following prerequisites that you need to be able to install watsonx.data.

- IBM Cloud Pak® for Data platform software
- IBM Cloud Pak foundational services
- Analytics Engine Powered by Apache Spark
- EDB PostgreSQL with IBM
- Red Hat OpenShift Container Platform
- Red Hat Enterprise Linux
- IBM Storage Ceph®
- IBM Storage Fusion Essentials

The watsonx.data is also optionally available in a Non-Production version.

Obtaining your IBM entitlement API key

You must have your IBM entitlement API key to access images in the IBM Entitled Registry.

After you purchase watsonx.data, an entitlement API key for the software is associated with your My IBM account. You need this key to complete the watsonx.data installation.

To obtain the IBM entitlement API key that is associated with your My IBM account:

1. Log in to Container software library on My IBM with the IBMid and password that are associated with the entitled software.
2. On the **Entitlement keys** tab, select **Copy** to copy the entitlement key to the clipboard.
3. Save the API key in a text file.

Obtaining Red Hat OpenShift Container Platform

Red Hat OpenShift Container Platform is a prerequisite for watsonx.data. You can use your watsonx.data entitlement to install self-managed OpenShift on the infrastructure of your choice.

You can download Red Hat OpenShift from the Red Hat Customer Portal.

When you purchase an IBM product that includes entitlement to Red Hat software, the Red Hat is accessible through your Red Hat account.

You can refer to watsonx.data license for details.

Obtaining IBM Cloud Pak for Data

IBM Cloud Pak for Data is a prerequisite for watsonx.data. Installing watsonx.data includes information on how to access or mirror the IBM Cloud Pak for Data images.

By default, the following services are automatically installed when you install watsonx.data:

- IBM Cloud Pak For Data
- IBM Cloud Pak foundational services
- EDB PostgreSQL with IBM 14.7

The License Service is automatically installed when you run the `cpd-cli manage apply-entitlement` command.

Obtaining watsonx.data

Images for the watsonx.data are available in the IBM Entitled Registry. Installing includes information on how to access or mirror the images for the software that you want to install.

Obtaining storage entitlement

You are entitled to use the following storage for watsonx.data

- IBM Storage Ceph
- IBM Storage Fusion Essentials

Obtaining IBM Storage Ceph

IBM Storage Ceph 5.3 Activation Key forIBM Storage Insights information is available on IBM Passport Advantage® (M0BCGML).

Obtaining IBM Storage Fusion Essentials

IBM watsonx.data users are entitled to obtain IBM Storage Fusion Essentials upon acceptance of the watsonx.data license.

IBM Storage Fusion Essentials provides the storage that you need to get started without purchasing a storage license. You can also upgrade your license to enable capabilities such as online backup and disaster recovery with no service disruption.

Entitlement terms

You are entitled to up to 100 TB of IBM Storage Fusion Data Foundation in internal deployment mode. If you exceed this amount, a separate license is required. There is no time usage limit, and this entitlement is supported by IBM.

Reporting your use

You are required to keep a record of the size of deployments to report to IBM as requested. If you are using Container Licensing, you can use the License Service to measure watsonx.data usage. For more information, see Monitoring and reporting use against license terms.

Planning for an environment with other IBM Cloud Paks

If you plan to install IBM watsonx.data on a cluster that already has other IBM Cloud Paks installed, it is recommended to install the IBM Cloud Pak for Data 4.7.1 or later release of watsonx.data.

watsonx.data on Red Hat OpenShift

Storage considerations

To install IBM watsonx.data, you must have a supported persistent storage solution that is accessible to your Red Hat OpenShift cluster.

watsonx.data on Red Hat OpenShift

Review the following sections to determine what storage is right for you:

What storage options are supported for the platform?

IBM watsonx.data supports and is optimized for several types of persistent storage.

watsonx.data uses dynamic storage provisioning. A Red Hat OpenShift cluster administrator must properly configure storage before watsonx.datais installed.

Note: watsonx.data supports the following object storage types: Amazon S3, Ceph, Hadoop Distributed File System (HDFS), IBM Cloud Object Storage (COS), and IBM Storage Scale.

Important: It is your responsibility to review the documentation for the storage that you plan to use. Ensure that you understand any limitations that are associated with the storage.

For more information about the storage that is supported, see Storage requirements.

Storage option	Version	Notes
Red Hat OpenShift Data Foundation	• Version 4.10 or later fixes • Version 4.12 or later fixes	Available in Red Hat OpenShift Platform Plus. Ensure that you install a version of Red Hat OpenShift Data Foundation that is compatible with the version of Red Hat OpenShift Platform that you are running. For details, see https://access.redhat.com/articles/4731161.
IBM® Storage Fusion Data Foundation	• Version 2.5.2 or later fixes	Available in IBM Storage Fusion.
IBM Storage Fusion Global Data Platform	• Version 2.3.0 or later fixes • Version 2.4.0 or later fixes • Version 2.5.2 or later fixes	Available in IBM Storage Fusion.
IBM Storage Scale Container Native (with IBM Storage Scale Container Storage Interface)	Version 5.1.5 or later fixesCSI Version 2.6.x or later fixes	Available in either • IBM Storage Fusion • IBM Storage Suite for watsonx.data

Storage option	Version	Notes
Portworx	• Version 2.9.1.3 or later fixes • Version 2.12.2 or later fixes	If you are running Red Hat OpenShift Container Platform Version 4.12, upgrade to Portworx Version 2.12.2 or later.
NFS	Version 3 or 4	Version 3 is preferred. If you use Version 4, ensure that your storage class uses NFS Version 3 as the mount option. For details, see Setting up dynamic provisioning .
Amazon Elastic Block Store (EBS)	Not applicable	In addition to EBS storage, your environment must also include EFS storage.
IBM Cloud Block Storage	Not applicable	
NetApp Trident	Version 22.4.0 or later fixes	
IBM Storage Ceph	Version 5.x or 6.x	

Note: IBM evaluates the preceding storage options. However, you must run the watsonx.data storage validation tool on your Red Hat OpenShift cluster to: Evaluate whether the storage on your cluster is sufficient for use with watsonx.data. Assess storage provided by other vendors. This tool does not support other types of storage. You can use other storage environments at your own risk.

What storage options are supported in the deployment environment?

If watsonx.data supports a storage option, you can install watsonx.data with that storage as long as it is supported on your deployment option. Ensure that you select a storage option that:

- Works on your chosen deployment environment.

  Some storage options are supported only on a specific deployment environment.

  For clusters hosted on third-party infrastructure, such as IBM Cloud or Amazon Web Services, use storage that is native to the infrastructure, if possible.
- Supports the services that you plan to install.

  Some services support a subset of the storage options that the platform supports. For details, see Storage requirements.

  Has sufficient I/O performance.

  For information on how to test I/O performance, see Disk requirements.

Deployment environment	**Managed Red Hat OpenShift**	**Self-managed Red Hat OpenShift**
On-premises	Not supported	The following storage options are supported on bare metal and VMware infrastructure: • Red Hat OpenShift Data Foundation • IBM Storage Fusion • IBM Storage Scale Container Native • Portworx • NFS • NetApp Trident
IBM Cloud	Red Hat OpenShift on IBM Cloud supports the following storage options: • Red Hat OpenShift Data Foundation • IBM Storage Fusion • Portworx • IBM Cloud Block Storage For up-to-date information about the storage supported on this environment, review **Storing data on persistent storage** in the Red Hat OpenShift on IBM Cloud documentation.	The following storage options are supported on *VPC* IBM Cloud infrastructure: • Red Hat OpenShift Data Foundation • IBM Storage Fusion • Portworx • NFS
Amazon Web Services (AWS)	Red Hat OpenShift Service on AWS (ROSA) supports the following storage options: • Amazon Elastic Block Store (EBS) • NetApp Trident (includes Amazon FSx for NetApp ONTAP)	The following storage options are supported on AWS infrastructure: • Red Hat OpenShift Data Foundation • IBM Storage Fusion • Amazon Elastic Block Store (EBS) • Portworx • NFS • NetApp Trident (includes Amazon FSx for NetApp ONTAP) • AWS GovCloud (US)

Deployment environment	Managed Red Hat OpenShift	Self-managed Red Hat OpenShift
Microsoft Azure	Azure Red Hat OpenShift (ARO) supports the following storage options: • Red Hat OpenShift Data Foundation • NFS	The following storage options are supported on Microsoft Azure infrastructure: • Red Hat OpenShift Data Foundation • IBM Storage Fusion • Portworx • NFS, specifically Microsoft Azure locally redundant Premium SSD storage
Google Cloud	Managed Red Hat OpenShift on Google Cloud is not supported.	The following storage options are supported on Google Cloud infrastructure: • Red Hat OpenShift Data Foundation • Portworx • NFS

What storage options are supported on the version of Red Hat OpenShift Container Platform?

Storage option	Version 4.10	Version 4.12
Red Hat OpenShift Data Foundation	√	√
IBM Storage Fusion	√	√
IBM Storage Scale Container Native	√	√
Portworx	√	√
NFS	√	√
Amazon Elastic Block Store (EBS)	√	√
IBM Cloud Block Storage	√	√

What storage options are supported on my hardware?

Storage option	x86-64
Red Hat OpenShift Data Foundation	√
IBM Storage Fusion	√
IBM Storage Scale Container Native	√
Portworx	√
NFS	√
Amazon Elastic Block Store (EBS)	√

Storage option	x86-64
IBM Cloud Block Storage	√

Storage Comparison

Use the following information to decide which storage solution is right for you:

- License requirements

  The following table lists whether you need a separate license to use each storage option.

Storage option	Details
Red Hat OpenShift Data Foundation	watsonx.data customers can obtain Red Hat OpenShift Data Foundation Essentials storage entitlement at no charge. **Entitlement terms** IBM Storage Fusion entitlement applies only to self-managed Red Hat OpenShift. You are entitled to use IBM Storage Fusion with the following limitations: – You can use up to 6 TB of IBM Storage Fusion storage. – You can use IBM Storage Fusion for up to 12 months. If you exceed these terms, a separate license is required. Contact your IBM Sales representative for access to this storage.
IBM Storage Fusion	watsonx.data customers can obtain IBM Storage Fusion storage entitlement at no charge. **Entitlement terms** IBM Storage Fusion entitlement applies only to self-managed Red Hat OpenShift. You are entitled to use IBM Storage Fusion with the following limitations: – You can use up to 6 TB of IBM Storage Fusion storage. – You can use IBM Storage Fusion for up to 12 months. If you exceed these terms, a separate license is required. Contact your IBM Sales representative for access to this storage.
IBM Storage Scale Container Native (with IBM Storage Scale Container Storage Interface)	You can use IBM Storage Scale Container Native as part of IBM Storage Fusion.
Portworx	Requires a separate license.
NFS	Does not require a license.

Storage option	Details
Amazon Elastic Block Store (EBS)	Requires a separate subscription.
IBM Cloud Block Storage	Requires a separate subscription.

- Storage classes

  The person who installs watsonx.data on the cluster must know which storage classes to use during installation. The following table lists the types of storage. When applicable, the table also lists the storage classes to use and points to more guidance on how to create the storage classes.

Storage option	Details
Red Hat OpenShift Data Foundation	The storage classes that are required are automatically created when you install Red Hat OpenShift Data Foundation. watsonx.data uses the following storage classes: – RWO block storage: `ocs-storagecluster-ceph-rbd`
IBM Storage Fusion	The recommended storage class is called `ibm-spectrum-scale-sc`. IBM Storage Fusion supports both ReadWriteMany (RWX access) and ReadWriteOnce (RWO access) with the same storage class.
IBM Storage Scale Container Native (with IBM Storage Scale Container Storage Interface)	The recommended RWX storage class is called `ibm-spectrum-scale-sc`. IBM Storage Scale Container Native supports both ReadWriteMany (RWX access) and ReadWriteOnce (RWO access) with the same storage class.
Portworx	The recommended storage classes are listed in Creating storage classes.
NFS	The recommended RWX storage class is called `managed-nfs-storage`. For details on setting up dynamic provisioning and creating the recommended storage class, see Setting up NFS storage.
Amazon Elastic Block Store (EBS)	Use either of the following RWO storage classes: – `gp2-csi` – `gp3-csi`
IBM Cloud Block Storage	Use the following RWO storage class: `ibmc-block-gold`

- Data replication for high availability
-

Storage option	Details
Red Hat OpenShift Data Foundation	Supported. By default, all services use multiple replicas for high availability. Red Hat OpenShift Data Foundation maintains each replica in a distinct availability zone.

Storage option	Details
IBM Storage Fusion	Supported. Replication is supported and can be enabled within the IBM Storage Scale Storage Cluster in various ways, see Data Mirroring and Replication in the IBM Storage Scale documentation.
IBM Storage Scale Container Native (with IBM Storage Scale Container Storage Interface)	Supported. Replication is supported and can be enabled within the IBM Storage Scale Storage Cluster in various ways, see Data Mirroring and Replication in the IBM Storage Scale documentation.
Portworx	Supported For details about the replicas for each storage class, see Creating storage classes.
NFS	Replication support depends on your NFS server.
Amazon Elastic Block Store (EBS)	Supported. When you create an EBS volume, it is automatically replicated within its Availability Zone to prevent data loss due to failure of any single hardware component.
IBM Cloud Block Storage	Supported. You can create a snapshot schedule to automatically copy snapshots to a destination volume in a remote data center for Data replication. For details, see Replicating data in the IBM Cloud documentation.

- Backup and restore
-

Storage option	Details
Red Hat OpenShift Data Foundation	Container Storage Interface support for snapshots and clones. Tight integration with Velero CSI plug-in for Red Hat OpenShift Container Platform backup and recovery.
IBM Storage Fusion	For storage level backup, see Protecting Data in the IBM Storage Fusion documentation.
IBM Storage Scale Container Native (with IBM Storage Scale Container Storage Interface)	For details, see Data protection and disaster recovery in the IBM Storage Scale documentation.
Portworx	**On-premises** Limited support. **IBM Cloud** Supported by the Portworx Enterprise Disaster Recovery plan.
NFS	Limited support.
Amazon Elastic Block Store (EBS)	
IBM Cloud Block Storage	

- Encryption of data at rest

•

Storage option	Details
Red Hat OpenShift Data Foundation	Supported. Red Hat OpenShift Data Foundation uses Linux Unified Key System (LUKS) version 2 based encryption with a key size of 512 bits and the aes-xts-plain64 cipher. Enable encryption for your whole cluster during cluster deployment to ensure encryption of data at rest. Encryption is disabled by default. Working with encrypted data incurs a small performance penalty. For details, see *Security considerations* in the Red Hat OpenShift Data Foundation documentation: – Version 4.10 – Version 4.12 **Restriction:** If you are using Red Hat OpenShift Data Foundation Essentials, you can only use keys that are managed on the cluster, you cannot use an external key management system (KMS). **Support for FIPS cryptography** By storing all data in volumes that use RHEL-provided disk encryption and enabling FIPS mode for your cluster, both data at rest and data in motion, or network data, are protected by FIPS Validated Modules in Process encryption. You can configure your cluster to encrypt the root file system of each node. For details, see *FIPS 140-2* in the Red Hat OpenShift Data Foundation documentation: – Version 4.10 – Version 4.12 If you have Red Hat OpenShift Data Foundation Advanced, you can also encrypt persistent volume claims (PVCs) in addition to enabling encryption for the whole cluster. You can enable PVC encryption for storage that is created by the `ocs-storagecluster-ceph-rbd` storage class.
IBM Storage Fusion	Supported. For details, see Encryption in the IBM Storage Scale documentation.
IBM Storage Scale Container Native(with IBM Storage Scale Container Storage Interface)	Supported. For details, see Encryption in the IBM Storage Scale documentation.

Storage option	Details
Portworx	Supported by Portworx Enterprise only. Portworx uses the LUKS format of dm-crypt and AES-256 as the cipher with xts-plain64 as the cipher mode. **On-premises deployments** Refer to Enabling Portworx volume encryption in the Portworx documentation. **IBM Cloud deployments** To protect the data in your Portworx volumes, encrypt the volumes with IBM Key Protect or Hyper Protect Crypto Services.
NFS	Check with your storage vendor on the steps to enable encryption of data at rest.
Amazon Elastic Block Store (EBS)	
IBM Cloud Block Storage	

- Network and I/O requirements
-

Storage option	Details
Red Hat OpenShift Data Foundation	**Network requirements** Your network must support a minimum of 10 Gbps. **I/O requirements** Each node must have at least one enterprise-grade SSD or NVMe device that meets the Disk requirements in the system requirements. For more information, see Planning your deployment in the Red Hat OpenShift Data Foundation documentation. If SSD or NVMe aren't supported in your deployment environment, use an equivalent or better device.
IBM Storage Fusion	**Network requirements** You must have sufficient network performance to meet the storage I/O requirements. **I/O requirements** For details, see Disk requirements in the system requirements.
IBM Storage Scale Container Native (with IBM Storage Scale Container Storage Interface)	**Network requirements** You must have sufficient network performance to meet the storage I/O requirements. **I/O requirements** For details, see Disk requirements in the system requirements.

Storage option	Details
Portworx	**Network requirements** Your network must support a minimum of 10 Gbps. For details, see Prerequisites in the Portworx documentation. **I/O requirements** For details, see Disk requirements in the system requirements.
NFS	**Network requirements** You must have sufficient network performance to meet the storage I/O requirements. **I/O requirements** For details, see Disk requirements in the system requirements.
Amazon Elastic Block Store (EBS)	**Network requirements** You must have sufficient network performance to meet the storage I/O requirements. **I/O requirements** For details, see Disk requirements in the system requirements.
IBM Cloud Block Storage	**Network requirements** You must have sufficient network performance to meet the storage I/O requirements. **I/O requirements** For details, see Disk requirements in the system requirements.

- This section describes the resource requirements for the various storage options.

  For information about the minimum amount of storage that is required for your environment, see Storage requirements.

-

Storage option	vCPU	Memory	Storage
Red Hat OpenShift Data Foundation	– 10 vCPU per node on three initial nodes. – 2 vCPU per node on any additional nodes For details, see Resource requirements.	– 24 GB of RAM on initial three nodes. – 5 GB of RAM on any additional nodes. For details, see Resource requirements.	A minimum of three nodes. On each node, you must have at least one SSD or NVMe device. Each device should have at least 1TB of available storage. For details, see Storage device requirements.

Storage option	vCPU	Memory	Storage
IBM Storage Fusion	8 vCPU on each worker node to deploy IBM Storage Scale Container Native and IBM Storage Scale Container Storage Interface Driver.See the IBM Storage Scale Container Native hardware requirements.	16 GB of RAM on each worker node. For details, see the IBM Storage Scale Container Native requirements.	1 TB or more of available space
IBM Storage Scale Container Native (with IBM Storage Scale Container Storage Interface)	8 vCPU on each worker node to deploy IBM Storage Scale Container Native and IBM Storage Scale Container Storage Interface Driver.See the IBM Storage Scale Container Native requirements.	16 GB of RAM on each worker node. For details, see the IBM Storage Scale Container Native requirements.	1 TB or more of available space
Portworx	**On-premises** 4 vCPU on each storage node **IBM Cloud** For details see the following sections of Storing data on software-defined-storage (SDS) with Portworx: – What worker node flavor in Red Hat OpenShift on IBM Cloud is the right one for Portworx? – What if I want to run Portworx in a classic cluster with non-SDS worker nodes?	4 GB of RAM on each storage node	A minimum of three storages nodes. On each storage node, you must have: – A minimum of 1 TB of raw, unformatted disk – An extra 100 GB of raw, unformatted disk for a key-value database.
NFS	8 vCPU on the NFS server	32 GB of RAM on the NFS server	1 TB or more of available space
Amazon Elastic Block Store (EBS)			
IBM Cloud Block Storage			

More documentation

Storage option	Documentation links
Red Hat OpenShift Data Foundation	**watsonx.data configuration guidance** For postinstallation guidance, see Configuring persistent storage . **Troubleshooting** Product documentation for Troubleshooting OpenShift Data Foundation 4.5
IBM Storage Fusion	**watsonx.data configuration guidance** For postinstallation guidance, see Configuring persistent storage . **Troubleshooting** • For IBM Storage Scale, see Troubleshooting and support in the IBM Storage Scale Container Native documentation. • For IBM Storage Scale Container Storage Interface, seeTroubleshooting and support in the IBM Storage Scale Container Storage Interface documentation.
IBM Storage Scale Container Native (with IBM Storage Scale Container Storage Interface)	**watsonx.data configuration guidance** For postinstallation guidance, see Configuring persistent storage . **Troubleshooting** • For IBM Storage Scale, see Troubleshooting and support in the IBM Storage Scale Container Native documentation. • For IBM Storage Scale Container Storage Interface, seeTroubleshooting and support in the IBM Storage Scale Container Storage Interface documentation.
Portworx	**watsonx.dataconfiguration guidance** For more information about postinstallation, see Configuring persistent storage . **Troubleshooting** Troubleshoot Portworx on Kubernetes.
NFS	**watsonx.data configuration guidance** For more information about postinstallation, see Configuring persistent storage . **Troubleshooting** Refer to the documentation from your NFS provider.
Amazon Elastic Block Store (EBS)	**watsonx.data guidance** For more information about postinstallation, see Configuring persistent storage . **Troubleshooting** See the AWS documentation.

Storage option	Documentation links
IBM Cloud Block Storage	**watsonx.data configuration guidance** For more information about postinstallation, see Configuring persistent storage . **Troubleshooting** Debugging Block Storage failures in the IBM Cloud documentation.

System requirements

watsonx.data on Red Hat OpenShift

Hardware requirements

Before you install IBM watsonx.data, review the hardware requirements for the control plane, the shared cluster components, and the services that you plan to install.

watsonx.data on Red Hat OpenShift

Hardware requirements of watsonx.data platform

Install watsonx.data on a Red Hat OpenShift Container Platform cluster. For information about the supported versions of Red Hat OpenShift Container Platform, see "Software requirements" on page 96.

It is recommended that you deploy watsonx.data on a highly available cluster.

The following requirements are the *minimum* recommendations for a small, stable deployment of watsonx.data. Use the minimum recommended configuration as a *starting point* for your cluster configuration. If you use fewer resources, you are likely to encounter stability problems.

The following configuration has been tested and validated by IBM®. However, Red Hat OpenShift Container Platform supports other configurations. If the configuration in the following table does not work in your environment, you can adapt the configuration based on the guidance in the Red Hat OpenShift documentation. (Links to the relevant Red Hat OpenShift documentation are available in the "Software requirements" on page 96.) In general, watsonx.data is primarily concerned with the resources that are available on your worker nodes.

Important:

Work with your IBM Sales representative to size your cluster. The size of your cluster depends on:

- The shared components that you need to install.
- The services that you plan to install.

  The sizing requirements for services are available in Service hardware requirements. If you install only a few services with small vCPU and memory requirements, you might not need additional resources. However, if you plan to install multiple services or services with large footprints, add the appropriate amount of vCPU and memory to the minimum recommendations below.
- The types of workloads that you plan to run.

  For example, if you plan to run complex analytics workloads in addition to other resource-intensive workloads, such as ETL jobs, you can expect reduced concurrency levels if you don't add additional computing power to your cluster.

  Because workloads vary based on several factors, use measurements from running real workloads with realistic data to size your cluster.

Node role	Hardware	Number of servers	Minimum available vCPU	Minimum memory	Minimum storage
Master + infra	• x86-64	3 master (for high availability) and 3 infrastructure on the same 3 nodes	8 vCPU per node	32 GB RAM per node	No additional storage is needed. For sizing guidance, refer to the Red Hat OpenShift Container Platform documentation.
Worker/ compute	• x86-64	3+ worker/ compute nodes	16 vCPU per node	• 64 GB RAM per node (minimum) • 128 GB RAM per node (recommende d)	300 GB of storage space per node for storing container images locally.
Load balancer	• x86-64	2 load balancer nodes	• x86-64: 4 vCPU per node	4 GB RAM per node Add another 4 GB of RAM for access restrictions and security control.	Add 100 GB of root storage for access restrictions and security control.

Load balancer

A load balancer is required when using three master nodes. The load balancer distributes the traffic load of the master and proxy nodes, securely isolates the master and compute node IP addresses, and facilitates external communication, including accessing the management console and API or making other requests to the master and proxy nodes.

Cluster node settings

The time on all of the nodes must be synchronized within 500 ms.

Disk requirements

To prepare your storage disks, ensure that you have good I/O performance, and prepare the disks for encryption.

I/O performance

When I/O performance is not sufficient, services can experience poor performance or cluster instability, such as functional failures with timeouts. This is especially true when you are running a heavy workload.

The I/O performance requirements for watsonx.data are based on extensive testing in various cloud environments. The tests validate the I/O performance in these environments. The requirements are based on the performance of writing data to representative storage classes by using the following block size and thread count combinations:

- To evaluate disk latency, the I/O tests use a small block (4 KB) with 8 threads.
- To evaluate disk throughput, the I/O tests use a large block (1 GB) with 2 threads..

To evaluate the storage performance on the cluster where you plan to install watsonx.data, run the watsonx.data storage performance validation playbook. Ensure that the results are comparable to the following recommended minimum values:

Disk latency (4 KB block with 8 threads)
For disk latency tests, 18 MB/s has been found to provide sufficient performance.

Disk throughput (1 GB block with 2 thread)
For disk throughput tests, 226 MB/s has been found to provide sufficient performance.

To ensure sufficient performance, both requirements should be satisfied.

Some storage types might have more stringent I/O requirements. For details, see "Storage considerations " on page 79.

Important: It is recommended that you run the validation playbook several times to account for variations in workloads, access patterns, and network traffic.

In addition, if your storage volumes are remote, network speed can be a key factor in your I/O performance. For good I/O performance, ensure that you have sufficient network speed, as described in "Storage considerations " on page 79.

Encryption with Linux® Unified Key Setup
To ensure that your data within watsonx.data is stored securely, you can encrypt your disks. If you use Linux Unified Key Setup-on-disk-format (LUKS), you must enable LUKS when you install Red Hat OpenShift Container Platform. For more information, see Encrypting disks during installation in the Red Hat OpenShift Container Platform documentation.

Shared cluster component requirements

Shared cluster components provide underlying functionality for the watsonx.data control plane and services.

Use the following information to determine whether you have the minimum required resources to install each component on your watsonx.data cluster.

Service	vCPU	Memory	Storage	Notes
IBM Cloud Pak foundational services	See the IBM Cloud Pak foundational services documentation.	See the IBM Cloud Pak foundational services documentation.	See the IBM Cloud Pak foundational services documentation.	Required. This software is installed once on the cluster.
Common core services	**Operator pods:** 0.1 vCPU **Catalog pods:** 0.01 vCPU **Operand:** 11 vCPU	**Operator pods:** 0.256 GB RAM **Catalog pods:** 0.05 GB RAM **Operand:** 18.3 GB RAM	**Persistent storage:** 500 GB **Ephemeral storage:** 100 GB **Image storage:** Approximately 25710 MB	Required in some situations. Depending on the services that you install, this software is installed once in each Red Hat OpenShift project where watsonx.data is installed. For details, see "Software requirements" on page 96.

Service hardware requirements

Use the following information to determine whether you have the minimum required resources to install each service that you want to use.

Service	vCPU	Memory	Storage	Notes
watsonx.data	**Operator pods:** 0.1 vCPU **Catalog pods:** 0.01 vCPU **Operand:** 9.8 vCPU	**Operator pods:** 0.256 GB RAM **Catalog pods:** 0.05 GB RAM **Operand:** 28.352 GB RAM	**Persistent storage:** 35 GB **Image storage:** Approximately 15620 MB	

Storage requirements

Before you install IBM watsonx.data, review the storage requirements.

watsonx.data on Red Hat OpenShift

watsonx.data platform storage requirements

A IBM watsonx.data deployment requires several types of storage:

Storage for images in the private container registry

Depending on your environment, you might need to store images in a private container registry rather than pulling them directly from the IBM Entitled Registry.

If you use a private container registry, you must have sufficient space for the watsonx.data components that you plan to install.

Sizing

A minimum of 300 GB of storage space in the private container registry.

Tip:

1. Review the image size information in hardware-reqs.dita to ensure that you have sufficient space for the images you plan to mirror.
2. Use the `watsonx_data-manage` delete-images command to remove unused images from the private container registry.

Local storage for container images

Each node on your cluster must have local storage for the container images that are running on that node.

Storage location

The container images are stored in the root file system on the nodes.

On Red Hat OpenShift Container Platform, local copies of the images are stored in `/var/lib/containers`. For more details, see **Data storage management** in the Red Hat OpenShift Container Platform documentation:

- Version 4.10
- Version 4.12

Sizing

A minimum of 300 GB of storage space per node.

Persistent storage

watsonx.data supports and is optimized for several types of persistent storage:

Storage option	Version	Notes
Red Hat OpenShiftData Foundation	• Version 4.10 or later fixes • Version 4.12 or later fixes	Available in Red Hat OpenShift Platform Plus. Ensure that you install a version of Red Hat OpenShift Data Foundation. It must be compatible with the version of Red Hat OpenShift Container Platform that you are running. For details, see https://access.redhat.com/articles/4731161.
IBM Storage Fusion Data Foundation	• Version 2.5.2 or later fixes	Available in IBM Storage Fusion. Ensure that you install a version of IBM Storage Fusion Data Foundation that is compatible with the version of Red Hat OpenShift Container Platform that you are running. For more details, see https://access.redhat.com/articles/4731161.
IBM Storage Fusion Global Data Platform	• Version 2.3.0 or later fixes • Version 2.4.0 or later fixes • Version 2.5.2 or later fixes	Available in IBM Storage Fusion.
IBM Storage Scale Container Native (with IBM Storage Scale Container Storage Interface)	Version 5.1.5 or later fixes CSI Version 2.6.x or later fixes.	Available in either: • IBM Storage Fusion • IBM Storage Suite for watsonx.data
Portworx	• Version 2.9.1.3 or later fixes • Version 2.12.2 or later fixes	If you are running Red Hat OpenShift Container Platform Version 4.12, upgrade to Portworx Version 2.12.2 or later.
NFS	Version 3 or 4	Version 3 is recommended. If you use Version 4, ensure that your storage class uses NFS Version 3 as the mount option. For details, see Setting up dynamic provisioning .
Amazon Elastic Block Store (EBS)	Not applicable	In addition to EBS storage, your environment must also include EFS storage.
IBM Cloud Block Storage	Not applicable	In addition to IBM Cloud Block Storage, your environment must also include IBM Cloud File Storage.
NetApp Trident	Version 22.4.0 or later fixes	

When you plan your environment, ensure that you review the storage types supports components that you plan to install:

- Service persistent storage requirements

Important:

IBM evaluates the preceding storage options. You can run the watsonx.data storage validation tool to assess storage that is provided by other vendors. However, this tool does not support for other types of storage. You can use other storage environments at your own risk.

Sizing

The minimum amount of storage depends on the type of storage that you plan to use. For details, see the *Resource requirements* section under Storage comparison in "Storage considerations " on page 79.

Review the storage comparison and ensure that you have sufficient storage space available for user data based on the type of storage that you select. You can add extra capacity depending on the data volume requirements.

IBM watsonx.data persistent storage requirements

The following table lists the recommended block storage classes and file storage classes for each type of storage. If you use different storage class names on your cluster, ensure that you specify equivalent storage classes.

- Block storage supports ReadWriteOnce (RWO) access.

Storage option	Recommended block storage class	Notes
Red Hat OpenShift Data Foundation	`ocs-storagecluster-ceph-rbd`	The default storage class names are different if you use external mode.
IBM Storage Fusion Data Foundation	`ocs-storagecluster-ceph-rbd`	
IBM Storage Fusion Global Data Platform	`ibm-spectrum-scale-sc`	
IBM Storage Scale Container Native	`ibm-spectrum-scale-sc`	
Portworx	`portworx-metastoredb-sc`	For more information about the Portworx storage classes, see Creating storage classes.
NFS	`managed-nfs-storage`	
Amazon Elastic Block Store	`gp2-csi` or `gp3-csi`	
IBM Cloud Block Storage	`ibmc-block-gold`	Your environment must include IBM Cloud File Storage in addition to IBM Cloud Block Storage.
NetApp Trident	`ontap-nas`	

Software requirements

Before you install IBM watsonx.data, review the software requirements for the control plane, the shared cluster components, and the services that you plan to install, and review the supported web browsers.

watsonx.data on Red Hat OpenShift

watsonx.data platform software requirements

You must have the following software to install watsonx.data:

Red Hat OpenShift Container Platformcluster

For entitlement information, see "Licenses and entitlements" on page 77.

IBM watsonx.data supports the following versions of Red Hat OpenShift Container Platform.watsonx.data supports the same operating system requirements asRed Hat OpenShift Container Platform.

Important: IBM watsonx.data supports only the specified even releases of Red Hat OpenShift Container Platform.

Different versions of watsonx.data support different versions of Red Hat OpenShift Container Platform.

Version	Learn more.	Cluster sizing guidance
Version 4.12.0 or later fixes	For more information, see the Red Hat OpenShift Container Platform documentation: • Installation overview • Minimum hardware requirements • Control plane node sizing	For more information about the cluster size and configuration, see "Hardware requirements" on page 91.
Version 4.14.0 or later fixes	For more information, see the Red Hat OpenShift Container Platform documentation: • Installation overview • Minimum hardware requirements • Control plane node sizing	For more information about the cluster size and configuration, see "Hardware requirements" on page 91.
Version 4.15.0 or later fixes	For more information, see the Red Hat OpenShift Container Platform documentation: • Installation overview • Minimum hardware requirements • Control plane node sizing	For more information about the cluster size and configuration, see "Hardware requirements" on page 91.

Container runtime

Your Red Hat OpenShift Container Platform cluster must include a container runtime.

Software	Notes®
CRI-O Version 1.13 or later fixes	NA

Kubernetes Metrics Server

If you do not enable the default monitoring stack on your Red Hat OpenShift Container Platform cluster and you want to gather use metrics for your pods and nodes, install **Kubernetes Metrics Server**.

Important: If you do not enable the default monitoring stack or install Kubernetes Metrics Server, the platform monitoring features in watsonx.datado not work.

Supported web browsers

You can use the following web browsers to access the watsonx.data web client.

- Mozilla Firefox (recommended)
- Google Chrome
- Safari
- Microsoft Edge

Use the latest available version or the latest version that is approved by your enterprise.

Private container registry requirements

watsonx.data on Red Hat OpenShift

IBM watsonx.data for Data images are accessible from the IBM Entitled Registry. In most situations, it is recommended that you mirror the necessary software images from the IBM Entitled Registry to a private container registry.

Important:

Mirror the necessary images to your private container registry in the following situations:

- Your cluster is air-gapped (also called an offline or disconnected cluster).
- Your cluster uses an *allowlist* to permit direct access by specific sites, and the allowlist does not include the IBM Entitled Registry.
- Your cluster uses a *blocklist* to prevent direct access by specific sites, and the blocklist includes the IBM Entitled Registry.

Even if these situations do not apply to your environment, you should consider by using a private container registry if you want to:

- Run security scans against the software images before you install them on your cluster.
- Ensure that you have the same images available for multiple deployments, such as development or test environments and production environments.

The only situation in which you might consider pulling images directly from the IBM Entitled Registry is when your cluster is not air-gapped, your network is reliable, and latency is not a concern. However, for predictable and reliable performance, you should mirror the images to a private container registry.

If you decide to use a private container registry, review the guidance in the following sections:

- Cluster requirements
- Setting up a private container registry
- Allowing required image prefixes
- Choosing a method for mirroring images

Cluster requirements

To use a private container registry, your cluster must support image content source policies (`ImageContentSourcePolicy`).

Setting up a private container registry

For details about private container registries you can use with Red Hat OpenShift Container Platform, see the Red Hat OpenShift Platform documentation:

Version	Guidance
Version 4.10	1. Review the guidance in OpenShift Container Platform registry overview. 2. Ensure that you follow the guidelines for configuring the registry in Image configuration resources.
Version 4.12	1. Review the guidance in OpenShift Container Platform registry overview. 2. Ensure that you follow the guidelines for configuring the registry in Image configuration resources.

Your private container registry must meet the following requirements:

- Support the Docker Image Manifest Version 2, Schema 2.
- Allow path separators in image names.
- Be in close proximity to your Red Hat OpenShift Container Platform cluster.
- Be accessible from all nodes in the cluster, and all of the nodes must have permission to push to and pull from the private container registry.

Restriction: You cannot use the integrated OpenShift Container Platform registry. It does not support multi-architecture images and is not compliant with the Docker Image Manifest Version 2, schema 2.

Allowing required image prefixes

IBM watsonx.data software uses the following prefixes to identify images:

Tag	Used for
cp.icr.io/cp	Images that are pulled from the IBM Entitled Registry that require an entitlement key to download. Most of the IBM watsonx.data software uses this tag.
icr.io/cpopen	Publicly available images that are provided by IBM and that don't require an entitlement key to download. The IBM watsonx.dataoperators use this tag.

Ensure that the following statements are true:

- The private container registry is configured to allow these prefixes.
- The credentials that you use to push images to the registry can push images with these prefixes.

Choosing a method for mirroring images

There are several ways that you can mirror images from the IBM Entitled Registry to your private container registry. Choose the most appropriate method for your environment by answering the following question:

Can you set up a client workstation that can connect to the internet and the private container registry?

Yes

You can mirror the images directly from the IBM Entitled Registry to the private container registry.

No, the private container registry is in a restricted network

You must mirror the images directly from the IBM Entitled Registry to the private container registry.. The `cpd-cli manage mirror-images` command automatically sets up an intermediary container registry on the client workstation. You must be able to move the intermediary container registry behind your firewall. For example, you can use:

Options	Details
Use a portable compute device, such as a laptop, that you can move behind your firewall.	You can use the same device to: • Mirror images from the IBM Entitled Registry to the intermediary container registry. • Mirror images from the intermediary container registry to the private container registry.
Use a portable storage device, such as a USB drive, that you can move behind your firewall.	Set up two client workstations: • A workstation that can connect to the internet. From this workstation, you can mirror the images from the IBM Entitled Registry to the intermediary container registry on the portable storage device. • A workstation that can connect to the private container registry. After you move the portable storage device to this workstation, you can mirror the images from the intermediary container registry to the private container registry.
Use a file transfer protocol, such as `scp` or `sftp`, to move images behind your firewall.	Set up two client workstations: • A workstation that can connect to the internet. From this workstation, you can mirror the images from the IBM Entitled Registry to the intermediary container registry. • A workstation that can connect to the private container registry. After you transfer the intermediary container registry to this workstation, you can mirror the images from the intermediary container registry to the private container registry.

Client workstation requirements

You can use a client workstation, such as a laptop, to run the IBM watsonx.data command-line interface (`cpd-cli`).

watsonx.data on Red Hat OpenShift

Operating system requirements for client workstations

The hardware that you are running determines which operating system the client workstation can run.

Operating system	x86-64	Notes
Linux®	✓	
Mac OS	✓	

Operating system	x86-64	Notes
Windows	✓	To run on Windows, you must install Windows Subsystem for Linux.

Container runtime requirements for client workstations

Some of the `cpd-cli` plug-ins require a container runtime, such as Docker or Podman, on the client workstation.

It is recommended that you use the latest version of the container runtime that is compatible with your operating system.

The minimum recommended versions are:

- Docker

  Choose the Docker offering that is compatible with your operating system:

  - Docker Desktop Version 4.2.0 or later

    For information on installing Docker Desktop, see https://docs.docker.com/desktop/.
  - Docker Engine Version 20.0.0 or later

    For information on installing Docker Engine, see https://docs.docker.com/engine/.
- Podman Version 4.0.0 or later

  For information on installing Podman, see https://podman.io/getting-started/installation.

Notice: Windows users: If you are running the `cpd-cli` on the Windows operating system, you must set up the container runtime inside Windows Subsystem for Linux.

Internet access requirements for client workstations

At a minimum, the workstation must be able to access the cluster network. However, some of the plug-ins require public internet access to complete a subset of tasks, such as downloading files and containerized software images.

If your cluster is in a restricted network, you can:

- Move the workstation behind the firewall after you complete the tasks that require an internet connection.
- Prepare a client workstation that can connect to the internet and a client workstation that can connect to the cluster and transfer any files from the internet-connected workstation to the cluster-connected workstation.

Security on watsonx.data

IBM watsonx.data supports several different mechanisms for securing your environment and your data.

watsonx.data on Red Hat OpenShift

Basic security features on Red Hat OpenShift Container Platform

IBM watsonx.data builds on OpenShift security features that protect sensitive customer data with strong encryption controls and improves access control across applications and the platform.

watsonx.data on Red Hat OpenShift

Features

- Red Hat OpenShift Container Platform enables an improved security posture with the addition of many capabilities that greatly increases the security of the platform.

- Uses Red Hat CoreOS as the immutable host operating system.
- Provides stronger platform security with FIPS (Federal Information Processing Standard) compliant encryption (FIPS 140-2 Level 1).
- Uses the Node Tuning Operator, which provides opportunities to further reduce prisomvilege requirements in the security context constraints (SCC). For more information, see Using the Node Tuning Operator.
- Supports encrypting data that is stored in etcd, which provides extra protection for secrets that are stored in the etcd database. For more information, see Encrypting etcd data.
- Provides a Network Bound Disk Encryption (NBDE) feature that can be used to automate remote enablement of LUKS encrypted volumes, making it better protected against physical theft of host storage.
- Enables SELinux as mandatory on Red Hat OpenShift Container Platform.

Service accounts and roles

watsonx.data runs in a separate namespace or project on the OpenShift cluster. In the OpenShift project, watsonx.data creates service accounts and RBAC role bindings for pods to use within that namespace.

- No cluster level access is permitted. All roles impose a restriction to work within that namespace only.

- Two roles are created: cpd-admin-role and cpd-viewer-role. These roles allow watsonx.data to ensure that the principle of least privilege can be applied even within the same namespace.

- Four service accounts are created: zen-admin-sa, zen-editor-sa, zenviewer-sa, and zen-norbac-sa. No SCCs are explicitly bound to these service accounts and hence they pick up restricted SCC by default.

- The default service account that is automatically created in every OpenShift project is not granted any RBAC privileges; that is, no roles are bound.

Service UIDs

Services use UIDs based on the Red Hat OpenShift Container Platform project where they are installed.

When you create a project, Red Hat OpenShift assigns a unique range of UIDs to the project. To determine the UIDs that are associated with a project, run the following command:

The expectation is that the default service account is used for user workloads such as Notebooks and Python jobs. To perform any action inside the namespace, is not allowed.

The default service account is associated with the restricted security context constraints (SCCs). However, some add-on services might still need custom SCCs, for example to support PCs. The default service account is associated with the restricted security context constraints (SCCs). However, some add-on services might still need custom SCCs, for example to support IPCs. For more information, see Security context constraints in the IBM® Cloud Platform Common Services documentation.

Service UIDs

Services use UIDs based on the Red Hat OpenShift Container Platform project where they are installed.

When you create a project, Red Hat OpenShift assigns a unique range of UIDs to the project. To determine the UIDs that are associated with a project, run the following command:

```
oc describe project project-name
```

Replace `project-name` with the name of the project where watsonx.data is installed.

Security hardening

Security hardening is enforced on watsonx.data on Red Hat OpenShift. The following security hardening actions are taken:

- Only nonroot processes are run in containers. The UIDs of the processes are in the OpenShift Project's pre-defined range only, enforced by the use of the restricted SCCs. The restricted SCC does not allow running containers as root.
- Cluster Admin privileges are not required for watsonx.data workloads at run time. Cluster Admin authority is needed only to set up projects and custom SCCs (and only for the services that do need them). Service accounts in each watsonx.data instance are granted privileges that are only scoped within their OpenShift project.
- watsonx.data users are typically not granted OpenShift Kubernetes access, and even if they are, it would be only for express purpose of installing or upgrading services inside their assigned OpenShift project.
- Strict use of service accounts with RBAC privileges is enforced, and the least privilege principle is applied. watsonx.data ensures that any pod that is running user code (such as scripts or analytics environments) is not granted any RBAC privileges.
- No host access is necessary for watsonx.data workloads at run time. This restriction is enforced by the SCCs. There is no access to host paths or networks.
- All pods have a restricted resource consumption. Pod resource requests and limits are set for each pod, which restricts the consumption. This approach helps protect against noisy neighbors that cause resource contention.
- Reliability gauges (liveness and readiness probes) are present for each pod to ensure that the pods are working correctly.
- For consumption Monitoring, each of the pods on watsonx.data is annotated with metering annotations to uniquely identify add-on service workloads on the cluster.

Prescriptive security practices during installation

You don't need an SSH to OpenShift cluster nodes to deploy or manage watsonx.data and its add-on services. The OpenShift `oc` command-line interface and the `ibmpak` command-line interface are used to deploy and manage the IBM watsonx.data platform operator and its services.

You can install the software on a cluster that is connected to the internet or a cluster that is air-gapped.

Installing watsonx.data on clusters that are connected to the internet

Recommended actions for extra security when you access images and to ensure reliability. It is highly recommended that you use the `ibmpak` utility from a client workstation to download the CASE packages and mirror images from the IBM Entitled Registry and other public container registries into your private container registry.

Installing watsonx.data on air-gapped clusters

watsonx.data supports mirroring of images to your private container registry. This procedure does not require that your private container registry and Red Hat OpenShift Container Platform cluster are able to access the internet. You are able to download the CASE packages and mirror images by using the `ibmpak` utility from a bastion node. If the bastion node does not have a direct access to the private container registry, then an intermediary container registry can be used. By using the intermediary container registry, you are able to mirror the images first on the bastion node. You need to make sure that your private container registry is accessible from your internal network, but not from public internet. Therefore, your Red Hat OpenShift cluster would be configured to pull from the private container registry. That way you are able to install the watsonx.data operators and services without access to the internet.

Namespace scope and Operator privileges

watsonx.data uses the Operator pattern to manage its workloads, which allows for a separation of concerns. That way Operators that are resident in one central namespace are granted access to manage the watsonx.data Service workloads in multiple different "instance" namespaces. Users in those instance namespaces can thus be granted far lesser privileges, scoped within that namespace. Operators too are scoped to operate only within these specific namespaces and are, by design, not permitted to manage non-watsonx.data namespaces in the cluster. You need the following scope and user privileges.

- To install the Operator Catalog Source, you need a Red Hat OpenShiftContainer Platform user, with privileges in the **openshift-marketplace**namespace.
- To create an "own Namespace" mode for the OperatorGroup and any Subscriptions for the needed Operators, you need users with privileges in either the **ibm-common-services** or **cpd-operators**.
- **Note:** For security reasons, the "All Namespaces" mode for the OperatorGroup is not recommended.
- To to restrict the namespaces that the watsonx.data Operators have authority over, use the IBM Cloud Pak foundational services Namespace-scope Operator. You can expand the Operator access to more watsonx.data instance namespaces where the service workloads are then deployed. For more information, see Authorizing foundational services to perform operations on workloads in a namespace.
- To create custom resources to deploy the individual watsonx.data services or to upgrade or scale them postinstallation, you need users with Project admin privileges in the "instance" namespaces.

Cluster admin responsibility

Cluster Admins are expected to manage the OpenShift cluster and prepare it for use by the watsonx.data services. This preparation includes the following tasks:

Node tuning and machine pool configurations for kernel settings and cri-o settings (such as pids-limit, ulimit).

Only for services that need them.

Set up the image content source policy and any secrets.

The setup is done to pull images from the private container registry.

Create OpenShift Projects.

For the IBM Cloud Pak foundational services and Operators.

For the instances of watsonx.data.

Configure the namespaces.

Define namespace quotas and Limit Ranges.

Grant access of the watsonx.data Admins to specific instance namespaces.

Create custom SCCs.

Only for services that need them.

Storage installation and configuration.

Storage that is used by the workloads.

Securely manage OpenShift.

Handle encryption and auditing as well as other operations such as adding nodes, replacing nodes and others.

IBM Containerized Software Security Summary

The Security Summary is a PDF file that explains the security posture of an IBM Containerized product. The summary indicates compliance with several IBM Certification security metrics. The security metrics that are included are based on IBM standards, guidelines, and best practices for delivering secure, enterprise grade software for Red Hat OpenShift Container Platform.

watsonx.data on Red Hat OpenShift

Enabling FIPS on your Red Hat OpenShift cluster

To run IBM watsonx.data on a Federal Information Processing Standards (FIPS) compliant system, use the following Red Hat OpenShift planning information steps.

watsonx.data on Red Hat OpenShift

Auditing watsonx.data

Auditing is the process of recording the activity that occurs on databases or applications. Auditing can help you detect and prioritize security threats and data breaches.

watsonx.data on Red Hat OpenShift

Network security

You can use network policies to control which connections are allowed or rejected.

watsonx.data on Red Hat OpenShift

Remember: You can optionally deploy multiple instances of IBM watsonx.data on your cluster. If you have more than one instance of watsonx.data on your cluster, you need to adjust your network policies to ensure that each instance works as expected. For background information about multitenancy, see Multitenancy support.

Network policies

If the network isolation mode for the OpenShift cluster is set to `NetworkPolicy` (the default), you can control the flow of traffic between different projects (namespaces). To define the flow, use the `NetworkPolicy` custom resource.

Learn more about network policies in the Red Hat OpenShift Container Platform documentation.

- Version 4.10

  4.6.x
- Version 4.12

  4.6.4 or later

By default, all pods in a project are accessible from other pods and network endpoints. To isolate one or more pods in a project, you can create `NetworkPolicy` objects in that project to indicate the allowed incoming connections.

A project administrator can create and manage the `NetworkPolicy` objects in the projects that they manage.

Network policies are cumulative, so you can combine multiple network policies to satisfy complex network requirements.

Network policies to isolate an instance of watsonx.data

Typically, each instance of watsonx.data on the cluster would be isolated within the project where it is installed.

You can use network policies to control which connections are allowed or rejected. It is advisable to restrict connections to known authorized individuals or end points to further reduce the risk of potential vulnerabilities in the deployed system.

A project administrator can create the following network policies to isolate an instance of watsonx.data:

- Deny all traffic.
- Only accept connections from pods within the same namespace.
- Only accept connections from Red Hat OpenShift Container Platform
- Only accept ingress traffic at the front door.

These policies are combined to provide controlled access to the instance. You can repeat this process for each instance of watsonx.data on the cluster.

Deny all traffic

To make the project deny all traffic by default, define a network policy that applies to all pods in that project and rejects all incoming traffic to the project:

```
cat <<EOF |oc apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-by-default
spec:
  podSelector:
    ingress: []
EOF
```

Only accept connections from pods within the same project

To make pods in the project accept connections from other pods in the same project, but reject connections from pods in other projects, define the following network policy.

```
cat <<EOF |oc apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-same-namespace
spec:
  podSelector: {}
  policyTypes:
    - Ingress
  ingress:
    - from:
      - podSelector: {}
EOF
```

To make pods in the project accept connections from the Red Hat OpenShift Container Platform monitoring stack, define the following network policy:

```
cat <<EOF |oc apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  ingress:
  - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: monitoring
EOF
```

Only accept ingress traffic at the front door

To accept ingress traffic at the front door, define the following network policy.

```
cat <<EOF |oc apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-access-to-front-door
spec:
  podSelector:
    matchLabels:
      component: "ibm-nginx"
  policyTypes:
    - Ingress
  ingress:
    - {}
EOF
```

Additional security measures to consider

Consider whether you need to take additional steps to restrict egress from watsonx.data. In some environments, such as clusters running in a restricted network, this might be unnecessary; your existing network configuration or firewall rules should be sufficient to restrict egress.

It is recommended that you review the Red Hat OpenShift security guide to determine which security measures you should implement in your environment.

Firewall requirements for watsonx.data

Configure the firewall rules for tour your data center or proxy access control list (ACL) to meet the requirements for IBM watsonx.data.

The following table lists outbound access for Red Hat® OpenShift® and watsonx.data.

watsonx.data on Red Hat OpenShift

Container port	Protocol	Port
Minio	TCP	9000, 9001
HMS	TCP	8281, 9083
QHMM	TCP	8443, 8444
lhconsole-api	TCP	3333
lhconsole-nodeclient	TCP	3001
lhconsole-ui	TCP	8443
ibm-lh-milvus	TCP	19530
Presto	TCP	8481, 8443

Proxy server configuration

See Setting up a client workstation to install Cloud Pak for Data for more information.

Installing watsonx.data

watsonx.data on Red Hat OpenShift

Setting up a client workstation

To install watsonx.data, you must have a client workstation that can connect to the Red Hat OpenShift Container Platform cluster.

watsonx.data on Red Hat OpenShift

You must have the following command-line interfaces on the client workstation:

1. Install the IBM Cloud Pak for Data command-line interface.
2. Install the OpenShift command-line interface.

Setting up a cluster

Before you install watsonx.data, you must install and setup a Red Hat OpenShift Container Platform cluster.

watsonx.data on Red Hat OpenShift

Procedure

Complete the appropriate tasks for your environment:

1. Installing Red Hat OpenShift Container Platform Installing Red Hat OpenShift Container Platform
2. Installing persistent storage
3. Setting up a private container registry

Collecting required information

To successfully install watsonx.data, you must have specific information about your environment.

watsonx.data on Red Hat OpenShift

Procedure

Complete the following tasks to ensure that you have the information that you need.

1. Obtaining your IBM entitlement API key.
2. Setting up installation environment variables.

Preparing to run watsonx.data installs from a private container registry

If you plan to use a private container registry to host the watsonx.data software images, you must mirror the images from the IBM Entitled Registry and configure the cluster to pull the images from the private container registry.

watsonx.data on Red Hat OpenShift

Procedure

To prepare to run installs from a private container registry:

1. Preparing to run cpd-cli manage commands in a restricted network
2. Mirroring images to a private container registry
3. Configuring an image content source policy
4. Pulling the olm-utils-v2 image from the private container registry

Preparing your cluster

Before you install watsonx.data, complete the following tasks to prepare your cluster.

watsonx.data on Red Hat OpenShift

1. Updating the global image pull secret
2. Setting up persistent storage

Creating image pull secret for IBM Cloud based installation

With OpenShift Container Platform, you can create a global image pull secret that each worker node in the cluster can use to pull images from a private registry.

watsonx.data on Red Hat OpenShift

Before you begin

- Download oc from OpenShift UI and `IBM_Cloud_CLI` .
- Install `ibmcloud plugin install container-service.`

Download the jq JSON processor command-line package.

Procedure

1. Create a secret value that holds the credentials to access your private registry and store the decoded secret value in a JSON file. When you create the secret value, the credentials are automatically encoded to `base64`. By using the `--dry-run` option, the secret value is created only and no secret object is created in your cluster. The decoded secret value is then stored in a JSON file to later use in your global pull secret.

```
oc create secret docker-registry <secret_name> \
--docker-server=${PROD_REGISTRY} \
--docker-username=${PROD_USER} \
--docker-password=${IBM_ENTITLEMENT_KEY} \
--docker-email=<email> \
--dry-run=client \
--output="jsonpath={.data.\.dockerconfigjson}" | base64 \
--decode > myregistryconfigjson
```

2. Retrieve the decoded secret value of the default global pull secret and store the value in a `dockerconfigjson` file.

```
oc get secret pull-secret -n openshift-config \
--output="jsonpath={.data.\.dockerconfigjson}" | base64 \
--decode > dockerconfigjson
```

3. Combine the downloaded private registry pull secret `myregistryconfigjson` file with the default global pull secret `dockerconfigjson` file.

```
jq -s '.[0] * .[1]' dockerconfigjson myregistryconfigjson > dockerconfigjson-merged
```

4. Update the global pull secret with the combined `dockerconfigjson` merged file.

```
oc set data secret/pull-secret -n openshift-config --from-
file=.dockerconfigjson=dockerconfigjson-merged
```

5. Verify that the global pull secret is updated. Check that your private registry and each of the default Red Hat registries are in the output of the following command.

```
oc get secret pull-secret -n openshift-config \
--output="jsonpath={.data.\.dockerconfigjson}" | base64 \
--decode
```

6. To pick up the global configuration changes, reload all the worker nodes in your cluster.

   a) Note the ID of the worker nodes in your cluster.

   ```
   ibmcloud oc worker ls -c <cluster_name_or_ID>
   ```

   b) Reload each worker node. You can reload multiple worker nodes by including multiple `-w` flags, but make sure to keep enough worker nodes running at the same time for your apps to avoid an outage.

   Note: For IBM VPC type clusters, the Reload option is not available. You can use the Replace option.

   ```
   ibmcloud oc worker reload -c <cluster_name_or_ID>
   -w <workerID_1> -w <workerID_2>
   ```

7. After the worker nodes are back in a working state, verify that the global pull secret is updated on a worker node.

   a) Start a debugging pod to log in to a worker node. Use the Private IP that you retrieved earlier for the `<node_name>`.

   ```
   oc debug node/<node_name>
   ```

   b) Change the root directory to the host so that you can view files on the worker node.

   ```
   chroot /host
   ```

   c) Verify that the Docker configuration file has the registry credentials that match the global pull secret that you set.

```
cat /.docker/config.json
```

Prerequisite for installing watsonx.data on Red Hat OpenShift Kubernetes Service (ROKS)

Before installing IBM watsonx.data on ROKS, complete the following steps.

Procedure

1. Export the following environment variables:

   ```
   export STG_REGISTRY=cp.stg.icr.io
   export ARTIFACTORY_REGISTRY=hyc-cp4d-team-bootstrap-2-docker-local.artifactory.swg-devops.com
   export ARTIFACTORY_CPFS_REGISTRY=docker-na-public.artifactory.swg-devops.com/hyc-cloud-
   private-daily-docker-local/ibmcom
   export STG_USER=<your user name>
   export STG_API_KEY=<your stg api key>
   export ARTIFACTORY_USER=<your ARTIFACTORY USER>
   export ARTIFACTORY_TOKEN=<your ARTIFACTORY api key>
   ```

2. Using `cpd-cli`, log in to OpenShift Container Platform and configure credentials by using the following commands.

   a.
   ```
   ${CPD_CLI_EXE} manage login-to-ocp --token=${OCP_TOKEN} --server=${OCP_URL}
   ```

   b.
   ```
   podman login cp.stg.icr.io -u $STG_USER -p $STG_API_KEY
   ```

   c.
   ```
   ${CPD_CLI_EXE} manage add-cred-to-global-pull-secret --registry=${STG_REGISTRY} --
   registry_pull_user=${STG_USER} --registry_pull_password=${STG_API_KEY}
   ```

   d.
   ```
   ${CPD_CLI_EXE} manage add-cred-to-global-pull-secret --registry=${ARTIFACTORY_REGISTRY} --
   registry_pull_user=${ARTIFACTORY_USER} --registry_pull_password=${ARTIFACTORY_TOKEN}
   ```

   e.
   ```
   ${CPD_CLI_EXE} manage add-cred-to-global-pull-secret --registry=$
   {ARTIFACTORY_CPFS_REGISTRY} --registry_pull_user=${ARTIFACTORY_USER} --
   registry_pull_password=${ARTIFACTORY_TOKEN}
   ```

3. Run the following command to install the IBM Cloud Container Service plug-in in the IBM Cloud CLI.

   ```
   ibmcloud plugin install container-service
   ```

4. Run the following command to log in to the IBM Cloud CLI using a single sign-on (SSO).

   ```
   ibmcloud login --sso
   ```

5. To list the clusters, run:

   ```
   ibmcloud ks cluster ls
   ```

6. To list the worker nodes in the cluster, run:

   ```
   ibmcloud oc worker ls -c <cluster_name>
   ```

7. Run the following command to reload every worker node in the cluster.

   ```
   ibmcloud cs worker reboot --cluster <cluster_name_or_ID> -w <workerID>
   ```

8. Configure image mirroring on the target cluster by updating `registries.conf` on each worker node.

   a) Run the following command to note the IP and ID of the worker nodes in your cluster.

   ```
   ibmcloud oc worker ls -c <cluster_name_or_ID>
   ```

   b) Run the following commands to configure image registry mirror on every worker node.

i) Use **oc debug** to enter the worker node.

```
oc debug node/<worker_IP>
```

ii) Change `root` to `/host`.

```
chroot /host
```

iii) Edit `registries.conf`.

```
vi etc/containers/registries.conf
```

Use the following content.

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
short-name-mode = ""

[[registry]]
  prefix = ""
  location = "cp.icr.io/cp"

  [[registry.mirror]]
    location = "cp.stg.icr.io/cp"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "hyc-cp4d-team-bootstrap-docker-local.artifactory.swg-devops.com"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "hyc-cp4d-team-databases-docker-local.artifactory.swg-devops.com"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "hyc-cp4d-team-bootstrap-2-docker-local.artifactory.swg-devops.com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "cp.icr.io/cp/cpd"

  [[registry.mirror]]
    location = "cp.stg.icr.io/cp/cpd"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "hyc-cp4d-team-bootstrap-docker-local.artifactory.swg-devops.com"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "hyc-cp4d-team-bootstrap-2-docker-local.artifactory.swg-devops.com"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "hyc-cp4d-team-databases-docker-local.artifactory.swg-devops.com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "docker.io/ibmcom"

  [[registry.mirror]]
    location = "cp.stg.icr.io/cp"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "hyc-cp4d-team-bootstrap-2-docker-local.artifactory.swg-devops.com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "hyc-cloud-private-daily-docker-local.artifactory.swg-devops.com/ibmcom"

  [[registry.mirror]]
    location = "docker-na-public.artifactory.swg-devops.com/hyc-cloud-private-daily-
docker-local/ibmcom"
    pull-from-mirror = "digest-only"
```

```
[[registry]]
  prefix = ""
  location = "icr.io/cpopen"

  [[registry.mirror]]
    location = "cp.stg.icr.io/cp/cpd"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "cp.stg.icr.io/cp"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "docker-na-public.artifactory.swg-devops.com/hyc-cloud-private-daily-
docker-local/ibmcom"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "hyc-cp4d-team-bootstrap-2-docker-local.artifactory.swg-devops.com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "icr.io/cpopen/cpfs"

  [[registry.mirror]]
    location = "cp.stg.icr.io/cp/cpd"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "docker-na-public.artifactory.swg-devops.com/hyc-cloud-private-daily-
docker-local/ibmcom"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "hyc-cp4d-team-bootstrap-2-docker-local.artifactory.swg-devops.com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "icr.io/db2u"

  [[registry.mirror]]
    location = "cp.stg.icr.io/cp/cpd"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "cp.stg.icr.io/cp"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "quay.io/opencloudio"

  [[registry.mirror]]
    location = "hyc-cp4d-team-bootstrap-2-docker-local.artifactory.swg-devops.com"
    pull-from-mirror = "digest-only"

  [[registry.mirror]]
    location = "docker-na-public.artifactory.swg-devops.com/hyc-cloud-private-daily-
docker-local/ibmcom"
    pull-from-mirror = "digest-only"
```

c) Exit the debug session by running:

```
exit
```

d) Reboot the worker node by running:

```
ibmcloud cs worker reboot --cluster <cluster_name_or_ID> -w <workerID>
```

To proceed with the installation, see Installing watsonx.data on Red Hat Open Shift.

Installing IBM Cloud Pak for Data and watsonx.data on Red Hat OpenShift

A cluster administrator can install IBM watsonx.data on a Red Hat OpenShift cluster.

watsonx.data on Red Hat OpenShift

Before you begin

- Install the IBM Cloud Pak for Data command-line interface.
- Set up the installation environment variables.
- Determine which watsonx.data components to install.
- To install watsonx.data on Red Hat OpenShift Kubernetes Service (ROKS), ensure to meet the prerequisite.

Restriction: If you are installing Red Hat OpenShift Container Platform Version 4.14, do not install the KubeVirt HyperConverged Cluster Operator on the cluster. It can cause problems when installing some Cloud Pak for Data software. (A previous version of this note stated that you could not install the OpenShift Virtualization Operator. However, that restriction no longer applies.)

Important: IBM watsonx.data can be co-located with watsonx.ai and watsonx.governance, and any supporting services that are included in the watsonx.data license. IBM watsonx.data cannot be co-located with Cloud Pak for Data Enterprise Edition, Cloud Pak for Data Standard Edition, or services included in other cartridge licenses. If you have an existing Cloud Pak for Data Enterprise Edition or Cloud Pak for Data Standard Edition installation, you must install the control plane and watsonx.data in its own operators project and operands project (namespace). Services included in the IBM watsonx.data license: Analytics Engine powered by Apache Spark (Starting with version 1.1.4, Analytics Engine powered by Apache Spark is automatically installed when you install watsonx.data.)

Note: If you are installing services other than watsonx.data, Analytics Engine, watsonx.ai, or watsonx.ai dependencies, install those in their own namespaces and then set `$PROJECT_CPD_INST_OPERATORS` and `$PROJECT_CPD_INST_OPERANDS` to different namespaces and run the installation steps.

Procedure

1. Log in to the `ocp` cluster as a user with sufficient permissions to complete this task.

   Option 1: Run the following command to log in to the cluster by providing a username and password:

   ```
   cpd-cli manage login-to-ocp \
   --username=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

   Option 2: Run the following command to log in to the cluster by providing a token:

   ```
   cpd-cli manage login-to-ocp \
   --server=${OCP_URL} \
   --token=${OCP_TOKEN}
   ```

2. Install the certificate manager and the license service.

   ```
   cpd-cli manage apply-cluster-components \
   --release=${VERSION} \
   --license_acceptance=true \
   --cert_manager_ns=${PROJECT_CERT_MANAGER} \
   --licensing_ns=${PROJECT_LICENSE_SERVICE}
   ```

3. Optional: Install the scheduling service.

   ```
   cpd-cli manage apply-scheduler \
   --release=${VERSION} \
   --license_acceptance=true \
   --scheduler_ns=${PROJECT_SCHEDULING_SERVICE}
   ```

4. Run the **`cpd-cli manage authorize-instance-topology`** command to apply the required permissions to the projects.

   ```
   cpd-cli manage authorize-instance-topology \
   --cpd_operator_ns=${PROJECT_CPD_INST_OPERATORS} \
   --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS}
   ```

5. Run the **`cpd-cli manage setup-instance-topology`** command to install IBM Cloud Pak foundational services and create the `ConfigMap`.

```
cpd-cli manage setup-instance-topology \
--release=${VERSION} \
--cpd_operator_ns=${PROJECT_CPD_INST_OPERATORS} \
--cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
--license_acceptance=true \
--block_storage_class=${STG_CLASS_BLOCK}
```

6. To install Cloud Pak for Data control plane and watsonx.data, run the following command:

```
export COMPONENTS=cpd_platform,watsonx_data
```

Note:

- For version 1.1.3 and earlier, if Spark is required, add `analyticsengine` to the components list.
- For version 1.1.4 and later, Spark is installed automatically as a dependency of watsonx.data. Adding the `analyticsengine` component is not required.

7. Install Cloud Pak for Data platform operator and service operator.

```
cpd-cli manage apply-olm \
--release=${VERSION} \
--cpd_operator_ns=${PROJECT_CPD_INST_OPERATORS} \
--components=${COMPONENTS}
```

8. Install the operands in the operands project for the instance.

```
cpd-cli manage apply-cr \
--release=${VERSION} \
--cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
--components=${COMPONENTS} \
--block_storage_class=${STG_CLASS_BLOCK} \
--file_storage_class=${STG_CLASS_FILE} \
--license_acceptance=true
```

9. Apply the entitlement for watsonx.data on Red Hat OpenShift.

```
cpd-cli manage apply-entitlement \
--cpd_instance_ns=<project-name> \
[--entitlement=watsonx-data] \
[--production=true|false] \
[--preview=true|false]
```

Uninstalling watsonx.data

If you need to uninstall IBM watsonx.data, you can remove the custom resources and the Operator Lifecycle Manager (OLM) objects that are associated with the components.

watsonx.data on Red Hat OpenShift

Uninstalling the components

If you want to uninstall the IBM watsonx.data software from your cluster, you must uninstall the running instance of the control plane and services.

watsonx.data on Red Hat OpenShift

About this task

If you plan to uninstall the watsonx.data operators, you must uninstall all instances of watsonx.data *before* you uninstall the operators.

Use the `cpd-cli manage delete-cr` command to remove the custom resources.

The instructions assume that you are removing all of the components at the same time, which enables you to complete the task in fewer steps.

Procedure

To uninstall watsonx.data components. The command that you run depends on where the operators are installed.

1. Run the following command to log in to OpenShift Container Platform (OCP) with your API token as a user with sufficient permissions to complete this task. For example,

   ```
   cpd-cli manage login-to-ocp --token=<access_token> --server=<cluster_url>
   ```

2. Run the following command to determine which components are installed in the project.

   ```
   cpd-cli manage get-cr-status --cpd_instance_ns <project name>
   ```

3. Set the `COMPONENTS` environment variable to include the components that were returned by the `get-cr-status` command. If you want to uninstall watsonx.data, set `export COMPONENTS=watsonx_data`.

   ```
   export COMPONENTS=<components>
   ```

4. Set the cpd instance namespace to where watsonx.data instance is installed.

   ```
   export PROJECT_CPD_INST_OPERANDS=<watsonx.data_instance_namespace>
   ```

5. Delete the custom resources for the specified components in the project.

   ```
   cpd-cli manage delete-cr --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} --components=$
   {COMPONENTS}
   ```

6. Run the `get-cr-status` to confirm that all of the components were removed.

   ```
   cpd-cli manage get-cr-status --cpd_instance_ns=<project name>
   ```

7. Clean up any remaining resources in the project.

   a) Set the RESOURCE_LIST environment variable.

   ```
   export
   RESOURCE_LIST=configmaps,persistentvolumeclaims,pods,secret,serviceaccounts,Service,Statef
   ulSets,deployment,job,cronjob,ReplicaSet,Route,RoleBinding,Role,PodDisruptionBudget,Operan
   dRequest
   ```

   b) Run the following command to identify any remaining resources in the `PROJECT_CPD_INST_OPERANDS` project.

   ```
   oc get ${RESOURCE_LIST} -n ${PROJECT_CPD_INST_OPERANDS} --ignore-not-found
   ```

   c) Run the following command to delete any resources that are returned.

   ```
   oc delete -n ${PROJECT_CPD_INST_OPERANDS} <object-type> <object-name>
   ```

8. Remove the PROJECT_CPD_INSTANCE project.

   ```
   oc delete project ${PROJECT_CPD_INST_OPERANDS}
   ```

Results

The components are uninstalled from the `${PROJECT_CPD_INST_OPERANDS}` project.

Uninstalling the OLM objects

If you want to completely remove the IBM watsonx.data software from your cluster, you must remove the Operator Lifecycle Manager (OLM) objects, such as operators, operator subscriptions, catalog sources, and cluster service versions.

watsonx.data on Red Hat OpenShift

About this task

Use the `cpd-cli manage delete-olm-artifacts` command to remove the following OLM objects for the specified components:

- Catalog sources
- Cluster service versions
- Operator subscriptions

The instructions assume that you are removing the OLM objects for all of the components at the same time, which enables you to complete the task in fewer steps.

Procedure

Run the command to delete the OLM objects for the specified components.

```
export COMPONENTS=<components>
export PROJECT_CPD_OPS=<operator_namespace>
cpd-cli manage delete-olm-artifacts
```

Getting started with watsonx.data

The following resources can help you get familiar with IBM watsonx.data and get productive.

watsonx.data on Red Hat OpenShift

Setting your preferences

You can customize the IBM watsonx.data web client to work for you.

watsonx.data on Red Hat OpenShift

You can customize your:

- Home page
- Profile photo

Customizing the contents of your home page

You can customize the cards that are displayed on your home page and the links that are displayed in the **Quick navigation** section.

To customize your home page:

1. Click the **Settings** icon:
   - To customize the cards on your home page:
     a. Select **Personalize cards**:
        - You can add new cards from the list of **Available cards**.
        - You can drag the cards in the **Overview** section to reorder them.

     b. Click **Save and close**.
   - To customize the links in the **Quick navigation**:
     a. Select **Personalize navigation**:
        - You can select up to 10 links from the **Add links** list.
        - You can drag the links in the **Quick navigation** section to reorder them.

     b. Click **Save and close**.

Changing your profile photo

Your profile photo is displayed next to projects that you create, notifications about actions that you've taken, and more. Your profile photo can make it easier for your coworkers to identify content that you contributed to.

To change your profile photo:

1. Click your avatar in the toolbar.
2. Click **Profile and settings**.
3. Click the **Replace photo** icon next to the current image.
4. Browse for or drag and drop the image that you want to use. The image can be up to 1 MB and must be a JPG or PNG file.
5. Click **Save**.

You might need to refresh your browser for the change to take effect.

Checking your permissions

You can see the permissions that you have from your profile. Your permissions are determined by the roles that are assigned to you.

watsonx.data on Red Hat OpenShift

To see the roles that are assigned to you, complete the following steps.

1. Click your avatar in the toolbar.
2. Click **Profile and settings**.
3. Open the **Roles** tab. The permissions that are associated with your role (or roles) are listed in the **Enabled permissions** column.

If you are a member of any user groups, you inherit the roles that are assigned to that group. These roles are also displayed on the **Roles** tab, and the group from which you inherit the role is specified in the **User groups** column. (A dash in **User groups** column means that the role is assigned directly to you.)

Generating API keys for authentication

With an API key, you can automatically authenticate to the IBM watsonx.data platform from a script or application. Your API keys are associated with your credentials and are specific to you. With API keys, you can authenticate without entering your password.

watsonx.data on Red Hat OpenShift

You can generate:

- A platform API key
- An instance API key

Your platform API key enables scripts and applications to access everything that you would typically be able to access when you log in to the watsonx.data web client. An instance API key enables access only to the specific instance from which is it generated.

Before you generate an API key, consider the following factors:

What level of access is required?

As a security best practice, it is recommended that you give the least amount of access necessary to the script or application. For example, if a script needs to interact only with a specific service instance, generate an API key for that service instance.

What type of key can you generate?

Some services do not support instance-level API keys. If you cannot generate an API key that is specific to the service, you must use your platform API key.

Important:

Be sure to store your API keys somewhere safe. If you lose an API key, you need to create a new key and update any scripts or applications where the key is used.

Platform API key

With a platform API key, you can access everything that you would typically be able to access when you log in to the watsonx.data web client.

To generate a platform API key, complete the following steps.

1. Log in to the web client.
2. From the toolbar, click your avatar.
3. Click **Profile and settings**.
4. Click **API key > Generate new key**.
5. Click **Generate**.
6. Click **Copy** and save your key. You cannot recover this key if you lose it.

If you lose your API key, repeat the preceding steps to generate a new API key. Your old API key becomes invalid, and any applications or scripts cannot authenticate to the platform until you provide your new API key.

If you believe that your API key is compromised, complete the following steps.

1. Log in to the web client.
2. From the toolbar, click your avatar.
3. Click **Profile and settings**.
4. Click **API key > Revoke current key**.
5. Click **Revoke**.

Any applications or scripts that use the key cannot authenticate to the platform.

Instance API keys

If you have access to a specific instance of a service, you can generate an API key to access only that service instance.

To generate an instance API key, complete the following steps.

1. Log in to the web client.
2. From the navigation menu, select **Services > Instances**.
3. Click the name of the instance for which you want to generate an API key.
4. Click **Instance API key > Generate API key**.
5. Click **Generate**.
6. Click **Copy** and save your key. You cannot recover this key if you lose it.

If you lose your API key, repeat the preceding steps to generate a new API key. Your old API key becomes invalid, and any applications or scripts cannot authenticate to the instance until you provide your new API key.

If you believe that your API key is compromised, complete the following steps.

1. Log in to the web client.
2. From the navigation menu, select **Services > Instances**.
3. Click the name of the instance for which you want to revoke your current API key.
4. Click **Instance API key > Revoke API key**.
5. Click **Revoke**.

Any applications or scripts that use the key cannot authenticate to the platform.

Upgrading watsonx.data from version 1.0.x or 1.1.x to 2.0.x

An instance administrator can upgrade IBM watsonx.data from version 1.0.x or 1.1.x to 2.0.x.

watsonx.data on Red Hat OpenShift

Before you begin

Complete the following steps:

1. Upgrading shared cluster components.
2. Upgrading the IBM Cloud Pak foundational services.
3. Upgrading IBM Cloud Pak for Data

Procedure

Complete the following steps to upgrade watsonx.data:

1. Log in to the Red Hat OpenShift Container Platform cluster:

   ```
   cpd-cli manage login-to-ocp \
   --username=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

2. Update the custom resource for watsonx.data.

   The command that you run depends on the storage on your cluster:

   - Red Hat OpenShift Data Foundation Storage

     Run the following command to create the custom resource.

     ```
     cpd-cli manage apply-cr \
     --components=watsonx_data \
     --release=${VERSION} \
     --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
     --block_storage_class=${STG_CLASS_BLOCK} \
     --file_storage_class=${STG_CLASS_FILE} \
     --license_acceptance=true \
     --upgrade=true
     ```

   - IBM Storage Fusion Data Foundation storage

     Run the following command to create the custom resource.

     ```
     cpd-cli manage apply-cr \
     --components=watsonx_data \
     --release=${VERSION} \
     --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
     --block_storage_class=${STG_CLASS_BLOCK} \
     --file_storage_class=${STG_CLASS_FILE} \
     --license_acceptance=true \
     --upgrade=true
     ```

   - IBM Storage Fusion Global Data Platform storage

     Remember: When you use IBM Storage Fusion storage, both `${STG_CLASS_BLOCK}` and `${STG_CLASS_FILE}` point to the same storage class, typically `ibm-spectrum-scale-sc` or `ibm-storage-fusion-cp-sc`.

     Run the following command to create the custom resource.

     ```
     cpd-cli manage apply-cr \
     --components=watsonx_data \
     --release=${VERSION} \
     --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
     --block_storage_class=${STG_CLASS_BLOCK} \
     ```

```
--file_storage_class=${STG_CLASS_FILE} \
--license_acceptance=true \
--upgrade=true
```

- IBM Storage Scale Container Native storage

  Remember: When you use IBM Storage Scale Container Native storage, both ${STG_CLASS_BLOCK} and ${STG_CLASS_FILE} point to the same storage class, typically `ibm-spectrum-scale-sc`.

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --block_storage_class=${STG_CLASS_BLOCK} \
  --file_storage_class=${STG_CLASS_FILE} \
  --license_acceptance=true \
  --upgrade=true
  ```

- Portworx storage

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --storage_vendor=portworx \
  --license_acceptance=true \
  --upgrade=true
  ```

- NFS storage

  Remember: When you use NFS storage, both ${STG_CLASS_BLOCK} and ${STG_CLASS_FILE} point to the same storage class, typically `managed-nfs-storage`.

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --block_storage_class=${STG_CLASS_BLOCK} \
  --file_storage_class=${STG_CLASS_FILE} \
  --license_acceptance=true \
  --upgrade=true
  ```

- AWS with EFS and EBS storage

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --block_storage_class=${STG_CLASS_BLOCK} \
  --file_storage_class=${STG_CLASS_FILE} \
  --license_acceptance=true \
  --upgrade=true
  ```

- NetApp Trident

  Remember: When you use NetApp Trident storage, both ${STG_CLASS_BLOCK} and ${STG_CLASS_FILE} point to the same storage class.

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --block_storage_class=${STG_CLASS_BLOCK} \
  ```

```
--file_storage_class=${STG_CLASS_FILE} \
--license_acceptance=true \
--upgrade=true
```

3. Validate the upgrade.

   watsonx.data is upgraded when the `apply-cr` command returns:

   ```
   [SUCCESS]... The apply-cr command ran successfully
   ```

   If you want to confirm that the custom resource status is `Completed`, you can run the `cpd-cli manage get-cr-status` command:

   ```
   cpd-cli manage get-cr-status \
   --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
   --components=watsonx_data
   ```

Upgrading watsonx.data from version 2.0.x to a later 2.0.x refresh

An instance administrator can upgrade IBM watsonx.data from version 2.0.x to a later 2.0.x refresh.

Before you begin

Complete the following steps:

1. Upgrading shared cluster components.
2. Upgrading the IBM Cloud Pak foundational services.
3. Upgrading IBM Cloud Pak for Data

Procedure

Complete the following steps to upgrade watsonx.data:

1. Log in to the Red Hat OpenShift Container Platform cluster:

   ```
   cpd-cli manage login-to-ocp \
   --username=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

2. Update the custom resource for watsonx.data.

   The command that you run depends on the storage on your cluster:

   - Red Hat OpenShift Data Foundation Storage

     Run the following command to create the custom resource.

     ```
     cpd-cli manage apply-cr \
     --components=watsonx_data \
     --release=${VERSION} \
     --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
     --block_storage_class=${STG_CLASS_BLOCK} \
     --file_storage_class=${STG_CLASS_FILE} \
     --license_acceptance=true \
     --upgrade=true
     ```

   - IBM Storage Fusion Data Foundation storage

     Run the following command to create the custom resource.

     ```
     cpd-cli manage apply-cr \
     --components=watsonx_data \
     --release=${VERSION} \
     --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
     --block_storage_class=${STG_CLASS_BLOCK} \
     --file_storage_class=${STG_CLASS_FILE} \
     --license_acceptance=true \
     --upgrade=true
     ```

- IBM Storage Fusion Global Data Platform storage

  Remember: When you use IBM Storage Fusion storage, both `${STG_CLASS_BLOCK}` and `${STG_CLASS_FILE}` point to the same storage class, typically `ibm-spectrum-scale-sc` or `ibm-storage-fusion-cp-sc`.

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --block_storage_class=${STG_CLASS_BLOCK} \
  --file_storage_class=${STG_CLASS_FILE} \
  --license_acceptance=true \
  --upgrade=true
  ```

- IBM Storage Scale Container Native storage

  Remember: When you use IBM Storage Scale Container Native storage, both `${STG_CLASS_BLOCK}` and `${STG_CLASS_FILE}` point to the same storage class, typically `ibm-spectrum-scale-sc`.

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --block_storage_class=${STG_CLASS_BLOCK} \
  --file_storage_class=${STG_CLASS_FILE} \
  --license_acceptance=true \
  --upgrade=true
  ```

- Portworx storage

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --storage_vendor=portworx \
  --license_acceptance=true \
  --upgrade=true
  ```

- NFS storage

  Remember: When you use NFS storage, both `${STG_CLASS_BLOCK}` and `${STG_CLASS_FILE}` point to the same storage class, typically `managed-nfs-storage`.

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --block_storage_class=${STG_CLASS_BLOCK} \
  --file_storage_class=${STG_CLASS_FILE} \
  --license_acceptance=true \
  --upgrade=true
  ```

- AWS with EFS and EBS storage

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --block_storage_class=${STG_CLASS_BLOCK} \
  --file_storage_class=${STG_CLASS_FILE} \
  ```

```
--license_acceptance=true \
--upgrade=true
```

- NetApp Trident

  Remember: When you use NetApp Trident storage, both ${STG_CLASS_BLOCK} and ${STG_CLASS_FILE} point to the same storage class.

  Run the following command to create the custom resource.

  ```
  cpd-cli manage apply-cr \
  --components=watsonx_data \
  --release=${VERSION} \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
  --block_storage_class=${STG_CLASS_BLOCK} \
  --file_storage_class=${STG_CLASS_FILE} \
  --license_acceptance=true \
  --upgrade=true
  ```

3. Validate the upgrade.

   watsonx.data is upgraded when the `apply-cr` command returns:

   ```
   [SUCCESS]... The apply-cr command ran successfully
   ```

   If you want to confirm that the custom resource status is `Completed`, you can run the `cpd-cli manage get-cr-status` command:

   ```
   cpd-cli manage get-cr-status \
   --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
   --components=watsonx_data
   ```

Administering watsonx.data

IBM watsonx.data administration includes post-installation setup tasks, ongoing maintenance tasks, managing users, and backing up and restoring watsonx.data.

watsonx.data on Red Hat OpenShift

Post-installation setup (Day 1 operations)

After you install IBM watsonx.data, complete the appropriate tasks to secure your environment and ensure that watsonx.data can run smoothly.

watsonx.data on Red Hat OpenShift

Securing your watsonx.data environment

After you install IBM watsonx.data, complete set-up tasks to ensure that your environment is secure.

watsonx.data on Red Hat OpenShift

Changing the password for the default admin user

When you install IBM watsonx.data, a default administrator is created.

watsonx.data on Red Hat OpenShift

What permissions do you need to complete this task?

To complete this task, you must have one of the following roles:

- Red Hat OpenShift Container Platform cluster administrator
- Red Hat OpenShift Container Platform project administrator on the project where watsonx.data is installed

When do you need to complete this task?

You should change the password for the `admin` user before you complete any tasks in the watsonx.data web client.

Before you begin

The workstation from which you will run the commands must be set up as a client workstation with the following command-line interfaces:

- watsonx.data CLI: `cpd-cli`
- OpenShift CLI: `oc`

For details, see: Setting up a client workstation.

Procedure

1. Run the `cpd-cli manage login-to-ocp` command to log in to the cluster as a user with sufficient permissions to complete this task. For example:

   ```
   cpd-cli manage login-to-ocp \
   --username=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

   Tip: The `login-to-ocp` command takes the same input as the `oc login` command. Run `oc login --help` for details.
2. Run the following command to get the URL and the default password of the `admin` user:

   ```
   cpd-cli manage get-cpd-instance-details \
   --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
   --get_admin_initial_credentials=true
   ```

3. Enter the URL in a web browser and log in to the web client as the `admin` user with the default password.
4. Change the password:
   a) Click your avatar in the toolbar.
   b) Click **Profile and settings**.
   c) Open the **Password** tab.
   d) Enter the default password in the **Current password** field.
   e) Enter and confirm the new password.
   f) Click **Save**.

Customizing and securing the route to the platform

watsonx.data on Red Hat OpenShift

Using a custom TLS certificate for HTTPS connections to the platform
IBM watsonx.data includes a self-signed TLS certificate that is used to enable HTTPS connections. However, this certificate is untrusted by all HTTPS clients. It is recommended that you replace the self-signed certificate with your own certificate.

watsonx.data on Red Hat OpenShift

About this task

What permissions do you need to complete this task?
To complete this task, you must have one of the following roles:

- Cluster administrator
- Instance administrator

When should you complete this task?

This task is optional, but recommended. Complete this task before you give users access to the watsonx.data.

Before you begin

The workstation from which you run the commands must be set up as a client workstation with the following command-line interfaces:

- watsonx.data CLI: `cpd-cli`
- OpenShift CLI: `oc`

For more details, see Setting up a client workstation to install watsonx.data

The files that you must provide depend on the type of route that you want to use for watsonx.data:

Route type	Default certificate	Custom TLS certificate and key	CA certificate
Passthrough	By default, the route uses an IBM self-signed certificate. However, it is recommended that you replace this certificate with a custom certificate.	Supported. If you use a passthrough route, this option is recommended. The files must be in an unencrypted PEM format.	Not used.
Re-encrypt (default)	By default, the route uses the default certificate that is provided by the Red Hat OpenShift Container Platform ingress controller. However, it is recommended that you replace this certificate with a custom certificate. For details, see *Replacing the default ingress certificate*the Red Hat OpenShift Container Platform documentation.	Uses the ingress controller settings by default. If you want to use custom certificates, you must specify: • TLS certificate • TLS key • CA certificate The files must be in an unencrypted PEM format.	Uses the ingress controller settings by default. If you want to use custom certificates, you must specify: • TLS certificate • TLS key • CA certificate The files must be in an unencrypted PEM format.

Tip: The `login-to-ocp` command takes the same input as the `oc login` command. Run `oc login --help`for details.

To provide the files to the `cpd-cli`, you can create a secret that you provide to the `cpd-cli`

- Command to create a secret with a TLS certificate and TLS key.
- Command to create a secret with a CA certificate, TLS certificate, and TLS key.

Procedure

1. Run the `cpd-cli manage login-to-ocp` command to log in to the cluster as a user with sufficient permissions to complete this task. For example

```
cpd-cli manage login-to-ocp \
--username=${OCP_USERNAME} \
--password=${OCP_PASSWORD} \
--server=${OCP_URL}
```

Tip: The `login-to-ocp` command takes the same input as the `oc login` command. Run `oc login --help` for details.

2. Run the `cpd-cli manage setup-route` command.

   The command that you run depends on whether you are using:

   - Passthrough route
   - Reencrypt route

   Tip: If you are unsure whether the route is a passthrough route or a reencrypt route, run:

   ```
   oc get route cpd -n=${PROJECT_CPD_INST_OPERANDS} -o "jsonpath={.spec.tls.termination}"
   ```

   The command returns either `passthrough` or `reencrypt`.

Modifying the route to the platform
The default watsonx.data is a `reencrypt` route that is named `cpd`.

watsonx.data on Red Hat OpenShift

About this task

You can modify the route by:

- Changing the hostname of the default route.
- Changing the default route from a `reencrypt` route to a `passthrough` route.

What permissions do you need to complete this task?
To complete this task, you must have one of the following roles:

- Cluster administrator
- Instance administrator

When do you need to complete this task?
This task is optional, but recommended. Complete this task before you give users access to the watsonx.data platform.

If you have multiple installations of watsonx.data on your OCP cluster, complete this task for each installations.

Before you begin

The workstation from which you run the commands must be set up as a client workstation with the following command-line interfaces:

- watsonx.data CLI: `cpd-cli`
- Red Hat OpenShift CLI: `oc`

Procedure

1. **About this task**

   This task covers how to change the hostname of the route. If you want to change the route from a passthrough route to a reencrypt route, see Using a custom TLS certificate for HTTPS connections to the platform

   Procedure

   Run the `cpd-cli manage login-to-ocp` command to log in to the cluster as a user with sufficient permissions to complete this task. For example,:

   ```
   cpd-cli manage login-to-ocp \
   --username=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

Tip: The `login-to-ocp` command takes the same input as the `oc login` command. Run `oc login --help` for details.

2. Run the `cpd-cli manage setup-route` command.

   The command that you run depends on whether you want to change the hostname of the default route or whether you want to change the route type.

   a) Changing the hostname of the route.

      Replace the following variables before you run the command:

      <hostname>
         The hostname to use in the route.

      ```
      cpd-cli manage setup-route \
      --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
      --custom_hostname=<hostname>
      ```

   b) Changing the type of the route

      ```
      cpd-cli manage setup-route \
      --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
      --route_type=passthrough
      ```

Turning off the host header injection check

IBM watsonx.data rejects requests that contain invalid external routes to prevent host header injection attacks. If you use a load balancer and reverse proxy servers to manage, host headers before requests are forwarded to watsonx.data, turn off the 'host header injection check' to enable watsonx.data to accept the incoming requests.

watsonx.data on Red Hat OpenShift

About this task

Who needs to complete this task?
To complete this task, you must be either:

- A cluster administrator
- An administrator of the project where watsonx.data is installed.

When do you need to complete this task?

- Complete this task only if your cluster uses a load balancer and reverse proxy servers to manage host headers.

Procedure

1. Log in to Red Hat OpenShift Container Platform as a user with sufficient permissions to complete the task:

   ```
   oc login ${OCP_URL}
   ```

2. Run the following command to turn off the host header injection check:

   ```
   oc patch configmap product-configmap \
   --namespace ${PROJECT_CPD_INST_OPERANDS} \
   --type=merge \
   --patch '{"data": {"HOST_INJECTION_CHECK_ENABLED":"false"}}'
   ```

3. Restart the `ibm-nginix` deployments:

   ```
   oc rollout restart deployment/ibm-nginx \
   --namespace ${PROJECT_CPD_INST_OPERANDS}
   ```

Results

The host header injection check is turned off.

Tip: If you need to turn on the host header injection check, you can rerun the preceding commands with `"HOST_INJECTION_CHECK_ENABLED":"true"`.

Integrating with the IAM Service

By default, IBM watsonx.data user records are stored in an internal repository database. However, it is recommended that you use an enterprise-grade password management solution, such as single sign-on (SSO) or LDAP.

watsonx.data on Red Hat OpenShift

Before you begin

Best practice: If you set up environment variables for your installation, you can run many of the commands in this task exactly as written. For the instructions, see Setting up installation environment variables.

Ensure that you source the environment variables before you run the commands in this task.

If you use LDAP, you can choose between the following options.

Mechanism	Benefits	Drawbacks
LDAP integration provided by watsonx.data.	You can use LDAP with or without SAML SSO. You can choose the level of integration with the LDAP server. You can use LDAP to: • Validate users' credentials • Manage access to the platform.	You can connect to a single LDAP server from each instance of watsonx.data The LDAP configuration cannot be shared across watsonx.data instances or used by any other IBM watsonx.data on the cluster.
LDAP integration that is provided by the Identity and Access Management Service (IAM Service) in IBM Cloud Pak foundational services.	You can connect to multiple LDAP servers, and the multiple instances of watsonx.data can use the connections.	If you have multiple LDAP servers that must be isolated from each other, do not use this method. For example, you maintain two instances of watsonx.data for different groups of users. Each group of users is managed by a different LDAP server, and you don't want the users to see information about users in the other LDAP server.

To use the LDAP integration provided by watsonx.data, see "Connecting to your identity provider" on page 174.

Permissions that you need for this task:

You must be either a cluster administrator or an administrator of the following projects:

- The project where IBM watsonx.data foundational services is installed (`ibm-common-services`).
- The project where watsonx.data is installed.

When you need to complete this task?

If you want to use the LDAP integration provided by the IAM Service, you must integrate watsonx.data with the IAM Service before you onboard users or create user groups. When you integrate with the IAM

Service, you delegate all authentication to the IAM Service. If you onboard users before you integrate with the IAM Service, existing users might not be able to log in to watsonx.data.

About this task

Important:

Integrating with the IAM Service is irreversible.

Contact **IBM Support** to reset watsonx.data to the previous state.

Procedure

1. Run the `cpd-cli manage login-to-ocp` command to log in to the cluster as a user with sufficient permissions to complete this task. For example:

   ```
   cpd-cli manage login-to-ocp \
   --username=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

   Tip: The `login-to-ocp` command takes the same input as the `oc login` command. Run `oc login --help` for details.
2. Run the following command to integrate with the IAM Service.

   ```
   cpd-cli manage setup-iam-integration \
   --enable=true \
   --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS}
   ```

   The command triggers a reconciliation in the Zen operator.

   If the IAM Service needs to be started and configured, it might take up to 20 minutes for the process to complete.
3. Wait for the `setup-iam-integration` command to return Succeeded.
4. To confirm that the integration is complete, confirm that the following command returns `true`.

   ```
   oc get zenservice lite-cr \
   -n ${PROJECT_CPD_INST_OPERANDS} \
   -o jsonpath='{.spec.iamIntegration}'
   ```
5. Get the initial password for the `admin` user from the IAM Service.

   ```
   oc get secret platform-auth-idp-credentials \
   -n ${} \
   -o jsonpath='{.data.admin_password}' | base64 --decode;
   ```

Configuring single sign-on

You can use Security Assertion Markup Language (SAML) for single sign-on (SSO) to the IBM watsonx.data web client.

watsonx.data on Red Hat OpenShift

Before you begin

Best practice: If you set up environment variables for your installation, you can run many of the commands in this task exactly as written. For the instructions, see Setting up installation environment variables.

Ensure that you source the environment variables before you run the commands in this task.

You must have an existing SAML identity provider (IdP). Work with your IdP administrator to gather the following information.

Parameter	Description	Value
`entryPoint`	The URL of the login page for your identity provider.	
`fieldToAuthenticate`	The name of the parameter you use to authenticate with the identity provider, such as `emailAddress` or `username`. If you plan to use LDAP and SAML, use the same attribute to identify users. This parameter must have the same value as the **User search field** in your LDAP configuration.	
`spCert`	The private key used to sign SAML requests to the identity provider. The certificate corresponding to this key needs to be set when you register watsonx.data with your identity provider so that your identity provider can verify the SAML requests. If you do not specify a certificate, the requests cannot be signed.	Remove the **`BEGIN PRIVATE KEY`** and **`END PRIVATE KEY`** lines and provide the private key as a single line.
`idpCert`	The certificate provided by the identity provider to verify SAML responses from the identity provider.	Remove the **`BEGIN CERTIFICATE`** and **`END CERTIFICATE`** lines and provide the certificate as a single line.
`issuer`	The name that you want to use to register watsonx.data with your identity provider. If you do not specify a value, the default (`ibm_privatecloud`) is used.	
`identifierFormat`	The format of requests from watsonx.data to the identity provider. Your identity provider must support the format. If you do not specify a format, the default format (`urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`) is used.	

Parameter	Description	Value
`callbackUrl`	An approved URL (that you set with the SAML identity provider) to redirect users to after they successfully authenticate by using SSO. For example, to redirect successfully authenticated users to the watsonx.data landing page, you can specify `https://cluster/auth/login/sso/callback`.	
`disableRequestedAuthnContext`	A Boolean parameter for AD FS client authentication. If set the true, the authentication context is disabled so that the IDP determines the method of authentication. If you do not specify a value, then the default is false.	

About this task

To configure SSO, you must specify information about your Identity Provider in a configuration file. Use the preceding table to get necessary information that you need to supply in the configuration file.

Important: It is recommended that you complete this task before you add users to watsonx.data. If you already added users to watsonx.data, you must add the users again with their SAML ID to enable them to use SSO.

Procedure

1. Log in to your Red Hat OpenShift cluster as a project administrator.

   ```
   oc login ${OCP_URL}
   ```

2. Enable SAML by running the following command.

   ```
   oc exec -it -n ${PROJECT_CPD_INST_OPERANDS} \
   $(oc get pod -n ${PROJECT_CPD_INST_OPERANDS} -l component=usermgmt | tail -1 | cut -f1 -d\ )
   \
   -- bash -c "vi /user-home/_global_/config/saml/samlConfig.json"
   ```

3. In the `samlConfig.json` file, specify the appropriate values for your environment.

   ```
   {
     "entryPoint": "",
     "fieldToAuthenticate": "",
     "spCert": "",
     "idpCert": "",
     "issuer": "",
     "identifierFormat": "",
     "callbackUrl": ""
   }
   ```

4. Save your changes to `samlConfig.json`.
   a) Press the Escape key.
   b) Press the Colon (:) key.
   c) Enter `:x`.
   d) Press the Enter key.
5. Run the following command to restart the `usermgmt` pods:

```
oc delete pods -n ${PROJECT_CPD_INST_OPERANDS} -l component=usermgmt
```

What to do next

Wait few minutes for the `usermgmt` pods to restart before you attempt to log in to the web client. If the pods are not running, you cannot log in.

If you previously added users to watsonx.data, you must readd the users with their SAML ID to enable them to use SSO. To add users, complete the following steps.

1. Go directly to the web client log in page by appending the following path to your watsonx.dataURL: `/auth/login/zen-login.html`.
2. Log in to the web client as an administrator with user management permissions.
3. Add users with their SAML IDs. For more information, see "Managing users " on page 173.

Disabling SAML

Procedure

1. Disable SAML by running the following command.

   ```
   oc exec -it -n ${PROJECT_CPD_INST_OPERANDS} \
   $(oc get pod -n ${PROJECT_CPD_INST_OPERANDS} -l component=usermgmt | tail -1 | cut -f1 -d\ )
   \
   -- bash -c "rm /user-home/_global_/config/saml/samlConfig.json"
   ```

2. Run the following command to restart the `usermgmt` pods.

   ```
   oc delete pods -n ${PROJECT_CPD_INST_OPERANDS} -l component=usermgmt
   ```

Setting up security for connections

If you use connections, such as platform connections, you should review the following information to determine whether there are any additional tasks that you must complete.

watsonx.data on Red Hat OpenShift

Managing certificates

Set up custom TLS certificates for connecting to the platform and CA certificates to connect to internal servers from the platform.

watsonx.data on Red Hat OpenShift

Using a custom TLS certificate for HTTPS connections to the platform

IBM watsonx.data includes a self-signed TLS certificate that is used to enable HTTPS connections. However, this certificate is untrusted by all HTTPS clients. It is recommended that you replace the self-signed certificate with your own certificate.

watsonx.data on Red Hat OpenShift

About this task

What permissions do you need to complete this task?

To complete this task, you must have one of the following roles:

- Cluster administrator
- Instance administrator

When should you complete this task?

This task is optional, but recommended. Complete this task before you give users access to the watsonx.data.

Before you begin

The workstation from which you run the commands must be set up as a client workstation with the following command-line interfaces:

- watsonx.data CLI: `cpd-cli`
- OpenShift CLI: `oc`

For more details, see Setting up a client workstation to install watsonx.data

The files that you must provide depend on the type of route that you want to use for watsonx.data:

Route type	Default certificate	Custom TLS certificate and key	CA certificate
Passthrough	By default, the route uses an IBM self-signed certificate. However, it is recommended that you replace this certificate with a custom certificate.	Supported. If you use a passthrough route, this option is recommended. The files must be in an unencrypted PEM format.	Not used.
Re-encrypt (default)	By default, the route uses the default certificate that is provided by the Red Hat OpenShift Container Platform ingress controller. However, it is recommended that you replace this certificate with a custom certificate. For details, see *Replacing the default ingress certificate*the Red Hat OpenShift Container Platform documentation.	Uses the ingress controller settings by default. If you want to use custom certificates, you must specify: • TLS certificate • TLS key • CA certificate The files must be in an unencrypted PEM format.	Uses the ingress controller settings by default. If you want to use custom certificates, you must specify: • TLS certificate • TLS key • CA certificate The files must be in an unencrypted PEM format.

Tip: The `login-to-ocp` command takes the same input as the `oc login` command. Run `oc login --help`for details.

To provide the files to the `cpd-cli`, you can create a secret that you provide to the `cpd-cli`

- Command to create a secret with a TLS certificate and TLS key.
- Command to create a secret with a CA certificate, TLS certificate, and TLS key.

Procedure

1. Run the `cpd-cli manage login-to-ocp` command to log in to the cluster as a user with sufficient permissions to complete this task. For example

```
cpd-cli manage login-to-ocp \
--username=${OCP_USERNAME} \
--password=${OCP_PASSWORD} \
--server=${OCP_URL}
```

Tip: The `login-to-ocp` command takes the same input as the `oc login` command. Run `oc login --help` for details.

2. Run the `cpd-cli manage setup-route` command.

   The command that you run depends on whether you are using:

   - Passthrough route
   - Reencrypt route

   Tip: If you are unsure whether the route is a passthrough route or a reencrypt route, run:

   ```
   oc get route cpd -n=${PROJECT_CPD_INST_OPERANDS} -o "jsonpath={.spec.tls.termination}"
   ```

   The command returns either `passthrough` or `reencrypt`.

Setting the idle session timeout

You can adjust the idle session timeout for IBM watsonx.data in accordance with your security and compliance requirements. If a user leaves their session idle in a web browser for the specified length of time, the user is automatically logged out of the web client.

watsonx.data on Red Hat OpenShift

Before you begin

Required permissions

To complete this task, you must have one of the following roles:

- Red Hat OpenShift cluster administrator
- Red Hat OpenShift project administrator on the project where watsonx.data is installed.

About this task

By default, watsonx.data logs users out after 12 hours. You can edit the watsonx.data `product-configmap` to adjust:

The length of time until a user's session expires (`TOKEN_EXPIRY_TIME`).

The default is 12 hours.

If you set `TOKEN_EXPIRY_TIME: "1"`, a user's session will expire in after 1 hour of inactivity. If you set `TOKEN_EXPIRY_TIME: "0.5"`, a user's session will expire after 30 minutes of inactivity. When the user leaves their session idle for the specified length of time, the user is automatically logged out of the web client.

It is recommended that you set the value between `0.1` and `1`.

The length of time that a user has to refresh their session (`TOKEN_REFRESH_PERIOD`).

The default is 12 hours.

If you set `TOKEN_REFRESH_PERIOD: "1"` and the user's session does not expire, the user's session is automatically refreshed during this 60-minute period. The session is extended based on the value that is set for the `TOKEN_EXPIRY_TIME` parameter. However, after the token refresh period passes, the user must log back into the web client when their current session expires.

It is recommended that you set the value between 1 and 24.

If you don't want to allow users to extend their sessions, set the value of the `TOKEN_REFRESH_PERIOD` parameter to a value less than the value of the `TOKEN_EXPIRY_TIME` parameter.

For example, as an administrator, you configure:

```
TOKEN_EXPIRY_TIME: "0.5"
TOKEN_REFRESH_PERIOD: "2"
```

If a user starts work at 8 AM and logs in to the web client, the user must be active in the web session within 30 minutes for their token to be refreshed:

- If the user stop using the web client at 8:10 and attempts to use the web client again until 8:41, the user must reauthenticate to the web client because their session expired.
- If the user remains active in their session and their token refreshes at 9:59 AM, their session lasts until 10:29 AM. However, when the session expires at 10:29, the user must reauthenticate to the web client because the token refresh period expired.

Procedure

1. Log in to your OpenShift cluster:

   ```
   oc login OpenShift_URL:port
   ```

2. Change to the project where watsonx.data is deployed:

   ```
   oc project ${PROJECT_CPD_INST_OPERANDS}
   ```

3. Run the following command to edit the watsonx.data product-configmap:

   ```
   oc edit configmap product-configmap
   ```

4. Add an entry for the `TOKEN_EXPIRY_TIME` parameter to the `data` section of the `product-configmap` file. For example:

   ```
   data:
     ...
     TOKEN_EXPIRY_TIME: "1"
     ...
   ```

5. Add an entry for the `TOKEN_REFRESH_PERIOD` parameter to the `data` section of the `product-configmap` file. For example:

   ```
   data:
     ...
     TOKEN_REFRESH_PERIOD: "1"
     ...
   ```

6. Save your changes to the product-configmap file. For example, if you are using `vi`, hit `esc` and enter:

   ```
   :wq
   ```

7. Restart the `usermgmt` pods for the changes to take effect. To restart the pods, run the following command:

   ```
   oc delete pod -l component=usermgmt
   ```

Limiting the user information that is returned by usermgmt API calls

By default, a user without the **Administer platform** or **Manage users** permission can see detailed information about other IBM watsonx.data users by running `usermgmt` API calls. You can set the `usermgmt_limit_user_info` parameter to limit the information that is returned by the API calls.

watsonx.data on Red Hat OpenShift

Before you begin

Who needs to complete this task?

To complete this task, you must be either:

- A cluster administrator
- An administrator of the project where watsonx.data is installed.

When do you need to complete this task?

Complete this task before you give users access to watsonx.data.

The `usermgmt_limit_user_info` is set to `true` by default.

About this task

By default, when a user without the **Administer platform** or **Manage users** permission runs `usermgmt` API calls, they can see detailed user records. For example:

```
[
   {
       "uid": "1000331009",
       "username": "user@email.com",
       "displayName": "First Last",
       "email": "user@email.com",
       "approval_status": "approved",
       "permissions": [
           "sign_in_only",
           "administrator",
           "can_provision"
       ],
       "user_roles": [
           "zen_user_role",
           "zen_administrator_role"
       ],
       "current_account_status": "enabled",
       "internal_user": false,
       "deletable": true,
       "authenticator": "external",
       "created_timestamp": 1663963657537,
       "last_modified_timestamp": 1665614744277,
       "misc": {
           "dark_mode": false
       },
       "role": "Admin",
       "groups": [
           {
               "group_id": 10000,
               "name": "All users",
               "description": "All users are implicitly part of this group",
               "created_by": "",
               "created_at": "",
               "updated_at": "",
               "misc": {},
               "members_count": "5"
           }
       ],
       "group_roles": []
   },
   .
   .
   .
]
```

However, if you set the `usermgmt_limit_user_info` parameter to `true`, the API calls return a subset of the properties in the user record. For example:

```
[
   {
       "uid": "1000331009",
       "username": "user@email.com",
       "displayName": "First Last",
       "email": "user@email.com",
       "permissions": [
           "User"
       ],
       "role": "User",
       "user_roles": [
           "sign_in_only"
       ],
       "groups": [],
       "group_roles": []
   },
   .
   .
   .
]
```

The limited record includes only the parameters that are necessary for other features, such as adding a user to a project and deployment space.

Procedure

1. Log in to Red Hat OpenShift Container Platform as a user with sufficient permissions to complete the task:

   ```
   oc login ${OCP_URL}
   ```

2. Run the following command to limit the information that is returned by `usermgmt` API calls:

   ```
   oc patch configmap product-configmap \
   --namespace ${PROJECT_CPD_INST_OPERANDS} \
   --type=merge \
   --patch '{"data": {"usermgmt_limit_user_info":"true"}}'
   ```

3. Restart the `usermgmt` pods:

   ```
   oc delete pod -n=${PROJECT_CPD_INST_OPERANDS} -l component=usermgmt
   ```

Results

The information that is returned by `usermgmt` API calls is limited.

Tip: If you need to return more detailed user records, you can rerun the preceding commands with `"usermgmt_limit_user_info":"false"`.

Displaying a term and conditions prompt

If you need users to accept terms and conditions before they use the web client, you can enable a dialog that prompts users to accept the terms and conditions before they can log in to the web client. For example, you might need to enable the prompt to comply with the Federal Information Security Management Act (FISMA) regulations.

watsonx.data on Red Hat OpenShift

Before you begin

Best practice: You can run many of the commands in this task exactly as written if you set up environment variables for your installation. For instructions, see Setting up installation environment variables.

Ensure that you source the environment variables before you run the commands in this task.

About this task

When you configure the web client to display a terms and conditions prompt, you must specify the following information:

- The header text for the dialog
- The terms and conditions that the user must accept.
- The prompt that the user must click to acknowledge that they accept the terms and conditions.

Procedure

1. Log in to your Red Hat OpenShift cluster as a project administrator:

```
oc login ${OCP_URL}
```

2. Create the login-dialog.json configuration file:

Property	Description
enabled	Set **enabled** to true to enable the dialog. For example: `"enabled": true,` Valid values: true or false
headerText	Specify the text to display as the header in the dialog. For example: `"headerText": "Terms of use",` Use standard JSON string format.
dialogText	Specify the terms and conditions that the user must agree to before they can access the web client. For example: `"dialogText": "The terms and conditions of use that your user must accept.",` Use standard JSON string format.
acceptText	Specify the text that the user must click to acknowledge that they agree to the terms and conditions. For example: `"acceptText": "I understand and accept the terms",` Use standard JSON string format.

3. Copy the login-dialog.json file into the config directory:

```
oc cp login-dialog.json $(oc get pod -n ${PROJECT_CPD_INST_OPERANDS} -l component=usermgmt |
tail -1 | cut -f1 -d\ ):/user-home/_global_/config/
```

Setting up watsonx.data for end users

watsonx.data on Red Hat OpenShift

Enabling email notifications

You can configure a connection to your SMTP server so that IBM watsonx.data can send email to users.

watsonx.data on Red Hat OpenShift

Before you begin

To complete this task, you must have one of the following permissions:

- **Administer platform**
- **Manage configurations**

In addition, you must have an email address associated with your user ID. You can check whether there is an email address associated with your account from your profile. If you do not have an email address associated with your user ID, see "Managing access to the platform" on page 185.

About this task

watsonx.data sends two types of emails:

Alerting emails
You can have watsonx.data send notifications to specific users when the monitoring and alerting framework issues an alert.

For more information about alerts, see "Monitoring and alerting in watsonx.data" on page 158.

Notification emails
If the IBM Cloud Pak for Data common core services are installed, users can choose whether they receive emails about the projects or deployment spaces that they are members of.

Procedure

To enable watsonx.data to send email:

1. Log in to the web client as an administrator.
2. From the menu, select **Administration** > **Configurations**.
3. On the **SMTP settings** page, specify the following information:
   - Your SMTP mail server address.
   - The port number of your SMTP server.

     Important: If you specify a secure port, you must select **Use TLS connection** so that watsonx.data can communicate with your SMTP server.
   - Specify the appropriate SMTP credentials for your environment:

Method of sending communications	SMTP server requires authentication	SMTP server does not require authention
My SMTP server uses a mailer daemon to send communications	You must specify the following fields: – **SMTP username** – **SMTP password**	You don't need to specify any fields. However, if you want to override the mailer daemon, you can specify a **From account**.

Method of sending communications	SMTP server requires authentication	SMTP server does not require authentication
My SMTP server uses a default account to send communications	You must specify the following fields: – **SMTP username** – **SMTP password** – **From account**	You must specify the following fields: – **From account**

4. Click **Save**. If your SMTP configuration is successful, you will receive a confirmation email.
   - If you specified a **From account** when you configured the connection to your SMTP server, the confirmation email is sent to the account specified in the **From account** field.
   - If you did not specify a **From account** when you configured a connection to your SMTP server, the confirmation email is sent to the account specified in the **SMTP username** field.

Results

Depending on your configuration, notification emails are sent from one of the following accounts:

- If you specified a **From account** when you configured the connection to your SMTP server, notifications are sent from the account specified in the **From account** field.
- If you did not specify a **From account** when you configured a connection to your SMTP server, notifications are sent from the mailer daemon.

Enabling users to access the web client from platform-generated emails

IBM watsonx.data can generate notifications. For example, collaborators in an analytics project get a notification when assets or new collaborators are added to the project. If you configure a connection to your SMTP server, users can receive these notifications through email. To ensure that these emails include active links to the web client, you must add the URL_PREFIX for your deployment to the watsonx.data `product-configmap`.

watsonx.data on Red Hat OpenShift

About this task

A Red Hat OpenShift project (namespace) administrator can edit the watsonx.data `product-configmap` to specify the URL_PREFIX for your deployment.

The URL_PREFIX is the domain name at the beginning of your deployment URL. For example, if your deployment of watsonx.data is accessible from `https://domain.my.company.com/zen`, your domain name is `domain.my.company.com`. Do not include the protocol in the value that you specify.

If you use the default port, 443, you do not need to specify the port number in the value for the URL_PREFIX parameter. However, if you use a non-standard port, include it in the URL_PREFIX. For example, if you use port 31843, your entry would be:

```
URL_PREFIX: domain.my.company.com:31843
```

Procedure

To enable users to access the web client from platform-generated emails:

1. Log in to your Red Hat OpenShift cluster as a project administrator:

   ```
   oc login OpenShift_URL:port
   ```

2. Change to the project where you installed watsonx.data:

   ```
   oc project Project
   ```

3. Run the following command to edit the watsonx.data `product-configmap`:

   ```
   oc edit cm product-configmap
   ```

4. Add an entry for the `URL_PREFIX` parameter to the `data` section of the `product-configmap` file. For example:

   ```
   data:
     ...
     SHOW_USER_APPROVAL: "false"
     URL_PREFIX: domain.my.company.com
     ...
   ```

5. Save your changes to the `product-configmap` file.

   For example, if you are using `vi`, hit `esc` and enter:

   ```
   :wq
   ```

   The changes are automatically applied to the platform.

Customizing the branding of the web client

As an IBM watsonx.data administrator, you can customize the web client with your company's branding. You can customize the name that is displayed in the web client and use a custom logo on your home page.

watsonx.data on Red Hat OpenShift

Before you begin

Required permissions

To customize the branding of the web client, you must have one of the following permissions:

- **Administer platform**
- **Manage configurations**

About this task

Product name

If you change the product name in the web client, the custom name is used throughout the web client. For example, the custom name is used on the home page and on the banner of every page in the watsonx.data web client.

Home page logo

If you upload a logo, the logo is displayed in the welcome section of the home page.

Browser icon

If you upload an icon, the icon is displayed in web browser tabs.

Home page image

If you upload a home page image, the image is displayed on the background of the home page.

Login image

If you upload a login image, the image is displayed on the background of the login page.

Procedure

1. Log in to watsonx.data.
2. From the navigation menu, select **Administration** > **Customizations**.
3. Click **Branding**.
4. To use a custom name, select **Custom name** and enter the name that you want to use in the web client.
   - The name must be 40 characters or less.
   - The product name cannot include the following characters: `<, >, /, or \`.

5. To use a custom logo, select **Use your own logo** under **Homepage logo** and drop a file or browse for a file to upload as the home page logo.
    - You can supply a JPG, PNG or SVG file.
    - The image must be 512 x 512 pixels.
    - The image must be 500 KB or less.
6. To use a custom browser icon, select **Use your own icon** under **Browser icon** and drop a file or browse for a file to upload as the icon in browser tabs.
    - You can supply a JPG, PNG, or ICO file.
    - The image must be 512 x 512 pixels.
    - The image must be 500 KB or less.
7. To use a custom image on the home page, select **Use your own image** under **Homepage image** and drop a file or browse for a file to upload as the background image on the home page.
    - You can supply a JPG, PNG, or SVG file.
    - For best results, use an image that is 500 KB or less. However, the image can be up to 20 MB.
8. To use a custom image on the login page, select **Use your own image** under **Login image** and drop a file or browse for a file to upload as the background image on the homepage.
    - You can supply a JPG, PNG, or SVG file.
    - For best results, use an image that is 500 KB or less. However, the image can be up to 20 MB.
9. Click **Apply**.

What to do next

If you want to remove your customizations, click **Restore defaults**.

Forwarding alerts from watsonx.data

watsonx.data on Red Hat OpenShift

Forwarding alerts to email accounts

You can forward alerts that are generated by the platform to specific email accounts.

watsonx.data on Red Hat OpenShift

Who needs to complete this task?
To configure alert forwarding, you must have one of the following permissions:

- **Administer platform**
- **Manage configurations**

When do you need to complete this task?
Complete this task if you want to forward alerts that were issued by IBM watsonx.data to specific email accounts.

Before you begin

You must "Enabling email notifications" on page 139 before you can forward alerts to email accounts.

About this task

You can send alerts to:

Platform users with the Manage platform health permission
Platform users receive alerts at the email address that is associated with their username. The platform users can click the link in the email to access more information about the alert.

Other email recipients

You can optionally forward alerts to users who don't have access to the platform. The alerts can keep these users informed about the health of platform.

Procedure

To configure alert forwarding to email accounts:

1. Log in to the web client.
2. From the navigation menu, select **Administration** > **Configurations**.
3. Open the **Alert forwarding** tab.
4. On the **Email** tab, click **Add recipients**.
5. Specify the users that you want to add:
   - On the **Users** page, select platform users to send alerts to.
   - On the **Other email recipients** page, enter an email address and click **Add recipient to list**. Repeat this step for each email address that you want to forward alerts to.
6. Click **Save**.

Results

The platform sends alerts to the specified email accounts.

Forwarding alerts to a Simple Network Management Protocol (SNMP) server

You can forward alerts that are generated by the platform to an SNMP server.

watsonx.data on Red Hat OpenShift

Who needs to complete this task?

To configure alert forwarding, you must have one of the following permissions:

- **Administer platform**
- **Manage configurations**

When do you need to complete this task?

Complete this task if you want to forward alerts that were issued by IBM watsonx.data to an SNMP server.

Before you begin

You must configure an SNMP server with a trap listener. For more information, see Installing Net-SNMP.

About this task

SNMP (simple network management protocol) is a standard protocol for collecting and organizing information about managed devices or services. The platform can send alerts as SNMP traps.

Procedure

To configure alert forwarding to email accounts:

1. Log in to the web client.
2. From the navigation menu, select **Administration** > **Configurations**.
3. Open the **Alert forwarding** tab.
4. On the **SNMP** tab enter the following information:
   - In the **SNMP server address** field, specify the hostname or IP address of the SNMP server.
   - In the **SNMP server port** field, specify the port number that the SNMP trap listens on. The default port is 162.

- In the **SNMP trap community string** field, enter the string that the SNMP server uses for SNMP traps. The string enables the platform to send an unrequested message to the SNMP manager to indicate that there is a problem.

5. Click **Save**.

Results

The platform sends alerts to the specified SNMP server.

Forwarding alerts to Slack

You can forward alerts that are generated by the platform to a channel in Slack.

watsonx.data on Red Hat OpenShift

Who needs to complete this task?
To configure alert forwarding, you must have one of the following permissions:

- **Administer platform**
- **Manage configurations**

When do you need to complete this task?
Complete this task if you want to forward alerts that were issued by IBM watsonx.data to a channel in Slack.

Before you begin

You must set up an incoming webhook in the workspace where you want the platform to post alerts.

Procedure

To configure alert forwarding to email accounts:

1. Log in to the web client.
2. From the navigation menu, select **Administration** > **Configurations**.
3. Open the **Alert forwarding** tab.
4. On the **Slack** tab enter the incoming webhook URL.
5. Click **Save**.

Results

The platform sends alerts to the Slack channel that authorized the webhook.

Exposing secure route to Presto server

Secure routes provide the ability to use several types of TLS termination to serve certificates to the client. To expose a secure route to Presto server in IBM watsonx.data, use the `reencrypt` route.

watsonx.data on Red Hat OpenShift

Attention: Use this procedure to expose a secure route to Presto server for watsonx.data v1.0.0 or v1.0.1.

For watsonx.data v1.0.2, routes are automatically created for each Presto engine that is provisioned.

Note: Routes must be exposed only when you need to access the Presto engine from outside the OpenShift cluster. Also, routes must be exposed for every new Presto engine that is provisioned, if a client from outside the OCP cluster needs to connect.

About this task

Complete the following steps to expose a secure route to Presto server in watsonx.data standalone deployment.

Procedure

1. Log in to the OpenShift container.

   Use one of the following method to establish a session to your OpenShift server.

   a. Run the following command to log in to the cluster by providing a username and password:

   ```
   oc login \
   --user=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

   b. Run the following command to log in to the cluster by providing a token:

   ```
   oc login \
   --server=${OCP_URL} \
   --token=${OCP_TOKEN}
   ```

2. Set up the `PROJECT_CPD_INST_OPERANDS` environment variable pointing to the namespace where watsonx.data is installed.

   ```
   export PROJECT_CPD_INST_OPERANDS=<wxd_namespace>
   ```

3. Extract the self-signed TLS certificates.

   ```
   oc extract secret/ibm-lh-tls-secret --keys=tls.crt -n ${PROJECT_CPD_INST_OPERANDS}
   ```

   The self-signed cert is extracted to `tls.crt`.

4. Identify the engine and service name that you want to expose:

   ```
   oc get wxdengine -n ${PROJECT_CPD_INST_OPERANDS} -o custom-columns='CR-
   NAME:metadata.name,ENGINE:spec.engineDisplayName,SERVICE:spec.engineUri' | sed 's/.'$
   {PROJECT_CPD_INST_OPERANDS}'.svc.cluster.local//'
   ```

   Example:

   If you have two engines created for the Presto server:

   ```
   # oc get wxdengine -n ${PROJECT_CPD_INST_OPERANDS} -o custom-columns='CR-
   NAME:metadata.name,ENGINE:spec.engineDisplayName,SERVICE:spec.engineUri' | sed 's/.'$
   {PROJECT_CPD_INST_OPERANDS}'.svc.cluster.local//'
   CR-NAME               ENGINE      SERVICE
   lakehouse-presto-01   presto-01   ibm-lh-lakehouse-presto-01-presto-svc
   lakehouse-presto314   jsizto-01   ibm-lh-lakehouse-presto314-presto-svc
   ```

   a) Set up the `ENGINE_SVC_TO_EXPOSE` environment variable pointing to the `SERVICE` name of the engine route you want to expose.

   ```
   export ENGINE_SVC_TO_EXPOSE=<SERVICE>
   ```

   Example:

   If you want to expose the secure route for engine `presto-01`, then set to this service:

   ```
   # export ENGINE_SVC_TO_EXPOSE=ibm-lh-lakehouse-presto-01-presto-svc
   ```

5. Create a re-encrypt route to expose the engine.

   ```
   oc create route reencrypt \
   --service=${ENGINE_SVC_TO_EXPOSE} \
   --dest-ca-cert=tls.crt \
   --port 8443 -n ${PROJECT_CPD_INST_OPERANDS}
   ```

6. Verify and record the new re-encrypt route.

   ```
   oc get route -n ${PROJECT_CPD_INST_OPERANDS} ${ENGINE_SVC_TO_EXPOSE}
   ```

   The secure route is under the HOST/PORT column.

   Example:

   In this example, the secure route name is:

   `ibm-lh-lakehouse-presto-01-presto-svc-cpd-instance.apps.example.cp.fyre.ibm.com`

   ```
   # oc get route -n ${PROJECT_CPD_INST_OPERANDS} ${ENGINE_SVC_TO_EXPOSE}
   NAME                                   HOST/
   PORT                                                                 PATH
   SERVICES                               PORT  TERMINATION  WILDCARD
   ibm-lh-lakehouse-presto-01-presto-svc  ibm-lh-lakehouse-presto-01-presto-svc-cpd-
   instance.apps.example.cp.fyre.ibm.com        ibm-lh-lakehouse-presto-01-presto-svc  8443
   reencrypt    None
   ```

7. To connect with the exposed Presto server, use the exposed secure route name as the hostname and use port 443 as port number with this route.

What to do next

- To connect with Presto by using ibm-lh-client package, see Using presto-cli and presto-run in the ibm-lh-client package.
- To connect with Presto by using an IDE or utility such as presto-cli, see Importing self-signed certificates from a Presto server to a Java™ truststore.

Ongoing maintenance (Day 2 operations)

Complete ongoing maintenance tasks as necessary to keep IBM watsonx.data running and to make adjustments to your environment.

watsonx.data on Red Hat OpenShift

Best practices for monitoring watsonx.data

It is critical to monitor the health of your IBM watsonx.data environment on a regular basis. By monitoring your watsonx.data deployments and the clusters where your deployments are running, you can ensure that the environment is performant and is scaled to support your business needs. In addition, routine monitoring can help prevent unexpected outages and instability.

watsonx.data on Red Hat OpenShift

Perform routine watsonx.data monitoring

Establish a schedule for monitoring your IBM watsonx.data deployments.

watsonx.data on Red Hat OpenShift

If you have multiple watsonx.data deployments, complete the same process for each deployment.

Who should perform this task?

A watsonx.data administrator with one of the following permissions must perform this task:

- **Administer platform.**
- **Manage platform health.**
- **View platform health.** (read-only access)

How frequently should you perform this task?

It is recommended that you perform this task *at least* once per day or once per shift.

However, if there is a large variation in the number of concurrent users or jobs, it is recommended that you perform this task more frequently during peak activity.

Your routine should include the following tasks:

1. **Important:** Ensure that a cluster administrator completed one of the following tasks:
    - Enable the default monitoring stack on the Red Hat OpenShift Container Platform cluster.
    - Install the Kubernetes Metrics Server on the Red Hat OpenShift Container Platform cluster.

    If neither of these tasks are completed, monitoring data will not be available on the watsonx.data **Monitoring**page.

Metric to check	Things to consider
vCPU and memory use	• Check the amount of vCPU and memory in use compared to the requests and limits. **Services** Review the vCPU and memory use and requests to identify which services consume the most resources. If you click a service, you can see historical resource use for the service. This information can help you determine whether a particular service is causing a spike in resource consumption. **Service instances** Review the vCPU and memory use and requests to identify which service instances consume the most resources. Look for any service instances that are over-sized or unused. For example, the resource use is consistently below the resource requests. If you click a service instance, you can see historical resource use for the service instance. This information can help you determine whether a particular service instance is causing a spike in resource consumption. **Projects** If you click a project, you can see historical resource use for the project. This information can help you determine whether a particular project is causing a spike in resource consumption. **Tool runtimes** Look for any runtimes that is over-sized or unused. For example, the resource use is consistently below the resource requests. **Pods** Review the vCPU and memory requests to identify which pods consume the most resources. Compare the current vCPU and memory use against the limits. The cluster terminates any pods that exceed the limits. • If you set quotas on the platform, services, or projects, check the vCPU and memory use against the thresholds that you set. In addition, compare the vCPU and memory requests against the quotas to determine whether you need to allocate additional resources or whether you can reduce the quotas. • If you are running out of vCPU or memory, determine whether there are any processes that you can stop or whether you need to add more vCPU or memory to your cluster. For example, there might be old jobs or tool runtimes that are consuming resources but that no longer provide value to your organization.

Metric to check	Things to consider
Services status summary	• Check for services in a critical state. This indicates that the service has one of the following issues: – Check if a service instance is in a failed state. – Check if a pod is in a failed or unknown state. • This indicates that the service has one of the following issues: – A service instance in a pending state. – A pod in a pending state. • Typically, a service in this state isn't a cause for concern unless the service has been in this state for a long time. Service instances and pods are in pending state when they are waiting to be scheduled. If they remain in pending state for a long time, it might mean that: Best-practice-routine-cpd-monitoring_pending-reasons: • The cluster has insufficient resources to schedule pods. • The quota settings are preventing pods from being scheduled.
Service instances status summary	• Check for service instances in a critical state . This indicates that the service instance has one of the following issues: – The instance is in a failed state. – A pod in a failed or unknown state. • Check for service instances in a warning state. This warning indicates that the service has one of the following issues: – The instance is in an unknown state. – A pod in a pending state. • Typically, a service instance in this state isn't a cause for concern unless the instance has been in this state for a long time. Service instances and pods are in pending state when they are waiting to be scheduled. If they remain in pending state for a long time, it might mean that: • The cluster has insufficient resources to schedule pods. • The quota settings are preventing pods from being scheduled.
Tool runtimes status summary	Check for tool runtimes that have at least one pod in a failed state. (

Metric to check	Things to consider
Pods status summary	• Check for pods that are in a failed or unknown state. For details, see • Check for pods in a warning state. . Typically, a pod in this state isn't a cause for concern unless the pod has been in this state for a long time. Pods are pending when they are waiting to be scheduled. If they remain in pending state for a long time, it might mean that: • The cluster has insufficient resources to schedule pods. • The quota settings are preventing pods from being scheduled. • Click the pod status summary to see more details: Check for pods with more than one restart. • Check for pods that are running but that have no ready containers or too few ready containers, for example 0/1 ready containers.
Projects summary status	Check the number of projects that are running on the platform. You can click this card to view more detailed information about each project. • Review the vCPU and memory requests to identify which projects consume the most resources. • Look for any orphaned or unused projects that can be deleted to free up resources. • Think about whether they can try to stagger jobs that are running in each project to reduce workload. • Ensure that any environments that are used by the projects are appropriately sized?

What should you do if you find pods in a failed state?

Pods in a failed state indicate an underlying problem with the service. For example, the pod might not be able to pull one or more required images or the pod is overloaded.

Review the pod details to look for status or event information that can help you determine why the pod failed and take the appropriate action to re mediate the issue. For example:

- Ensure that the repository that you pull images from is running.
- Ensure that the appropriate image pull secret exists.
- Increase the resources allocated the service, instance, or runtime.
- Increase the amount of memory of vCPU on the cluster.

If you cannot determine the root cause of the problem, run a diagnostic job to collect the relevant information to open a case with IBM® Support.

2. By default, the platform runs several pre-defined monitors (scripts that check the state of an entity) every 10 minutes.

   - If the monitor reports a critical event 3 times in a row, the platform issues a critical alert .
   - If the monitor reports a warning event 5 times in a row, the platform issues a warning alert .

Default alerting rules

Warning alerts

After 5 occurrences, the platform issues warning alerts for the following events:

- A quota setting is preventing a service from creating new pods.

- A service has one or more service instances or pods in a pending state.
- A service instance has one or more pods in a pending state.

Critical alerts

After 3 occurrences, the platform issues critical alerts for the following events:

- A service does not have enough replicas.

  If this occurs, determine how you can allocate sufficient resources to the service to enable it to create the required number of replicas.
- A persistent volume claim (PVC) is not associated with a storage volume, which means that the service cannot store data.

  If this situation occurs, ensure that you have sufficient storage to create the requested PVC.
- A service has insufficient resources to fulfill requests.

  The service cannot create new pods if the new pods push the service over the memory quota or the vCPU quota. These pods remain in pending state until sufficient resources are available.

  If the resources are insufficient, you can:
- Wait for existing process to complete so that additional resources become available.
- Determine whether you can stop any process to free up resources.
- Adjust the appropriate quota settings to allocate more resources to the platform or to the service.
- Add vCPU or memory to your cluster.
- A service instance is in a failed state or a pod is in a failed or unknown state. For troubleshooting tips, see
- One or more pods that are associated with a service instance are in a failed or unknown state. For troubleshooting tips, see

The platform also issues alerts if one or more of the preceding monitors do not complete successfully.

Perform routine cluster monitoring

Establish a schedule for monitoring your watsonx.data deployments on Red Hat OpenShift Container Platform.

watsonx.data on Red Hat OpenShift

The health of your cluster can have a huge impact on the health of your watsonx.data deployments.

Who needs to perform this task?

A cluster administrator must perform this task.

How frequently must you perform this task?

It is recommended that you perform this task *at least* one time in a day or per shift.

However, if there is a large variation in the number of concurrent users or jobs, it is recommended that you perform this task more frequently during peak activity.

Your routine must include the following tasks:

1. If your storage is remote, ensure that your network is running at 10 Gbps or greater.
2. Run the storage performance validation playbook to confirm that there are no underlying performance issues with your persistent storage.
3. Review the monitoring data from the OpenShift Container Platform web console.**Important:** Ensure that you enable monitoring for the user-defined projects where watsonx.data software is installed.
   - Relevant OpenShift documentation

OpenShiftVersion	Resources
Version 4.10	– Monitoring overview – Enabling monitoring for user-defined projects – Reviewing monitoring dashboards
Version 4.12	– Monitoring overview – Enabling monitoring for user-defined projects – Reviewing monitoring dashboards

Review the following dashboards:

- API Performance
- Kubernetes / Compute Resources / Cluster
- Kubernetes / Compute Resource / Node (Pods)
- Kubernetes / Compute Resources / Namespace(Pods)

4. Check the status of the Operand Deployment Lifecycle Manager objects on the cluster:

   a. Confirm that the catalog sources on the cluster are `Ready`:

   ```
   oc get catalogsource -A \
   -o jsonpath="{range .items[*]}{.metadata.name}{': '}
   {.status.connectionState.lastObservedState}{'\n'}{end}"
   ```

   b. Get information about the watsonx.data operator subscriptions to determine the channel and to confirm that the current CSV is the same as the installed CSV:

   ```
   oc get subscription -n ${PROJECT_CPD_INST_OPERATORS} \
   -o jsonpath="{range .items[*]}{.metadata.name}{' - channel: '}{.spec.channel}{',
   installedCSV: '}{.status.installedCSV}{', currentCSV: '}{.status.currentCSV}{'\n'}{end}"
   ```

   c. Confirm that the operator deployments are ready and have available replicas:

   `oc get deploy -n ${PROJECT_CPD_INST_OPERATORS}`

   d. Check the status of the operator pods and determine whether any of the pods are restarted:

   `oc get pods -n ${PROJECT_CPD_INST_OPERATORS}`

Integrate with Prometheus

Run Prometheus queries to monitor watsonx.data deployments on Red Hat OpenShift Container Platform.

watsonx.data on Red Hat OpenShift

About this task

Who needs to complete this task?
Cluster administrator A cluster administrator must perform this task.

When do you need to complete this task?

- One-time setup: Enable watsonx.data to export metrics to Prometheus only one time after you install watsonx.data.
- Repeat as needed: Run Prometheus queries from the command line or OpenShift interface as often as necessary to monitor your watsonx.datadeployments. It is recommended that you perform these tasks *at least*one time in a day or per shift.

Procedure

Complete the appropriate tasks for your environment:

- **Exporting watsonx.data metrics to Prometheus**

Enable watsonx.data to export metrics to Prometheus so that you can query Prometheus for metric data and view it in the OpenShift Console.

- **Running Prometheus queries from the command line**

  Run Prometheus queries from the command line to monitor CPU, disk, and watsonx.data resource usage.
- **Running Prometheus queries from the OpenShift Console**

  Run Prometheus queries from the OpenShift Console to visualize metric data.

Monitoring watsonx.data Presto (Java) JMX metrics with Grafana in Red Hat OpenShift cluster

You can push watsonx.data Presto (Java) JMX metrics into Red Hat OpenShift Prometheus and use the Grafana dashboard to display the JMX metrics.

watsonx.data on Red Hat OpenShift

The process has three sections.

1. Enable watsonx.data default monitoring in Cloud Pak for Data
2. Export Cloud Pak for Data metrics to Prometheus
3. Integrate Grafana with Prometheus

1. Enable watsonx.data default monitoring in Cloud Pak for Data

watsonx.data uses default monitoring in Cloud Pak for Data to collect Presto (Java) JMX metrics from Presto (Java) pod and push the data into zen-watchdog service. Before starting the configuration, install `cpd-cli` and start the `olm-utils` container. For more information, see Cloud Pak for Data command-line interface (`cpd-cli`).

1. Run the following command to log in to Red Hat OpenShift cluster with `cpd-cli`.

   ```
   cpd-cli manage login-to-ocp -u kubeadmin -p {{ kubeadmin-password }} --server https://
   {{cluster url}}:{{ cluster port }}
   ```

   It takes the following parameters:

   -u
   The username of the OpenShift Container Platform cluster administrator.

   -p
   The password of the OpenShift Container Platform cluster administrator.

   --server
   The URL and port number of the OpenShift Container Platform cluster API server, in the format `<cluster_url>:<port>`.
2. Run the following command to enable watsonx.data default monitoring.

   ```
   cpd-cli manage apply-service-monitor --cpd_instance_ns={{ cpd-instance-namespace }} --
   monitors=cp4d-watsonxdata-info --monitor_schedule="*/10,*,*,*,*"
   ```

   It takes the following parameters:

   --cpd_instance_ns
   The namespace where the CPD instance is installed.

   --monitors
   A comma-separated list of the services to monitor.

   After the new `cronjob` is created, the following confirmation is displayed.

```
oc get cj | grep servicecollection-cronjob
servicecollection-cronjob              */10 * * * *   False     0        4m5s            2d1h
```

2. Export Cloud Pak for Data metrics to Prometheus

After enabling watsonx.data default monitoring in Cloud Pak for Data, the data is pushed from the Presto (Java) JMX metrics into the zen-watchdog service.

1. Export the metrics from the zen-watchdog service into Red Hat OpenShift Prometheus. For more information, see Exporting Cloud Pak for Data metrics to Prometheus. Make sure that all of the required nodes are started successfully.

```
oc get pod -n openshift-monitoring
prometheus-operator-6c84644559-6kvqw                      2/2     Running   0          4d4h
prometheus-operator-admission-webhook-6f5668f5dd-65mw9    1/1     Running   0          4d4h
prometheus-operator-admission-webhook-6f5668f5dd-69gtg    1/1     Running   0          4d4h
thanos-querier-647bd9fcb6-ljnxb                           6/6     Running   0          2d2h
thanos-querier-647bd9fcb6-w2q9n                           6/6     Running   0          2d2h
```

2. Verify in the Red Hat OpenShift console to check that the data is pushed into the **Metrics** page.
   a. Log in to the Red Hat OpenShift console.
   b. In the **Metrics targets** page, make sure that the **zenmetrics** status is **Up**.
   c. Go to **Observe** -> **Metrics** to view the JMX metrics.

3. Integrate Grafana with Prometheus

Install the Grafana operator and dashboard in Red Hat OpenShift. If your cluster is an air-gapped environment, see Red Hat documentation (Mirroring images for a disconnected installation).

1. Create a namespace and install Grafana in the namespace.
2. Create a Grafana instance.

```
cat <<EOF |oc apply -f -
kind: Grafana
apiVersion: grafana.integreatly.org/v1beta1
metadata:
  name: grafana-a
  namespace: my-grafana
spec:
  config:
    auth:
      disable_login_form: 'false'
  log:
    mode: console
  security:
    admin_password: {{ password }}
    admin_user: root
EOF
```

3. Run the following commands to create role-based access control (RBAC) and token.
   a.
   ```
   oc create serviceaccount grafana -n my-grafana
   ```
   b.
   ```
   oc create clusterrolebinding grafana-cluster-monitoring-view   --clusterrole=cluster-
   monitoring-view   --serviceaccount=my-grafana:grafana
   ```
   c.
   ```
   oc create token grafana --duration=100000000s -n my-grafana
   ```
4. Configure the Grafana data source.
   a. Expose Grafana route for Grafana service. For more information, see Configure Grafana.
   b. Log in to Grafana from a web browser by using the username and password that you got in step 2.
   c. Create a new Prometheus data source with the header value as `Bearer {{ token }}` and the URL. To get the URL, run the following command.

   ```
   oc get route -n openshift-monitoring | grep thanos
   ```

5. Create a Grafana dashboard with the metrics in Prometheus. For more information, see Create a dashboard.

Making monitoring data highly available

IBM watsonx.data monitoring data is stored in an embedded Influx Db database. By default, only one replica of the database is deployed. However, if you want to ensure that the monitoring data is highly available, you can increase the number of replicas.

watsonx.data on Red Hat OpenShift

About this task

Permissions that you need to complete this task.

You must be an administrator on the Red Hat OpenShift Container Platform project where watsonx.data is installed.

When you need to complete this task.

It is recommended that you make the monitoring data highly available if you expect to run large workloads. In large or complex deployments, having timely access to monitoring data is critical. Monitoring your deployment can help you prevent unexpected outages. You can complete this task at any time. However, it is recommended that you complete this task before you make watsonx.data available to users.

About this task

Having multiple replicas of the monitoring data improves reliability and availability. In some situations, it can also reduce latency.

Note: Scaling the watsonx.data control plane does not have any impact on the InfluxDB database. Similarly, you can make the monitoring data highly available without scaling the watsonx.data control plane.

Procedure

1. To increase the number of `InfluxDB` replicas:

   Log in to Red Hat OpenShift Container Platform as a user with sufficient permissions to complete the task:

   ```
   oc login ${OCP_URL}
   ```

2. Edit the `ZenService` custom resource. For example,:

   ```
   oc edit ZenService lite-cr --namespace cpd-instance
   ```

3. Add the `zen_monitoring_scale_config` property to the `spec` section of the custom resource.

   ```
   ...
   spec:
   zen_monitoring_scale_config: medium
   ```

4. Save your changes to the `ZenService` custom resource. For example, if you are using `vi`, press ESC and enter `wq`.
5. Check the status of the `ZenService` custom resource:

   ```
   oc get ZenService lite-cr \
   --namespace cpd-instance \
   -o jsonpath="{.status}"
   ```

   Wait for the command to return **`Completed`**.

When you set the `zen_monitoring_scale_config` property to **`medium`**, the Zen operator starts a second replica of the `InfluxDB` database and an instance of the `InfluxDB` relay service, which ensures that the replicas contain the same information.

Getting the status of installed components

A Red Hat OpenShift project administrator can run the `cpd-cli manage get-cr-status` command to determine the status of the IBM watsonx.data components that are installed in a project (namespace).

watsonx.data on Red Hat OpenShift

Before you begin

You must run this task from a client workstation that has `cpd-cli` and oc installed. For details, see Setting up a client workstation.

Procedure

1. Run the `cpd-cli manage login-to-ocp` command to log in to the cluster as a user with sufficient permissions to complete this task. For example:

```
cpd-cli manage login-to-ocp \
--username=${OCP_USERNAME} \
--password=${OCP_PASSWORD} \
--server=${OCP_URL}
```

2. Run the following command to get the status of all of the custom resources in the project:

```
cpd-cli manage get-cr-status \
--cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS}
```

For each component, the output of the command includes the following information:

- The status from the custom resource
- The version of the component
- The timestamp when the custom resource was created

The output has the following format:

```
component,CR-kind,CR-name,status,version,creationtimestamp,reconciled-version,operator-info
```

The following table provides more information about the statuses for `watsonx.data`:

Component	Description	Statuses
`watsonx.data`	IBM watsonx.data	**Completed** The component is running and ready to use. **Failed** The component failed. Check the operator logs for errors. **InProgress** The component is being installed or updated and is not available to use.

Changing the retention period for monitoring data

IBM watsonx.data monitoring data is stored in an embedded InfluxDB database. By default, the data is retained for 30 days. You can increase the retention period to store additional data or decrease the retention period if you need to reduce the size of the database.

watsonx.data on Red Hat OpenShift

About this task

Before you begin

To complete this task, you must be an administrator on the Red Hat OpenShift Container Platform project where IBM watsonx.data is deployed.

About this task

To change the retention period, you must change the settings of the embedded InfluxDB database.

Procedure

1. Log in to your Red Hat OpenShift Container Platform cluster as an instance administrator:

   ```
   oc login OpenShift_URL:port
   ```

2. Change to the project where IBM watsonx.datais deployed:

   ```
   oc project Project_name
   ```

3. Get the password for the InfluxDB service:

   ```
   oc get secret -o yaml dsx-influxdb-auth
   ```

   Locate the `influxdb-password` entry in the response.
4. Decode the password:

   ```
   echo value-from-influxdb-password | base64 -d
   ```

   Save the decoded password somewhere secure.
5. Get the ID of an InfluxDB pod:

   ```
   oc get pods | grep influx
   ```

6. Open a bash shell inside the pod:

   ```
   oc exec oc exec -it dsx-influxdb-0 bash
   ```

7. Determine the current data retention policy.
   a) Connect to InfluxDB:

      ```
      influx -ssl -unsafeSsl
      ```

   b) Authenticate to InfluxDB:

      ```
      auth admin decoded-InfluxDB-password
      ```

   c) View the current retention policy:

      ```
      show retention policies on WATCHDOG;
      ```

      ```
      name         duration  shardGroupDuration replicaN default
      ----         --------  ------------------ -------- -------
      autogen      1440h0m0s 24h0m0s            1        true
      thirty_days  720h0m0s  24h0m0s            1        false
      three_days   72h0m0s   24h0m0s            1        false
      ```

   d) Change the retention policy:

      ```
      ALTER RETENTION POLICY thirty_days ON WATCHDOG DURATION duration REPLICATION 1
      ```

      Use InfluxDB notations to change the duration. For example,:

      - To set the retention policy to 45 days, change `duration` to 45d

- To set the retention policy to 6 weeks, change `duration` to `6w`

For more information about the InfluxDB notation, see Duration units in the InfluxDB documentation.

Monitoring and alerting in watsonx.data

watsonx.data on Red Hat OpenShift

SNMP response codes

Use the following information to interpret the response codes that are sent by watsonx.data when you send alerts to your SNMP server.

watsonx.data on Red Hat OpenShift

Replica status check codes

IBM watsonx.data is configured to maintain a specific number of `Deployment` replicas and `StatefulSet` replicas. The `check-replica-status` event monitors the status of `Deployment` replicas and `StatefulSet` replicas that are associated with watsonx.data and reports any issues.

Response code	Severity	Description
102	Critical	The watsonx.data does not have enough replicas.
100	Information	The monitor that checks the status of the replicas ran. There are no issues to report.

PVC status check codes

A persistent volume claim (PVC) is a request for storage that meets specific criteria, such as a minimum size or a specific access mode. The `check-pvc-status` event monitors the status of the PVCs that are associated with watsonx.data and reports any issues.

Response code	Severity	Description
202	Critical	The PVC is not associated with a storage volume, which means that watsonx.data cannot store data.
200	Information	The monitor that checks the status of the PVCs ran. There are no issues to report.

Quota status check codes

An administrator set a vCPU quota and a memory quota for watsonx.data or for the platform. The `check-quota-status` event monitors the quotas and requests that are associated with watsonx.data to determine whether it has sufficient resources to fulfill requests.

Response code	Severity	Description
302	Critical	watsonx.data has insufficient resources to fulfill requests. watsonx.data cannot create new pods if the new pods push it over the memory quota or the vCPU quota. These pods remain in pending state until sufficient resources are available.
301	Warning	Check the quota settings and the available resources on the cluster.
300	Information	The monitor that checks the status of the PVCs ran. There are no issues to report.

Monitor status check codes.

A monitor is a script that checks the state of an entity periodically and generates events based on the state of the entity. The `check-monitor-status` event monitors the status of Monitoring jobs to determining whether the jobs completed successfully.

Response code	Severity	Description
402	Critical	One or more jobs did not complete successfully.
400	Information	The monitor that checks the status of Monitoring jobs ran. There are no issues to report.

Service check status codes

watsonx.data is consisted of pods and one or more service instances. The `check-service-status` event monitors the status of watsonx.data to determine whether the pods and instances that are associated with it are running as expected.

Response code	Severity	Description
502	Critical	A watsonx.data instance is in a failed state or a pod is in a failed or unknown state.
501	Warning	Check the status of watsonx.data. A pod that is associated with it might be pending.
500	Information	The monitor that checks the status of each service ran. There are no issues to report.

Service instance check status codes

A watsonx.data instance is consisted of one or more pods. The `check-instance-status` event monitors the status of service instances to determine whether the pods that are associated with the instance are running as expected.

Response code	Severity	Description
602	Critical	One or more pods that are associated with the instance are in a failed or unknown state.
601	Warning	Check the status of the instance. A pod that is associated with the instance might be pending.
600	Information	The monitor that checks the status of each instance ran. There are no issues to report.

Creating custom monitors

Developers can set up custom monitors using the alerting framework.

watsonx.data on Red Hat OpenShift

About this task

Developers can set up custom monitors using the alerting framework.

Monitors check the state of entities periodically and generate events that are stored in Metastore database. Administrators might be interested in node resource efficiency, memory quotas, license usage, user management events, and provisioning diagnostics.

Monitors can be registered into watsonx.data through `configmap` extension. The `configmap` extension has all the details that are needed to create a `cron job`, including the details of the script, the image to be used, the schedule for the cron job, and any environment variables. This ensures that the alerting framework has all the necessary information to create a `cron job`, monitor events frequently, and trigger alerts if and when needed.

Procedure

For example, consider the following monitor extension:

```
  extensions: |
      [
      {
          "extension_point_id": "zen_alert_monitor",
          "extension_name": "zen_alert_monitor_diagnostics",
          "display_name": "Diagnostic alert monitor",
          "details": {
            "name":"diagnostics",
            "image": "sample-monitor",
            "command": ["/bin/sh", "-c"],
            "schedule": "*/10 * * * *",
            "args": ["curl -X GET http://zen-watchdog-svc:3333/zen-watchdog/v1/monitoring/
samples/diagnostics"],
            "event_types": [
              {
                "name": "check-replica-status",
                "alert_type": "watsonx.data",
                "purge_frequency": 3,
                "description": "one or more unavailable replicas.",
                "resolution": "Check pods status and associated events."
              },
              {
                "name": "check-pvc-status",
                "alert_type": "watsonx.data",
                "purge_frequency": 3,
                "description": "Unbound or Failed PVC.",
                "resolution": "Check PVC logs."
              },
              {
                "name": "check-resource-status",
                "alert_type": "watsonx.data",
                "purge_frequency": 3,
                "description": "High CPU/Memory usage relative to set quotas.",
                "resolution": "Increase quotas if necessary or revoke running workloads."
              },
              {
                "name": "check-monitor-status",
                "alert_type": "watsonx.data",
                "purge_frequency": 3,
                "description": "Failed jobs as part of the monitor cronjob run.",
                "resolution": "Check pod logs for the failed jobs."
              }
            ]
          }
        }
      ]
```

For more information, see Tutorial: Creating a custom monitor

Tutorial: Creating a custom monitor

You can use this tutorial to build a custom monitor that tracks persistent volume claims and reports events based on the status.

watsonx.data on Red Hat OpenShift

About this task

If a PVC is bound, information is registered for the PVC. If a PVC is in an unbound or failed state, a critical event is recorded. This monitor is based on Python 3.7 and uses the Kubernetes Python SDK to interact with the cluster.

Procedure

1. Create the Python script to monitor and record PVC status.

   The following script uses the in-cluster config to authenticate to Kubernetes and access the resources. By default, you have access to the following volumes:

   `user-home-pvc`

   `zen-service-broker-secret`

   The following environment variables are made available as part of the cron job initialization:

   - `ICPD_CONTROLPLANE_NAMESPACE`: The control plane namespace.

   The following Python script lists PVCs and generates events based on their state. All nonbound PVCs are recorded with critical severity. The events are sent as a JSON array to the POST events endpoint, authorized with the service broker token.

```
import os
import requests
import json
from kubernetes import client, config, watch


def main():
    # setup the namespace
    ns = os.environ.get('ICPD_CONTROLPLANE_NAMESPACE')
    if ns is None:
        ns = ""
    monitor_type = "sample-monitor"
    event_type = "check-pvc-status"

    # configure client
    config.load_incluster_config()
    api = client.CoreV1Api()

    # configure post request and set secret headers
    url = 'https://zen-watchdog-svc:4444/zen-watchdog/v1/monitoring/events'
    with open('/var/run/sharedsecrets/token', 'r') as file:
        secret_header = file.read().replace('\n', '')
    headers = {'Content-type': 'application/json', 'secret': secret_header}

    # Print PVC list, set status as critical for unbound or failed pvc
    pvcs = api.list_namespaced_persistent_volume_claim(namespace=ns, watch=False)
    events = []
    for pvc in pvcs.items:
        severity = "info"
        print(pvc.status.phase)
        if pvc.status.phase != 'Bound':
            severity = "critical"
        data = {"monitor_type":monitor_type, "event_type":event_type,
"severity":severity, "metadata":"PVC Bound", "reference":pvc.metadata.name}
        events.append(data)
    json_string = json.dumps(events)
    # post call to zen-watchdog to record events
    r = requests.post(url, headers=headers, data=json_string, verify=False)
    print(r.status_code)

if __name__ == '__main__':
    main()
```

2. Create and run the Dockerfile.

   For the Python script to run in a containerized environment, you need access to a Python executable with Kubernetes Python SDK included.

   You need the following requirements.txt:

   `kubernetes==11.0.0`

   You need the following Dockerfile:

```
# set base image (host OS)
FROM python:3.8
RUN mkdir /pvc-monitor
# set the working directory in the container
WORKDIR /pvc-monitor
ADD . /pvc-monitor
# install dependencies
RUN pip install -r requirements.txt
# command to run on container start
CMD [ "python", "./pvc_check.py" ]
```

   The requirements.json file holds all the required packages for the image. The final structure of the project is similar to the following example:

```
-----Python app to monitor and alert for PVCs
Pvc-monitor/
  Pvc-monitor.py
  Requirements.txt
  Dockerfile
```

   When this structure is in place, you can build the Docker image by using the following command:

   `docker build -f Dockerfile -t pvc-monitor:latest .`

   Finally, tag and push the Docker image into the OpenShift® registry so that the alerting cron job can access it.

```
docker tag <docker-image-id> <docker-registry>/<namespace>/pvc-monitor:latest
docker push <docker-registry>/<namespace>/pvc-monitor:latest
```

3. Set up the extension configmap for monitor

   After you push the Docker image to the OpenShift registry, you can create an extension configmap that points to the image. This step ensures that the alert manager picks up the image and creates a cron job, which ensures that the script is run at scheduled intervals.

   You can use the following sample configmap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: zen-alert-monitor-extensions
  labels:
    icpdata_addon: "true"
    icpdata_version: "1.0"
data:
  extensions: |
      [
      {
          "extension_point_id": "zen_alert_monitor",
          "extension_name": "zen_alert_monitor_sample",
          "display_name": "Sample alert monitor",
          "details": {
            "name":"sample-monitor",
            "image": "<docker-registry>/<image-name>:<image-tag>",
            "schedule": "*/10 * * * *",
            "event_types": [
              {
                "name": "check-pvc-status",
                "alert_type": "platform",
                "purge_frequency": 3,
                "description": "Unbound or failed PVC.",
                "resolution": "Check PVC logs."
              }
            ]
          }
        }
      ]
```

Managing resources

watsonx.data on Red Hat OpenShift

Scaling resources for services

.

watsonx.data on Red Hat OpenShift

Manually scaling resources
You can adjust IBM watsonx.data service by scaling the components that they use to support high availability or to increase processing capacity. Components can be scaled based on predefined properties configurations.

watsonx.data on Red Hat OpenShift

Permissions that you need for this task
You must be either:

- A cluster administrator.
- An administrator of the project where the software is installed, for example `watsonx-data-instance`.

When you need to complete this task
If a service supports scaling, you can scale the service at any time after you install it.

Scaling changes the capacity of components by adjusting the memory resource settings, the number of CPUs, and the number of pods that are available. Scaling a service component from small to medium increases the processing capacity of the application.

Before you begin

Before you scale up a service, make sure that your cluster can support the additional workload. If necessary, contact your IBM Support representative. When you change the scaling configuration of a service, you also must scale the related services individually. For more information about service dependencies, see Software requirements, and then scale the services as necessary.

Best practice: Set up the environment variables and run the commands in this task exactly as written. For instructions, see Setting up installation environment variables.

Make sure that you source the environment variables before you run the commands in this task.

Procedure

Complete the steps for the service that you are scaling:

1. Set the SIZE environment variable to the appropriate value for your environment.

   ```
   export SIZE=<size>
   ```

   The following sizes are supported:

   - small (default)
   - small_mincpureq
   - medium
   - large
   - xlarge
   - xxlarge
2. Follow the steps to scale the addon components **UI**, **API**, and **Nodeclient** and verify whether scaling is completed.
   a) Run the following command to scale the addon components **UI**, **API**, and **Nodeclient**:

   ```
   oc patch wxdaddon/wxdaddon \
     --type=merge \
      -n ${PROJECT_CPD_INST_OPERANDS} \
      -p '{ "spec": { "scaleConfig": "'$SIZE'" } }'
   ```

   b) Run the following command to verify whether correct *scaleConfig* is applied to the addon components:

   ```
   watch --color "oc get wxdaddon -n ${PROJECT_CPD_INST_OPERANDS} -o custom-
   columns='NAME:metadata.name, RECONCILE:status.wxdStatus, SIZE:spec.scaleConfig'"
   ```

   Note: The reconcile status changes from In-progress to Completed, indicating successful reconciliation.

   If reconciliation is unsuccessful, make sure the entered Size value is accurate. If the Size value is correct, contact the support team for assistance.
3. Follow the steps to scale the instance components **Postgres**, **Hive-metastore**, and **MinIO** and verify whether scaling is completed.
   a) Run the following command to scale the instance components **Postgres**, **Hive-metastore**, and **MinIO**:

   ```
   oc patch wxd/lakehouse \
     --type=merge \
      -n ${PROJECT_CPD_INST_OPERANDS} \
      -p '{ "spec": { "scaleConfig": "'$SIZE'" } }'
   ```

   b) Run the following command to verify whether correct *scaleConfig* is applied to the instance components:

   ```
   watch --color "oc get wxd -n ${PROJECT_CPD_INST_OPERANDS} -o custom-
   columns='NAME:metadata.name, RECONCILE:status.wxdInstanceStatus, SIZE:spec.scaleConfig'"
   ```

   Note: The reconcile status changes from In-progress to Completed, indicating successful reconciliation.

   If reconciliation is unsuccessful, make sure the entered Size value is accurate. If the Size value is correct, contact the support team for assistance.
4. Follow the steps to scale Presto engine components **Presto**, **Presto-coordinator**, and **Presto-worker** and verify whether scaling is completed.
   a. Run the following command to get the Presto engine name:

   ```
   oc get wxdengine -n ${PROJECT_CPD_INST_OPERANDS}
   ```

```
oc get wxdengine
NAME                  AGE
lakehouse-presto-01   75m
```

b. SIZE environment variable selection:

   Run the following commands to scale Presto components in the pods:

```
oc patch wxdengine/lakehouse-presto-01 \
  --type=merge \
   -n ${PROJECT_CPD_INST_OPERANDS} \
   -p '{ "spec": { "scaleConfig": "'$SIZE'" } }'
```

c. DEPLOY_TYPE environment variable selection:

   According to the Presto that is installed, `lakehouseDeploymentType` can be `singlenode` or `multinode`.

   Set the DEPLOY_TYPE environment variable to the appropriate value for your environment.

```
export DEPLOY_TYPE=<DEPLOY_TYPE>
```

   Run the following commands to set the deployment type for Presto:

```
oc patch wxdengine/lakehouse-presto-01 \
  --type=merge \
   -n ${PROJECT_CPD_INST_OPERANDS} \
   -p '{ "spec": { "lakehouseDeploymentType": "'$DEPLOY_TYPE'" } }'
```

   Note: `singlenode` refers to only one **`Presto`** pod and `multinode` refers to **`Presto-coordinator`** and **`Presto-worker`**.

d. Run the following command to verify whether correct *scaleConfig* is applied to the instance components:

```
watch --color "oc get wxdEngine -n $
{PROJECT_CPD_INST_OPERANDS} -o custom-columns='NAME:metadata.name,DISPLAY
NAME:spec.engineDisplayName,RECONCILE:status.engineStatus,STATUS:status.middleEndStatus,
SIZE:spec.scaleConfig'"
```

   Note: The reconcile status changes from `In-progress` to `Completed`, indicating successful reconciliation.

   If reconciliation is unsuccessful, make sure the entered `Size` value is accurate. If the Size value is correct, contact the support team for assistance.

Automatically scaling resources
Some IBM watsonx.data components support autoscaling by using the Red Hat OpenShift Horizontal Pod Autoscaler (HPA). The HPA changes the resource allocation of components by increasing or decreasing the number of pods in response to CPU or memory consumption.

watsonx.data on Red Hat OpenShift

Permissions that you need for this task
You must be either:

- A cluster administrator
- An administrator of the project where the software is installed, for example `watsonx-data-instance`

When you need to complete this task

When you want to dynamically scale resources for services based on usage.

Instead of using the manual, fixed approach to scaling resources for components, you can enable autoscaling to dynamically scale up and down. The scaling is based on the user workloads to ensure that clusters are not underutilized while available resources cannot be used by workloads that require more resources.

Before you begin

Best practice: You can run the commands in this task exactly as written if you set up environment variables. For instructions, see Setting up installation environment variables.

Ensure that you source the environment variables before you run the commands in this task.

Procedure

1. Run the following command to enable horizontal pod autoscaling in the addon components **`UI`**, **`API`**, and **`Nodeclient`**:

   ```
   oc patch wxdaddon/wxdaddon \
     --type=merge \
      -n ${PROJECT_CPD_INST_OPERANDS} \
      -p '{ "spec": { "autoScaleConfig": true } }'
   ```

2. Follow the steps to enable horizontal pod autoscaling of **`Presto workers`**:

   a. Run the following command to get the Presto engine name:

   ```
   oc get wxdengine
   ```

   ```
   oc get wxdengine
   NAME                   AGE
   lakehouse-presto-01    75m
   ```

   b. Run the following commands to enable horizontal pod autoscaling in the pods:

   ```
   oc patch wxdengine/lakehouse-presto-01 \
     --type=merge \
      -n ${PROJECT_CPD_INST_OPERANDS} \
      -p '{ "spec": { "autoScaleConfig": true, "lakehouseDeploymentType": "multinode" } }'
   ```

Shutting down and restarting watsonx.data version 1.1.0

When you do not use IBM watsonx.data, but you plan to use it in the future, you can manually shut down watsonx.data to prevent it from using resources. You can restart watsonx.data when you need to use it again.

watsonx.data on Red Hat OpenShift

Permissions that you need for this task

You must be either:

- A cluster administrator
- An administrator of the project where the software is installed, for example `watsonx-data-instance`

When you need to complete this task

When you want to temporarily shutdown watsonx.data. For example, over holidays or during maintenance, you might shut down watsonx.data to conserve CPU and memory resources, and then restart watsonx.data when you want to use it again.

Important: You cannot shut down watsonx.data when autoscaling is enabled for the service. For more information, see "Automatically scaling resources" on page 165.

Procedure

1. To shut down watsonx.data service, you must shut down the following components of the service:

   a) To shut down **wxd**, run the following command:

   ```
   oc patch wxd lakehouse \
   --namespace ${PROJECT_CPD_INST_OPERANDS} \
   ```

```
--patch '{"spec":{"shutdown":"true"}}' \
--type=merge
```

Alternatively, to force shutdown, run the following command:

```
oc patch wxd lakehouse \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"force"}}' \
--type=merge
```

Note: When **wxd** is shutdown, the Postgres server process stops. However, the Postgres pods that are associated with IBM watsonx.data instance remain running.

b) To shut down **wxdAddon**, run the following command:

```
oc patch wxdAddon wxdaddon \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"true"}}' \
--type=merge
```

Alternatively, to force shutdown, run the following command:

```
oc patch wxdAddon wxdaddon \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"force"}}' \
--type=merge
```

c) To shut down **wxdEngine**, you must first get the Presto engine name and run the command for shutdown:

i) Get the Presto engine name that you want to shut down. Run the following command:

```
oc get wxdengine --namespace ${PROJECT_CPD_INST_OPERANDS}
```

Example of the result:

```

[admin@bastion-gym-lan work]$ oc get wxdengine --namespace ${PROJECT_CPD_INST_OPERANDS}
NAME                  VERSION   TYPE     DISPLAY NAME   SIZE    RECONCILE   STATUS
AGE
lakehouse-presto-01   1.1.4     Presto   presto-01      small   Completed   RUNNING
2d19h
```

ii) To shut down **wxdEngine**, run the following command:

```
oc patch wxdEngine <engine_instance> \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"true"}}' \
--type=merge
```

Alternatively, to force shutdown, run the following command:

```
oc patch  wxdEngine <engine_instance>  \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"force"}}' \
--type=merge
```

Note: When you shut down watsonx.data, the service pods are terminated and removed from the $
{PROJECT_CPD_INST_OPERANDS} project.

2. To restart watsonx.data service, do the following steps:

a) Run the following command to restart **wxd**:

```
oc patch wxd lakehouse \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"false"}}' \
--type=merge
```

b) Run the following command to restart **wxdAddon**:

```
oc patch wxdAddon wxdaddon \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"false"}}' \
--type=merge
```

c) To restart **wxdEngine**, you must first get the Presto engine name and run the command to restart:

i) Get the Presto engine name that you want to restart. Run the following command:

```
oc get wxdengine --namespace ${PROJECT_CPD_INST_OPERANDS}
```

Example of the result:

```
[admin@bastion-gym-lan work]$ oc get wxdengine --namespace ${PROJECT_CPD_INST_OPERANDS}
NAME                  VERSION   TYPE     DISPLAY NAME   SIZE    RECONCILE   STATUS
AGE
lakehouse-presto-01   1.1.4     Presto   presto-01      small   Completed   RUNNING
2d19h
```

ii) Run the following command to restart **wxdEngine**:

```
oc patch wxdEngine <engine_instance> \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"false"}}' \
--type=merge
```

Shutting down and restarting watsonx.data version 1.1.1 and later
In watsonx.data version 1.1.1 and later, shutting down the watsonx.data service also shuts down **wxd**, engine, and service components. You can also shut down **wxd**, engine, and service components individually.

When watsonx.data service is shutdown, the PostgreSQL server process stops. However, the PostgreSQL pods that are associated with watsonx.data instance remain in `Running` state.

Procedure

- To shut down a watsonx.data service, run the following command:

```
cpd-cli manage shutdown \
--components=watsonx_data \
--cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS}
```

Note: To shut down an engine or a service, do the following steps:

1. Run the following command with the namespace to get the engine or service instance names:

```
oc get wxdengine --namespace ${PROJECT_CPD_INST_OPERANDS}
```

Example of the result:

```
[admin@bastion-gym-lan work]$ oc get wxdengine --namespace ${PROJECT_CPD_INST_OPERANDS}
NAME                  VERSION   TYPE     DISPLAY NAME   SIZE    RECONCILE   STATUS    AGE
lakehouse-presto-01   1.1.4     Presto   presto-01      small   Completed   RUNNING
2d19h
```

2. To shut down the engine or service, run the following command:

```
oc patch wxdEngine <engine_or_service_instance> \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"true"}}' \
--type=merge
```

Alternatively, to force shutdown, run the following command:

```
oc patch  wxdEngine <engine_or_service_instance>  \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"force"}}' \
--type=merge
```

Note: When you shut down the watsonx.data service, the service pods are terminated and removed from the `${PROJECT_CPD_INST_OPERANDS}` project.

- To restart a shut-down watsonx.data service, run the following command:

```
cpd-cli manage restart \
  --components=watsonx_data \
  --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS}
```

Note: Restarting a watsonx.data service restarts all the engines and services except those, which were shut down individually.

Note: To restart an engine or a service, do the following steps:

1. Run the following command with the namespace to get the engine or service instance names:

```
oc get wxdengine --namespace ${PROJECT_CPD_INST_OPERANDS}
```

Example of the result:

```
[admin@bastion-gym-lan work]$ oc get wxdengine --namespace ${PROJECT_CPD_INST_OPERANDS}
NAME                  VERSION   TYPE     DISPLAY NAME   SIZE    RECONCILE   STATUS    AGE
lakehouse-presto-01   1.1.4     Presto   presto-01      small   Completed   RUNNING   
2d19h
```

2. Run the following command to restart the engine or service:

```
oc patch wxdEngine <engine_or_service_instance> \
--namespace ${PROJECT_CPD_INST_OPERANDS} \
--patch '{"spec":{"shutdown":"false"}}' \
--type=merge
```

Adding nodes to your watsonx.data cluster

You can add nodes to scale out an existing IBM watsonx.data cluster on Red Hat OpenShift.

watsonx.data on Red Hat OpenShift

Ensure that any node that you plan to add meets the hardware requirements for compute nodes.

To add a node to your watsonx.data cluster, follow the appropriate steps for your environment:

Environment	**Instructions**
On premises	For instructions on adding a node to an on-premises OpenShift cluster, see *Adding RHEL compute machines to an OpenShift Container Platform cluster* in the Red Hat OpenShift Container Platform documentation: • Version 4.10 4.6.x • Version 4.12 4.6.4 or later
IBM Cloud Pak for Data System	See Adding new nodes to the system in IBM Cloud Pak for Data System documentation.

Environment	Instructions
Amazon Web Services	The options depend on how you installed watsonx.data: **AWS Quick Start** You cannot scale your cluster. **Terraform** See Auto Scaling in the Terraform installation instructions.
Microsoft Azure	The options depend on how you installed watsonx.data: **Microsoft Azure Quick Start** You cannot scale your cluster. **Terraform** See Auto Scaling in the Terraform installation instructions.
Google Cloud	See Scaling a MachineSet manually in the Red Hat OpenShift documentation.

Configuration best practice

From 1.1.1 release of watsonx.data, some settings are enabled by default to improve the performance. You can still customize these settings based on your needs.

`JVM Xmx` is calculated based on the selected t-shirt size. If the memory allocated to the Presto (Java) pod is more than 8GB, 80% of memory is assigned to `JVM Xmx` else 70% is assigned. The users still have an option to customize the value.

watsonx.data on Red Hat OpenShift

Following properties are set to maximum by default based on `JVM Xmx` allocated to Presto (Java) coordinator or worker:

- Maximum total memory per node (**`query_max_total_memory_per_node`**): `JVM Xmx` * 0.795
- Maximum memory per node (**`query_max_memory_per_node`**): `JVM Xmx` * 0.795
- Memory heap headroom per node (**`memory_heap_headroom_per_node`**): `JVM Xmx` * 0.2
- Maximum memory for query (**`query_max_memory`**): 1TB
- Maximum total memory for query **`query_max_total_memory`**): 2TB

You can customize these properties.

If you have a customized `JVM`, the properties are calculated based on that customized `JVM`. You can customize the `JVM` to resolve the OOM issues and maximize the memory usage.

Following cache types are enabled by default:

- Metastore versioned cache
- Header and footer cache (metadata cache) for Parquet and ORC and DWRF

Metadata caching is calculated considering `JVM Xmx`.

Audit events

The following list describes auditable events that are generated for watsonx.data and forwarded by the Audit Logging Service.

watsonx.data on Red Hat OpenShift

Authorization and authentication management

- `lakehouse.ams_engines.create` - An event is generated when you create an engine user.
- `lakehouse.ams_engines.update` - An event is generated when you update an engine user.
- `lakehouse.ams_engines.delete` - An event is generated when you delete an engine user.
- `lakehouse.ams_buckets.create` - An event is generated when you create a storage user.
- `lakehouse.ams_buckets.update` - An event is generated when you update a storage user.
- `lakehouse.ams_buckets.delete` - An event is generated when you delete a storage user.
- `lakehouse.ams_catalogs.create` - An event is generated when you create a catalog user.
- `lakehouse.ams_catalogs.update` - An event is generated when you update a catalog user.
- `lakehouse.ams_catalogs.delete` - An event is generated when you delete a catalog user.
- `lakehouse.ams_databases.create` - An event is generated when you create a database connection user.
- `lakehouse.ams_databases.update` - An event is generated when you update a database connection user.
- `lakehouse.ams_databases.delete` - An event is generated when you delete a database connection user.
- `lakehouse.ams_metastores.create` - An event is generated when you create a metastore user.
- `lakehouse.ams_metastores.update` - An event is generated when you update a metastore user.
- `lakehouse.ams_metastores.delete` - An event is generated when you delete a metastore user.
- `lakehouse.ams_data_policies.create` - An event is generated when you create a data policy.
- `lakehouse.ams_data_policies.update` - An event is generated when you update a data policy.
- `lakehouse.ams_data_policies.delete` - An event is generated when you delete a data policy.

Console

- `lakehouse.bucket_registrations.create` - An event is generated when you create a storage.
- `lakehouse.bucket_registrations.get` - An event is generated when you get storages.
- `lakehouse.bucket_registrations.update` - An event is generated when you update a storage.
- `lakehouse.bucket_registrations.delete` - An event is generated when you delete a storage.
- `lakehouse.bucket_registrations.activate` - An event is generated when you activate a storage.
- `lakehouse.catalogs.create` - An event is generated when you create a catalog.
- `lakehouse.catalogs.get` - An event is generated when you get catalogs.
- `lakehouse.catalogs.update` - An event is generated when you update a catalog.
- `lakehouse.catalogs.delete` - An event is generated when you delete a catalog.
- `lakehouse.database_registrations.create` - An event is generated when you create a database.
- `lakehouse.database_registrations.get` - An event is generated when you get databases.
- `lakehouse.database_registrations.update` - An event is generated when you update a database.
- `lakehouse.database_registrations.delete` - An event is generated when you delete a database.

Presto Rest Server

- `presto.governance.update` - An event is generated when you add or update the WKC properties.

- `presto.catalog.update` - An event is generated when you update a catalog.
- `presto.server.start` - An event is generated when you star the Presto server.
- `presto.server.restart` - An event is generated when you restart the Presto server.
- `presto.server.stop` - An event is generated when you stop the Presto server.
- `presto.keystore.update` - An event is generated when you update the keystore.
- `presto.dumps.triggered` - An event is generated when you trigger the Presto dumps (thread/heap).

Spark

- `lakehouse.spark_engine.list` - An event is generated when you list Spark engines.
- `lakehouse.spark_engine.create` - An event is generated when you create a Spark engine.
- `lakehouse.spark_engine.update` - An event is generated when you update the details of a Spark engine.
- `lakehouse.spark_engine.delete` - An event is generated when you delete a Spark engine.
- `lakehouse.spark_engine_application.list`- An event is generated when you list the Spark applications under a Spark engine.
- `lakehouse.spark_engine_application.get` - An event is generated when you get the details of a Spark application.
- `lakehouse.spark_engine_application.create` - An event is generated when you run a Spark application.
- `lakehouse.spark_engine_application.delete` - An event is generated when your stop a running Spark application.
- `lakehouse_spark_engine_application.ui` - An event is generated when you access the Spark UI of an application.
- `lakehouse.spark_engine_history_server.get` - An event is generated when you get the Spark history server details.
- `lakehouse.spark_engine_history_server.start` - An event is generated when you start the Spark history server.
- `lakehouse.spark_engine_history_server.stop` - An event is generated when you stop the Spark history server.
- `lakehouse.spark_engine_history_server.ui` - An event is generated when you access Spark history UI.
- `lakehouse.spark_engine_catalog.list` - An event is generated when you list Spark engine associated catalogs.
- `lakehouse.spark_engine_catalog.add` - An event is generated when catalogs are associated with a Spark engine.
- `lakehouse.spark_engine_catalog.remove` - An event is generated when catalogs are dissociated from a Spark engine.
- `lakehouse.spark_engine_catalog.get` - An event is generated when you get the details of a Spark engine catalog.
- `lakehouse.spark_engine_cluster.list` - An event is generated when you list Spark lab clusters.
- `lakehouse.spark_engine_cluster.create` - An event is generated when you create a Spark lab cluster.
- `lakehouse.spark_engine_cluster.get` - An event is generated when you get the details of a Spark lab cluster.
- `lakehouse.spark_engine_cluster.delete` - An event is generated when you delete a Spark lab cluster.

- `lakehouse.spark_engine_cluster.connect` - An event is generated when you connect with a Spark lab cluster.

Managing users

As an administrator, you are responsible for determining and implementing the best approach for authenticating and managing IBM watsonx.data users.

watsonx.data on Red Hat OpenShift

Best practices

IBM watsonx.data user records are stored in an internal repository database. However, it is strongly recommended that you use an enterprise-grade password management solution, such as SAML SSO or an LDAP provider for password management.

You can use SAML SSO and LDAP together or individually.

SAML SSO

If you plan to use SAML for single sign-on (SSO), it is strongly recommended that you complete the steps in "Configuring single sign-on" on page 129 before you add users.

If you add users before you configure SSO, you will need to re-add the users with their SAML ID to enable them to use SSO.

LDAP

You can use an enterprise-grade LDAP provider for password management.

For details, see "Connecting to your identity provider" on page 174.

Ensure that you grant watsonx.data administrator privileges to a user in your LDAP server.

After you grant an LDAP user administrator privileges, you can further secure your watsonx.data system by disabling the default **admin** user. For details, see "Disabling the default admin user" on page 187.

SAML SSO and LDAP

If you want to use both SAML SSO and LDAP, you must ensure that both configurations use the same attribute to identify users:

- SAML SSO configuration: **`fieldToAuthenticate`**
- LDAP: **User search field**

User management

A IBM watsonx.data administrator can manage the permissions that users and groups have on the *platform*. However, users might have additional permissions in services, catalogs, and projects. For example, a user could be a project administrator and be an editor on the **Platform connections** catalog.

A user or group can have multiple roles. Additionally, a user can have roles that are directly assigned to them and roles that they inherit from groups.

If a user has multiple roles, the user has all of the permissions from all of the roles that are assigned to them.

Tip: You can see all of the roles (and permissions) that a user has from the user's profile page, which you can access from the **Access control** > **Users** page.

For a summary of all the permissions that a user has, click **View assigned permissions**.

Important:

If you update a user's role or their group membership and the user is logged in, the user must log out and log back in for the changes to take effect. If the user does not log out, their session will be refreshed after the `TOKEN_EXPIRY_TIME` is reached. For details, see "Setting the idle session timeout" on page 134.

Before you add users to the platform, consider the following questions:

- Are you using the Identity and Access Management Service (IAM Service) from IBM Cloud Pak foundational services to manage users?
- If you are not using the IAM Service:
  - Do you want to use an LDAP server to manage users' passwords?
  - Do you want to use an LDAP server to manage access to the platform?
- Do you want to use user groups to manage users with similar access requirements?
- Do you want to add all the users in an LDAP group to a user group?
- Do the default roles meet your business requirements?

Jump to the appropriate topic for more information:

Connecting to your identity provider

You can optionally configure a connection to an existing identity provider, such as an LDAP server. At a minimum, you can use the identity provider to validate users' credentials. However, you can also use your identity provider to manage access to the platform. The information that you specify when you connect to your identity provider determines whether you use the identity provider for password management or for access management.

watsonx.data on Red Hat OpenShift

Restriction: If you configure a connection to an identity provider, all password management tasks, such as changing or resetting passwords, must be completed by the identity provider administrator.

If the Identity and Access Management Service (IAM Service) is not enabled, you can connect to a single LDAP server.

If the IAM Service is enabled, you can connect to multiple identity providers.

Permissions you need for this task

The permissions that you must have depend on whether watsonx.data is configured to use the Identity and Access Management Service (IAM Service):

IAM Service is not configured (default)

To configure the connection to your LDAP server, you must have one of the following permissions:

- **Administer platform**
- **Manage platform roles**

IAM Service is configured

To configure the connection to your identity provider, you must have the **Administer platform** permission.

When you need to complete this task

Complete this task when you set up watsonx.data.

You can configure a connection to your LDAP server from the **Access control** page.

Follow the appropriate instructions for your environment:

- The IAM service is not enabled

  If the IAM Service is not enabled, the **Access control** page includes a link to the **LDAP configuration**.
- The IAM service is enabled

  If the IAM Service is enabled, the **Access control** page includes a link to the **Identity provider configuration**.

The IAM service is not enabled

1. Log in to the Cloud Pak for Data web client.

2. From the menu, click **Administration** > **Access control**.
3. Click **LDAP configuration**.
4. In the **LDAP server information** section, provide the following information about your LDAP server:

Field	Description
LDAP protocol	• If you are connecting to a secure port on your LDAP server, select **ldaps://**. • If you are connecting to an unsecured port on your LDAP server, select **ldap://**.
LDAP hostname	Enter the host name of the LDAP server.
LDAP port	Enter the port that you are connecting to. Standard ports are 389 for `ldap` and 636 for `ldaps`.
User search base	Enter the point in the LDAP tree from which users are searched. This is also referred to as the `baseDN` for the LDAP configuration.
User search field	Enter the LDAP attribute that is used to identify users. For example, `cn`, `uid`, or `sAMAccountName`. If you plan to use LDAP and a SAML identity provider, ensure that you use the same attribute to identify users. This field should have the same value as the **`fieldToAuthenticate`** parameter in your SSO configuration.
Domain search user	If your LDAP server requires authentication to perform lookups, enter the username of a user that can perform lookups on the LDAP server. This is also referred to as the `bindDN` for the LDAP configuration.
Domain search password	If you specified a **Domain search user**, specify the password for this user.

5. If you want to add LDAP groups to user groups, select **Use LDAP group** and provide the following information about your LDAP server:

Field	Description
Group search base	Enter the point in the LDAP tree from which groups are searched.
Group search field	Enter the LDAP attribute that is used to identify groups. For example, `cn`.

6. If you want to use the LDAP server to manage access to the platform, provide the LDAP attributes that map to the following values:

Field	Description
First name	Enter the LDAP attribute that is used to identify a user's given name. For example, **`givenName`**.
Last name	Enter the LDAP attribute that is used to identify a user's surname. For example, **`sn`**.
Email	Enter the LDAP attribute that is used to identify a user's email address. For example, **`mail`**.
Group membership	If you selected **Use LDAP group**, enter the LDAP attribute that is used to identify all of the LDAP groups that a user is a member of. For example **memberOf**.

Field	Description
Group member field	If you selected **Use LDAP group**, enter the LDAP attribute that is used to identify all of the members of a given group. For example **member**. If you use Microsoft Active Directory and you want to enable the nested groups search, add the following extension ID to the LDAP attribute: `:1.2.840.113556.1.4.1941:` For example: `memberOf:1.2.840.113556.1.4.1941:` **Important:** If you use nested group search in Microsoft Active Directory, you must disable the default LDAP sync on log in option and enable the periodic sync job. For details, see "Syncing Cloud Pak for Data with your LDAP server" on page 176.

7. To verify that you can connect to your LDAP server, enter the following information in the **Test connection** section:

Field	Description
Username	Enter the username of a user that exists in one of the following locations: • The user search base • The group search base
Password	Enter the password for the specified user.

   Note: These credentials are not saved.
8. Click **Test connection**.
9. After you verify that you can connect to your LDAP server, click **Save**.

Syncing Cloud Pak for Data with your LDAP server

Cloud Pak for Data provides two options for syncing with your LDAP server:

Sync on log in (default)

This is the default method. When this method is used, the platform syncs each user's data when the user logs in to Cloud Pak for Data:

- The first time that a user logs in to Cloud Pak for Data, the platform creates a user profile and assigns the user the correct user groups based on their LDAP group membership.
- If the user has logged in before, the platform updates the use group membership based on their LDAP group membership.

This is the recommended method for most environments. If you want to continue using this method, no additional action is required.

Periodic sync job

This option is required if you use nested groups in Microsoft Active Directory. However, this method can cause a lot of overhead for Cloud Pak for Data instances that have large LDAP groups.

If you want to use this method:

1. Log in to Red Hat OpenShift Container Platform as a project administrator:

```
oc login OpenShift_URL:port
```

2. Disable the sync on log in (`LDAP_SYNC_ON_LOGIN`) by running the following command:

```
oc patch configmap product-configmap \
--namespace <cpd-instance> \
--patch '{"data": {"LDAP_SYNC_ON_LOGIN" : "false"}}'
```

3. Delete the `usermgmt` pods:

```
oc delete pod \
--namespace <cpd-instance> \
-l component=usermgmt
```

4. Enable the periodic sync job:

```
oc patch cj usermgmt-ldap-sync-cron-job \
--namespace <cpd-instance> \
--patch '{"spec": {"suspend": false}}'
```

The IAM Service is enabled

You can configure a connection to one or more identity providers from the IBM Cloud Pak Administration Hub.

To access the IBM Cloud Pak Administration Hub

1. Log in to the Cloud Pak for Data web client.
2. From the menu, click **Administration** > **Access control**.
3. Click **Identity provider configuration**.

To configure a connection to an identity provider, see Configuring an LDAP connection in the IBM Cloud Pak foundational services documentation.

Predefined roles and permissions

The permissions and predefined roles that are available depend on the services that are installed on top of watsonx.data. Administrator is the predefined role in watsonx.data. When you add a user or group, the administrator must specify the role that they have.

watsonx.data on Red Hat OpenShift

Jump to the appropriate section for more information:

- “Predefined roles” on page 177
- “Roles assigned to the admin user” on page 178
- “Permissions” on page 178

Predefined roles

A *role* defines the permissions that a user or group has.

A user or group can have multiple roles. Also, a user can have roles that are directly assigned to them and roles that they inherit from groups.

You can edit the default roles or create new roles if the default set of permissions in a role does not align with your business needs.

Administrator

The role is created by the watsonx.data control plane.

The following table specifies the permissions that are associated with the **Administrator** role.

Note: The watsonx.data control plane contributes to the Administrator permission.

Permission	Category
Administer platform	Platform administration
Manage configurations	Platform administration
Manage platform health	Platform administration

Permission	Category
View platform health	Platform administration
Create service instances	Service instances
Manage service instances	Service instances
Manage platform roles	User administration
Manage user groups	User administration
Manage users	User administration
Add vaults	Vaults
Manage vaults and secrets	Vaults
Share secrets	Vaults

Note: For details about permissions associated with the Administrator role, see Permissions.

User

The role is created by the watsonx.data control plane.

By default, no permissions are associated with this role.

If you do not install any services that contribute permissions to this role, users who are assigned to the **User** role can:

- Sign in to watsonx.data.
- Access any services or assets that do not require explicit permissions.

In addition, users who own or manage the components in watsonx.data can give the users access to those components.

Roles assigned to the `admin` user

When you install watsonx.data, the following roles are automatically assigned to the default user (`admin`):

- "Administrator" on page 177

Best practice: For a more secure environment, remove the default `admin` user. For details, see "Disabling the default admin user" on page 187.

Permissions

A permission describes the actions that a user can perform in watsonx.data. The permissions at the platform level can be categorized as follows:

- Platform administration
- Service instance
- User administration
- Vaults

Note: The first three categories are associated with the watsonx.data control plane.

The permissions are grouped into the following categories:

Platform administration

Permissions in this category enable a user to configure, customize, monitor, and manage the platform. The following table provides the details:

The category includes the following permissions:

Permission	Description	Actions
Administer platform	This permission offers the most comprehensive set of actions for managing and monitoring the platform. This permission allows users to have elevated privileges and can grant or revoke all permissions, including other administrative permissions.	See the actions listed in the following permissions: • Manage configurations • Manage platform health • Manage platform roles • Manage user profiles • Manage user groups • Manage service instances
Manage configurations	This permission allows users to customize the platform, integrate the platform with other applications, and enable connections to unsupported data sources. This permission allows users to can access the **Customizations** page, the **Configurations** page, and the **JDBC drivers** tab on the **Platform connections** page.	• Configure connection to SMTP server. • Configure alert forwarding. • Configure integration with IBM Guardium appliances. • Configure connections to Hadoop clusters. • Customize branding. • Enable and disable home page cards. • Enable and disable default support links. • Add and delete custom support links. • Enable and disable guided tours. • Import JDBC drivers
Manage platform health	This permission allows users to monitor, set quotas and alerts, manage workloads to maintain the health of the platform, and gather diagnostic data when problems occur. This permission allows users to access the **Monitoring** page and the **Diagnostics** page.	• Monitor workloads and resource use. • Stop any runtime environment. • View pod status, details, and logs • Restart pods • View platform quotas and service quotas • View event history and alerts • Set and edit platform resource quotas.

Permission	Description	Actions
		• Set and edit individual service resource quotas. • Create and run diagnostics jobs. • Delete diagnostics jobs.
View platform health	This permission allows users to monitor and workloads across the platform to gauge the health of the platform. This permission allows users to have read-only access to the **Monitoring** page.	• Monitor workloads and resource use. • View pod status, details, and logs • View platform quotas and service quotas • View event history and alerts

Service instances

A service instance is a specific deployment of watsonx.data. The following table provides the details about permissions that are related to service instance.

Some service instances have their own access controls.

The category includes the following permissions:

Permission	Description	Actions
Create service instances	This permission allows users to create service instances and storage volumes.	• Create service instances.
Manage service instances	This permission allows users to manage access to any service instance or delete any service instance from the **Instances** page.	• Create service instances. • View all service instances. • Add users to any service instance. • Assign an instance role to instance users. • Remove users from a service instance. • Delete any service instance.

User administration

Permissions in this category enable a user to manage users, groups, and roles. The permissions in this category apply to the platform. The following table provides the details:

Permission	Description	Actions
Manage platform roles	This permission allows users to modify platform roles or create custom roles. Roles determine the permissions that a user or user group has.	• Create platform roles

Permission	Description	Actions
	This permission allows users to access the **Roles** tab on the **Access control** page. This permission does not apply to service instances or assets, such as projects, catalogs, and deployment spaces.	• Edit platform roles • Delete platform roles
Manage user groups	This permission allows users to create and edit user groups. User groups make it easy to manage the roles (and permissions) of users with similar access requirements. This permission allows users to access the **User groups** tab on the **Access control** page.	• Create user groups • Edit user groups • Delete user groups • Assign roles to user groups • Remove roles from user groups
Manage users	This permission allows users to onboard users to the platform, edit user profiles, and assign platform roles to users. This permission allows users to access the **Users** tab on the **Access control** page.	• Add users • Edit user profiles • Assign roles to users • Remove roles from users • Remove users

Vaults

The category is created when you enable the vaults interface.

Note: The permissions are not associated with any roles by default.

Secrets are sensitive data, such as credentials or API keys. A vault is a secure place to store and manage secrets.

Users can add secrets to the internal vault or connect to an external vault to use existing secrets. By default, only the user who added the secret can use the secret.

The category includes the following permissions:

Permission	Description	Actions
Add vaults.	Users with this permission can connect to external vaults and add secrets from their connected vaults.	• Add a connection to external vaults. • Add secrets from their connected vaults.
Manage vaults and secrets.	Users with this permission can see a list of all of the external vaults that users connected to and the list of secrets in each vault. However, users with this permission cannot see detailed information about the vaults or access the secrets in the vaults. Users with this permission can remove secrets from any vault and remove connections to any external vault.	• View a list of all connected vaults • View a list of all secrets in each vault • Remove external vaults. • Remove secrets added from an external vault. • Delete secrets from the internal vault.

Permission	Description	Actions
Share secrets.	Users with this permission can give other users access to secrets that they add. Users with this permission cannot share secrets that are shared with them.	• Share owned secrets. • Revoke access to shared secrets.

Managing user groups

You can create user groups to simplify the process of managing large groups of users.

watsonx.data on Red Hat OpenShift

User groups make it easier to manage a large number of users with similar access requirements. If a member of the group leaves the company, you can remove the user from the group, rather than looking for all of the assets that the user has access to.

Permissions you need for this task
To manage user groups, you must have one of the following permissions:

- **Administer platform**
- **Manage user groups**

When you need to complete this task
You can complete this task any time you need to create, edit, or delete a user group.

About this task

You can create and edit groups from the **User groups** tab of the **Access control** page.

Important: By default, IBM watsonx.data includes the **All users** group. As the name suggests, all watsonx.data users are automatically included in this group. You cannot edit or delete this group.

The type of group that you can create depends on your environment:

	Not integrated with an LDAP server	Integrated with an LDAP server
Not integrated with the IAM Service	• "Creating a user group without an LDAP server" on page 182	• "Creating an assigned user group with an LDAP server" on page 183
Integrated with the IAM Service	• "Creating a user group without an LDAP server" on page 182	• "Creating an assigned user group with an LDAP server" on page 183 • "Creating a dynamic user group with an LDAP server" on page 184

Creating a user group without an LDAP server

If you have not connected to an identity provider, you can create a group by specifying the users that you want to include in the group.

To create a user group:

1. Log in to the web client as an administrator.
2. From the navigation menu, select **Administration** > **Access control**.
3. Open the **User groups** tab.
4. Click **New user group**.

5. Enter a name and a description for the role.
6. Specify the users to include in the group.

   You can select the existing platform users that you want to add to the group

   If you have the **Manage users** permission and you don't see the user that you want to add to the group, you can create a new user.
7. Click **Next**.
8. Select the one or more roles that you want to assign to this group.

   If you have the **Manage platform roles** permission and you don't see a role that meets your needs, you can create a new role.
9. Click **Next**.
10. Review the summary. If the values are correct, click **Create**.

Creating an assigned user group with an LDAP server

In an assigned user group, you must specify the platform users, LDAP users, and LDAP groups that belong to the user group.

To create an assigned user group:

1. Log in to the web client as an administrator.
2. From the navigation menu, select **Administration** > **Access control**.
3. Open the **User groups** tab.
4. Click **New user group**.
5. Enter a name and a description for the role.
6. If you integrated with the IAM Service, select **Assigned**. (If you are not integrated with the IAM Service, the group is automatically an assigned group.)
7. Click **Next**.
8. Specify the users to include in the group.

   The available options depend on whether your LDAP server has LDAP groups.

Groups	Instructions
LDAP does not have groups	If LDAP is configured, you can select one or more of the following types of users: **Existing platform users** If you want to add existing platform users to the group, click **Existing users** and select the users that you want to add. **LDAP users** If you want to add users from the LDAP server, click **Identity provider users** and search for the users that you want to add.
LDAP is configured with groups	If LDAP is configured with groups, you can select one or more of the following types of users: **Existing platform users** If you want to add existing platform users to the group, click **Existing users** and select the users that you want to add. **LDAP users** If you want to add users from the LDAP server, click **Identity provider users** and search for the users that you want to add.

Groups	Instructions
	LDAP groups If you want to add all of the users in an LDAP group to the user group, click **Identity provider groups** and search for the group that you want to add. If you add users from an LDAP group, the users aren't immediately added to the watsonx.data user group. When a user logs in to watsonx.data, the platform determines whether the user is a member of an LDAP group. If the user does not have a profile, the platform creates a user profile and adds the user to the watsonx.data user group.

9. Click **Next**.
10. Select the one or more roles that you want to assign to this group.

    If you have the **Manage platform roles** permission and you don't see a role that meets your needs, you can create a new role.
11. Click **Next**.
12. Review the summary. If the values are correct, click **Create**.

Creating a dynamic user group with an LDAP server

Restriction: This option is available only if you integrate with the IAM Service.

In a dynamic user group, you can create attribute-based rules to determine which users are included in the group. You can use the following attributes to define dynamic user groups:

- Location
- Nationality
- Organization
- User type

Users are automatically added or removed from the user group based on the attributes that are assigned to them on the identity provider. For example, you create a user group for people managers (user type) in the finance group (organization) in Canada (location). If Annette is hired as a people manager for the finance group in Canada, she will automatically become a member of the group. Similarly, if Rajesh is transferred to Spain, he will automatically be removed from the group.

To create a dynamic user group:

1. Log in to the web client as an administrator.
2. From the navigation menu, select **Administration** > **Access control**.
3. Open the **User groups** tab.
4. Click **New user group**.
5. Enter a name and a description for the role.
6. Select **Dynamic**.
7. Click **Next**.
8. Define the membership rule for the group:
   a. Specify how the conditions are enforced:
      - Choose **All conditions (AND)** to include users only if all of the conditions are met.
      - Choose **Any condition (OR)** to include users if at least one condition is met.

   b. Specify one or more conditions by specifying:
      - An attribute: **Location**, **Nationality**, **Organization**, or **User type**

    - An operator: **Equal**, **Not equal**, **Match**, or **Not match**
    - The value for the condition.
9. Click **Next**.
10. Select the one or more roles that you want to assign to this group.

    If you have the **Manage platform roles** permission and you don't see a role that meets your needs, you can create a new role.
11. Click **Next**.
12. Review the summary. If the values are correct, click **Create**.

When a user logs in to the platform, the user is automatically added or removed from the group based on the attributes that are assigned to them on the LDAP server.

Managing access to the platform

When you add a user to the platform, a user profile (or record) is created for the user.

watsonx.data on Red Hat OpenShift

You can add users to the platform in the following ways:

- You can give individual users access to the platform by manually creating a user profile.
- You can give individual LDAP users access to the platform by adding them to a user group. When you add an LDAP user to a user group, the platform automatically creates a profile for the LDAP user.
- You can give all of the members of an LDAP group access to the platform by adding the LDAP group to a user group. When you add the LDAP group to a user group, the platform automatically creates a profile for each LDAP user in the group. (The platform skips this step for any members of the group who already have a user profile on the platform.)

Permissions you need for this task

To manage access to the platform, you must have one of the following permissions:

- **Administer platform**
- **Manage users**

When you need to complete this task

You can complete this task any time you need to onboard users to the platform, remove users from the platform, edit user profiles, and assign platform roles to users.

About this task

You can create and edit user profiles from the **Users** tab of the **Access control** page.

Procedure

To give users access to the web client:

1. Log in to the web client as an administrator.
2. From the navigation menu, select **Administration** > **Access control**.
3. Open the **Users** tab.
4. Click **Add user**.
5. Specify the appropriate information for your environment:

Environment	Information to specify
Connected to an LDAP server	Search for the user that you want to add. The user's information is retrieved from the LDAP server.

Environment	Information to specify
Connected to a SAML identity provider	• The user's full name • The username that the user will authenticate with The appropriate value depends on the attribute that you specified for the **fieldToAuthenticate** in the SAML SSO configuration. • The user's email address
Not connected to an LDAP server	• The user's full name • The username that the user will authenticate with • The user's email address • A temporary password for the user

6. Click **Next**.
7. Specify how you want the user to get their permissions:

Choices	Instructions
Assign roles to the user	a. Click **Assign roles directly**. b. Select the roles the appropriate roles to assign to the user. If you have the **Manage platform roles** permission, you can optionally create a new role for the user. c. Click **Next**.
Add the user to a user group	a. Click **Add to user group**. b. Select the groups that you want to add the user to. The user inherits the permissions from each group they are added to. c. Click **Next**.

8. Review the summary and click **Add**.
9. If you are not connected to an LDAP server or SAML IDP, copy the temporary password that you specified and send an email to the user with their username and temporary password.

Managing access to the Hive Metastore (HMS)

You can give individual users access to the Hive Metastore (HMS) by assigning the `Metastore Admin` role to the users.

watsonx.data on Red Hat OpenShift

Before you begin

Ensure to have at least one `Admin` user to be able to update the role of an `Admin` user. For more information, see “Managing access to the platform” on page 185.

Procedure

To give users access to HMS, complete the following steps:

1. Log in to the web client as an administrator.
2. From the navigation menu, select **Instances**.
3. On the Instances page, locate the **lakehouse** instance.
4. From the action menu, select **Manage access**.
5. To add a user with `Metastore Admin` role:

    a. Click **Add users**.
    b. From the list of users, select the user and grant the `Metastore Admin` role.
    c. Click **Add**.
6. To change the role of a `User` or `Admin` to a `Metastore Admin`:
    a. From the **Access Management: lakehouse** page, select the user with `Admin` or `User` role.
    b. Click the **Service role** menu and change the role to `Metastore Admin`.

Monitoring user activity

An IBM watsonx.data administrator can check whether a user is online or offline and see information about the user's current session and previous session. You can use this information to identify suspicious activity and block the user from accessing the web client while the activity is investigated.

watsonx.data on Red Hat OpenShift

Best practice: In addition to monitoring user activity, you should also forward audit logs to an external security information and event management (SIEM) solution.

Who needs to complete this task?
To monitor user activity, you must have one of the following permissions:

- **Administer platform**
- **Manage users**

When do you need to complete this task?
Complete this task as necessary to maintain the security of your watsonx.data environment.

Procedure

To monitor user activity:

1. Log in to the web client as an administrator.
2. From the menu, select **Administration** > **Access control**.
3. Open the **Users** tab.
4. Select the user for whom you want to see their activity.
5. Open the **Activity** tab.

   From the **Activity** tab you can see:

   - Whether the user is currently online or offline.
   - If the user is online, you can see the user's current sessions and the duration of their current sessions.
   - The user's previous session and the duration of the previous session.

   Use this information to identify suspicious activity. For example, the user is misusing their access or the user's account has been compromised.
6. Optional: You can prevent the user from logging in to web client while you investigate the activity. To prevent the user from logging in, click **Block**. Then, confirm that you want to block the user.

   If you determine that the activity was justified or if the user's password is updated to prevent unauthorized log in, you can unblock the user from the **Activity** tab.

Disabling the default admin user

If you are using an enterprise-grade LDAP server for user management, you can further secure your IBM watsonx.data system by disabling the default **admin** user.

watsonx.data on Red Hat OpenShift

Before you begin

Best practices: You can run the commands in this task exactly as written if you set up environment variables. For instructions, see Setting up installation environment variables.

Ensure that you source the environment variables before you run the commands in this task.

Procedure

To disable the default **admin** user:

1. Log in to your Red Hat OpenShift cluster as a project administrator:

   ```
   oc login ${OCP_URL}
   ```

2. Run the following command:

   ```
   oc exec -it -n ${PROJECT_CPD_INST_OPERANDS} \
   $(oc get pod -n PROJECT_CPD_INST_OPERANDS -l component=usermgmt | tail -1 | cut -f1 -d\ ) \
   -- bash -c "/usr/src/server-src/scripts/manage-user.sh --disable-user admin"
   ```

What to do next

If you encounter a problem and cannot log in to the web client with any of your LDAP user names, you can re-enable the admin user. For example, you might need to do this if there is a connectivity issue with your LDAP server or SAML IDP server.

To re-enable the default **admin** user:

1. Log in to your Red Hat OpenShift cluster as a project administrator:

   ```
   oc login ${OCP_URL}
   ```

2. Run the following command:

   ```
   oc exec -it -n ${PROJECT_CPD_INST_OPERANDS} \
   $(oc get pod -n ${PROJECT_CPD_INST_OPERANDS} -l component=usermgmt | tail -1 | cut -f1 -d\ )
   \
   -- bash -c "/usr/src/server-src/scripts/manage-user.sh --enable-user admin"
   ```

3. When prompted, specify a new password for the **admin** user.

Connecting to external object stores over `https`: Cloud Pak for Data

IBM watsonx.data needs a valid signer certificate to establish a connection with the object stores secured with **`https`**.

watsonx.data on Red Hat OpenShift

Procedure

1. Import the certificate from the object store server.

   ```
   export STORE_ENDPOINT=<endpoint>:<port number>
   export CERT=$(echo QUIT | openssl s_client -showcerts -connect $STORE_ENDPOINT | awk '/-----
   BEGIN CERTIFICATE-----/ {p=1}; p; /-----END CERTIFICATE-----/ {p=0}' | awk '{printf "%s\\n",
   $0}')
   ```

2. Patch the watsonx.data service instance with the new certificate.

   Warning: Updating the watsonx.data truststore is a disruptive action. Updating the truststore causes the watsonx.data pods, including engines performing workloads to restart. To minimize the impact, it is recommended to wait for any long-running workloads to complete before updating the truststore.

```
oc patch wxd/lakehouse --type=merge -n ${PROJECT_CPD_INST_OPERANDS} -p "{ \"spec\": {
          \"update_ca_certs\": true,
          \"extra_ca_certs_secret\": \"$CERT\"
      } }"
```

3. Wait for the pods to restart with the updated truststore.

   Note: If you restart the pods in `ibm-cert-manager` or the entire cluster, patch the watsonx.data service instance with the new certificate again and wait for the pods to restart with the updated truststore.

What to do next

For adding storages, see Adding a storage-catalog pair.

Enhancing the query performance through caching

You can configure different layers of caching in watsonx.data to optimize the query performance.

About this task

This procedure helps you to add (enable), edit, and delete different layers of caching that watsonx.data supports.

- **Data cache:** It reduces the need to repeatedly fetch the same data from the storage layer for running similar queries and the overall query processing time.
- **Fragment result cache:** It stores intermediate results of queries to speed up the repeated or similar queries.
- **Metastore versioned cache:** It stores the metadata about tables, like structure and schema, to reduce the need for repeated metadata lookups.
- **File list cache:** It keeps a list of files for a specified table and partition to avoid querying the storage system repeatedly for this information.
- **File and stripe footer cache:** It stores the footers of files and stripes in ORC/Parquet files, reducing the I/O for reading file metadata.

Metastore versioned cache and **File and stripe footer cache** are enabled by default with an option to update any of the properties listed in the following procedure:

Note: Data cache and Fragment result cache are not supported if multiple presto pods are running on a single node.

Procedure

1. Set up the caching on a **`wxdengine`** instance.

   a) Set the namespace in the console.

   ```
   oc project ${PROJECT_CPD_INST_OPERANDS}
   ```

   b) Determine which Presto engine you want to update.

   ```
   oc get wxdengine -o custom-columns='DISPLAY NAME:spec.engineDisplayName,ENGINE
   ID:metadata.labels.engineName'
   ```

   Example:

```
oc get wxdengine -o custom-columns='DISPLAY NAME:spec.engineDisplayName,ENGINE
ID:metadata.labels.engineName'
  DISPLAY NAME   ENGINE ID
  presto         presto750
  presto-01      presto-01
```

c) Add the cache configuration under the **spec** section of engine configuration.

```
oc patch wxdengine/<engine-name> \
  --type=merge \
  -n ${PROJECT_CPD_INST_OPERANDS} \
  -p '{ "spec": { "<property1>": "<value1>", "<property2>": "<value2>" } }'
```

Example:

```
oc patch wxdengine/lakehouse-presto-01 \
 --type=merge \
 -n ${PROJECT_CPD_INST_OPERANDS} \
  -p '{ "spec": { "hive_metastore_cache_partition_versioning_enabled": "true",
"hive_metastore_cache_scope": "PARTITION", "hive_metastore_cache_ttl": "3d",
"hive_metastore_cache_refresh_interval": 4d, "hive_metastore_cache_maximum_size":
"20000000" } }'
```

Note: Add all the configurations that you want to change in **json** format.

Customizable configurations for different levels of cache

Data cache

```
cacheStorageClass: <nfs-client>
cacheStorageSize: <10Gi>
cache_alluxio_max_cache_size: <8GB>
```

- **cacheStorageSize** is the name of the **StorageClass**.
- **cacheStorageSize** is the size of storage that is required for the cache mount.
- **cache_alluxio_max_cache_size** is the maximum size of the cache that is allowed for data cache. This must be less than or equal to **cacheStorageSize**.

Fragment result cache

```
fragment_result_cache_enabled: true
fragment_result_cache_max_cached_entries: 1000000
fragment_result_cache_ttl: 36h
fragment_result_cache_partition_statistics_based_optimization_enabled: true
fragmentCacheStorageClass: <your storage class>
fragmentCacheStorageSize: <10Gi>
```

- The properties **fragment_result_cache_max_cached_entries**, **fragment_result_cache_ttl**, and **fragment_result_cache_partition_statistics_based_optimization_enabled** are optional.
- The default value for **fragment_result_cache_max_cached_entries** is 1000000.
- The default value for **fragment_result_cache_ttl** is 36 h.
- The default value for `fragment_result_cache_partition_statistics_based_optimization_enabled` is true.

Metastore versioned cache

You can add/update the following properties into the engine configuration by using the `oc patch` command.

```
hive_metastore_cache_partition_versioning_enabled: false
hive_metastore_cache_scope: ALL
hive_metastore_cache_ttl: 10m
hive_metastore_cache_maximum_size: 10000000
```

```
hive_metastore_cache_timeout: 3m
hive_metastore_cache_refresh_interval: 2m
```

- The property `hive_metastore_cache_partition_versioning_enabled` is disabled by default. All the other properties except `hive_metastore_cache_maximum_size` are defaulted with the values listed here.

File list cache

```
file_status_cache_expire_time: 24h
file_status_cache_tables: "*"
file_status_cache_size: 100000000
```

- If you add any of the three properties, file list caching is added with the default values of the other two properties that are specified in previous steps.

File and stripe footer cache

- **For ORC or DWRF:**

  The default values are:

  ```
  file_metadata_orc_file_tail_cache_enabled: true
  file_metadata_orc_file_tail_cache_size: Calculated based on jvm.Xmx
  file_metadata_orc_file_tail_cache_ttl_since_last_access: 6h
  file_metadata_orc_stripe_metadata_cache_enabled: true
  file_metadata_orc_stripe_footer_cache_size: Calculated based on jvm.Xmx
  file_metadata_orc_stripe_footer_cache_ttl_since_last_access: 6h
  file_metadata_orc_stripe_stream_cache_size: Calculated based on jvm.Xmx
  file_metadata_orc_stripe_stream_cache_ttl_since_last_access: 6h
  file_metadata_orc_use_column_names: true
  ```

  You can add/update the following properties into the engine configuration by using the **`oc patch`** command.

- The properties **`file_metadata_orc_file_tail_cache_enabled`** and **`file_metadata_orc_stripe_metadata_cache_enabled`** are set as **`true`** by default. The default values for the other properties are listed under the default values.

- **For Parquet:**

  The default values are:

  ```
  file_metadata_parquet_metadata_cache_enabled: true
  file_metadata_parquet_metadata_cache_size: Calculated based on jvm.Xmx
  file_metadata_parquet_metadata_cache_ttl_since_last_access: 6h
  file_metadata_parquet_metadata_batch_read_optimization_enabled: true
  file_metadata_parquet_metadata_use_column_names: true
  ```

  You can add/update the following properties into the engine configuration by using the **`oc patch`** command.

  The property **`file_metadata_parquet_metadata_cache_enabled`** is set as **`true`** by default. The default values for the other properties are listed under the default values.

If you want to update the properties for Metastore versioned cache, File list cache, and File and stripe footer cache without deleting the stateful sets, then update the `cr.yaml` file and run the following command to track the status of operator reconcile and the config maps.

```
 watch --color "oc get wxdEngine -n $
{PROJECT_CPD_INST_OPERANDS} -o custom-columns='NAME:metadata.name,DISPLAY
NAME:spec.engineDisplayName,RECONCILE:status.engineStatus,STATUS:status.middleEndStatus'"
```

The `RECONCILE` status first appears as `RUNNING` and then changes to `COMPLETED`. Restart all Presto pods to apply the new configuration.

2. **Optional: Set up Persistent Volume (PV).**

   This step is for setting storage classes for data cache and fragment result cache. You need two separate PVs and storage classes for data cache and fragment cache. Use the **`localStorageProvisioner`**, and complete the following steps:

Note: This step is optional.

a) Create a **`yaml`** file.

```
vi pv1.yaml
```

b) Copy the content to **`pv1.yaml`** file and save it.

Note: The values for **`name`**, **`storage`**, **`storageClassName`**, **`path`**, and **`nodeAffinity`** are based on the client requirement.

For example, in `name: presto-cache-pv1` in the following output, **`presto-cache-pv1`** is customizable.

You must set a path, which has sufficient storage.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: presto-cache-pv1
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: cache-storage
  local:
    path: /tmp/cache
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - worker0.bicorn.cp.fyre.ibm.com
          - worker1.bicorn.cp.fyre.ibm.com
```

c) To get the values field under **`nodeAffinity`**, run **`oc get nodes`** and use the name of the worker nodes that have the disk space available to mount the cache of Presto pods.

d) Access the debugging session on the selected worker node and create the necessary directory structure.

```
oc debug node/<name of node> -- chroot /host mkdir -p <path used under local, /tmp/cache>
```

Note: Repeat steps **c** and **d** for all the selected nodes.

e) Run the following command to apply the PV configurations.

```
oc apply -f pv1.yaml
```

f) Provision more PVs based on the t-shirt sizing. If you have three Presto pods, create **`pv2.yaml`**, and **`pv3.yaml`**. For **pv2**, use name as **`presto-cache-pv2`** and path as **`path /tmp/cache/pv2`**. For **pv3**, use name **`presto-cache-pv3`** and path as **`path /tmp/cache/pv3`**.

g) Create two or more PVs based on the requirements.

```
oc apply -f pv2.yaml pv3.yaml
```

3. Delete the caching.

a) Determine the Presto instance from which you want to remove the cache.

```
oc get wxdengine -o custom-columns='DISPLAY NAME:spec.engineDisplayName,ENGINE
ID:metadata.labels.engineName'
```

Example:

```
oc get wxdengine -o custom-columns='DISPLAY NAME:spec.engineDisplayName,ENGINE
ID:metadata.labels.engineName'
  DISPLAY NAME   ENGINE ID
  presto         presto750
  presto-01      presto-01
```

b) Remove the configurations of the cache that you want to delete.

```
oc patch wxdengine/<engine-name> -n ${PROJECT_CPD_INST_OPERANDS} --type='json'
-p='[{"op": "remove", "path": "/spec/property1"}, {"op": "remove", "path": "/spec/
property2"}]'
```

c) Delete the Persistent Volume Claims (PVC) of the cache.

For data cache

i) List the PVCs of the data cache.

```
$ oc get pvc | grep ibm-lh-cache-mount
```

ii) Delete the PVCs.

```
$ oc delete pvc <PVC's name>
```

For fragment-result cache

i) List the PVCs of the fragment-result cache.

```
$ oc get pvc | grep ibm-lh-fragment-cache
```

ii) Delete the PVCs.

```
$ oc delete pvc <PVC's name>
```

d) Delete the PVs if you have completed step 2.

```
oc delete pv <pv name>
```

Reference

The following is the complete list of available settings:

Data cache

- `cacheStorageClass`
- `cacheStorageSize`
- `cache_alluxio_max_cache_size`

Fragment result cache

- `fragment_result_cache_enabled`
- `fragment_result_cache_max_cached_entries`
- `fragment_result_cache_ttl`
- `fragment_result_cache_partition_statistics_based_optimization_enabled`
- `fragmentCacheStorageClass and fragmentCacheStorageSize`

Metastore versioned cache

- `hive_metastore_cache_partition_versioning_enabled`
- `hive_metastore_cache_scope`
- `hive_metastore_cache_ttl`
- `hive_metastore_cache_refresh_interval`
- `hive_metastore_cache_maximum_size`
- `hive_metastore_cache_timeout`

File list cache

- `file_status_cache_expire_time`
- `file_status_cache_tables`
- `file_status_cache_size`

File and stripe footer cache

- `file_metadata_parquet_metadata_cache_enabled`
- `file_metadata_parquet_metadata_cache_size`
- `file_metadata_parquet_metadata_cache_ttl_since_last_access`
- `file_metadata_parquet_metadata_batch_read_optimization_enabled`
- `file_metadata_parquet_metadata_use_column_names`
- `file_metadata_orc_file_tail_cache_enabled`
- `file_metadata_orc_file_tail_cache_size`
- `file_metadata_orc_file_tail_cache_ttl_since_last_access`
- `file_metadata_orc_stripe_metadata_cache_enabled`
- `file_metadata_orc_stripe_footer_cache_size`
- `file_metadata_orc_stripe_footer_cache_ttl_since_last_access`
- `file_metadata_orc_stripe_stream_cache_size`
- `file_metadata_orc_stripe_stream_cache_ttl_since_last_access`
- `file_metadata_orc_use_column_names`

Configuring a local staging directory

You can create a local staging directory to precisely control the temporary storage. With a local staging directory, you can optimize the use of resources during data operations.

Local Staging Directory (hive.s3.staging-directory) is primarily used to avoid pod crash filling up when you run large operations (like CTAS operations) in the Query workspace. It is used to specify a temporary local storage for the hive. During such large operations, intermediate result set is stored in hive.s3.staging-directory before it is written or read out to S3. By default, hive.s3.staging-directory is set to `java.io.tmpdir` that is `/tmp` mounted on `/filesystem` in the worker pods. During large operations, it can fill up and this might lead to a pod restart. So, configuring local staging directory to appropriate storage is important.

Procedure

1. Create a PV. Use the `localStorageProvisioner` and complete the following steps.

   Note: This is an optional step for setting a storage class for the local staging directory.

   a) Create a `yaml` file.

   ```
   vi pv1.yaml
   ```

   b) Copy the following content to `yaml` file and save the file.

   Note: The values for name, storage, `storageClassName`, path, and `nodeAffinity` are based on the client requirement. For example, in `name: presto-staging-storage-pv1` in the following configuration format, `presto-staging-storage-pv1` is customizable. The path must be same in all the PVs.

   ```
   apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: presto-staging-storage-pv1
   spec:
     capacity:
   ```

```
      storage: 100Gi
    volumeMode: Filesystem
    accessModes:
    - ReadWriteOnce
    persistentVolumeReclaimPolicy: Delete
    storageClassName: staging-storage
    local:
      path: /dev/stagingStorage
    nodeAffinity:
      required:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
            - worker0.o1-380113.cp.fyre.ibm.com
            - worker1.o1-380113.cp.fyre.ibm.com
            - worker2.o1-380113.cp.fyre.ibm.com
```

c) To get the values field under `nodeAffinity`, run the following command. Use the name of the worker nodes that have the disk space available to mount the staging directory of Presto pods.

```
oc get nodes
```

d) Access the debugging session on the chosen worker node and create the necessary directory structure.

```
oc debug node/<name of node> -- chroot /host mkdir -p <path used under local, /dev/
stagingStorage>
```

Repeat the steps **c** and **d** for all the selected nodes.

e) Run the following command to apply the PV configurations.

```
oc apply -f pv1.yaml
```

f) Provision more PVs based on the t-shirt sizing. If there are three Presto pods, create `pv2.yaml`, and `pv3.yaml` For PV2, use the name as `presto-staging-storage-pv2`. For PV3, use name `presto-staging-storage-pv3`.

g) Create two or more PVs based on the requirements.

```
oc apply -f pv2.yaml pv3.yaml
```

2. Set up local staging directory.

a) Set the namespace in the console.

```
oc project ${PROJECT_CPD_INST_OPERANDS}
```

b) Determine which Presto engine you want to update.

```
oc get wxdengine -o custom-columns='DISPLAY NAME:spec.engineDisplayName,ENGINE
ID:metadata.labels.engineName'
```

Example:

```
oc get wxdengine -o custom-columns='DISPLAY NAME:spec.engineDisplayName,ENGINE
ID:metadata.labels.engineName'
  DISPLAY NAME   ENGINE ID
  presto         presto750
  presto-01      presto-01
```

c) Add the cache configuration under the spec section of engine configuration.

```
hive_s3_staging_directory_enabled: true
s3StagingStorageClass: staging-storage
s3StagingStorageSize: 2Gi
```

Use `oc patch` tool to add the properties into the spec:

```
oc patch wxdengine/lakehouse-presto-01 \
 --type=merge \
 -n cpd-instance \
  -p '{ "spec": { "hive_s3_staging_directory_enabled": "true",   "s3StagingStorageClass":
"staging-storage", "s3StagingStorageSize": "2Gi" } }'
```

To enable a local staging directory, set **`hive_s3_staging_directory_enabled`** as **`true`**.

The default values for **`s3StagingStorageClass`** and **`s3StagingStorageSize`** are as specified.

Ensure that the **`storageClassName`** matches with the **`s3StagingStorageClass`**.

3. Remove the local staging directory.

   a) Determine the Presto instance from which you want to remove the local staging directory.

   ```
   oc get wxdengine -o custom-columns='DISPLAY NAME:spec.engineDisplayName,ENGINE
   ID:metadata.labels.engineName'
   ```

   Example:

   ```
   oc get wxdengine -o custom-columns='DISPLAY NAME:spec.engineDisplayName,ENGINE
   ID:metadata.labels.engineName'
     DISPLAY NAME   ENGINE ID
     presto         presto750
     presto-01      presto-01
   ```

   b) Remove the local staging directory configurations.

   ```
   oc patch wxdengine/<engine_name> --type='json' -p='[{"op": "remove",
   "path": "/spec/hive_s3_staging_directory_enabled"}, {"op": "remove", "path": "/spec/
   s3StagingStorageSize"}, {"op": "remove", "path": "/spec/s3StagingStorageClass"}]'
   ```

Accessing Hive Metastore (HMS) using NodePort

You can access the watsonx.data Hive Metastore (HMS) service from outside the OpenShift Container Platform cluster by exposing the HMS using a `NodePort` service.

Important: In watsonx.data, HMS is not exposed by default. It must be exposed using a NodePort service to facilitate connection from external applications (outside the watsonx.data namespace), such as Spark to watsonx.data. Not exposing HMS will result in Thrift connection failure and thereby failure to connect with watsonx.data.

watsonx.data on Red Hat OpenShift

About this task

You can connect to Hive Metastore (HMS) service from inside the cluster by using the internal service address `thrift://ibm-lh-lakehouse-hive-metastore-svc.<namespace>.svc.cluster.local:9083`. However to access the Hive Metastore (HMS) externally from outside the cluster, you need to expose the HMS using a over a `NodePort`.

Procedure

1. Create a `NodePort` service.

   a) Update the watsonx.data instance to create a `NodePort` for Hive Metastore. Run the following commands:

   ```
   export PROJECT_CPD_INST_OPERANDS=<INSTANCE_NAMESPACE>
   ```

   ```
   oc patch wxd/lakehouse \
     --type=merge \
      -n ${PROJECT_CPD_INST_OPERANDS} \
      -p '{ "spec": { "expose_hive_metastore": true } }'
   ```

b) Get the NodePort value that is created. Run the following command:

```
oc get svc ibm-lh-lakehouse-hive-metastore-nodeport
NAME                                        TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)                      AGE
ibm-lh-lakehouse-hive-metastore-nodeport    NodePort    172.21.40.106   <none>
9083:32519/TCP               11h
```

2. Configure the NodePort with an Ingress controller.

   If you use an external infrastructure node to route external traffic into the Red Hat OpenShift cluster, the cluster might be in a private zone and you need to configure an external-facing Ingress controller to route the traffic to the OpenShift nodes.

   Because Hive Metastore is externally exposed through a NodePort, the Ingress Controller also needs to expose the NodePort in order to allow traffic into the cluster.

   The configuration below is only applicable with an HAProxy Ingress Controller. For more detail about configuring networking, see Understanding networking in the OpenShift documentation.

   a) On the infrastructure node, open the HAProxy configuration file located at /etc/haproxy/haproxy.cfg.

   b) Modify the haproxy.cfg file to include the OpenShift NodePort you want to expose:

```
frontend hms-nodeport
        bind *:<node-port>
        default_backend hms-nodeport
        mode tcp
        option tcplog
backend hms-nodeport
        balance source
        mode tcp
        server worker0 <worker-0-private-ip>:<node-port> check
        server worker1 <worker-1-private-ip>:<node-port> check
        server worker2 <worker-2-private-ip>:<node-port> check
```

   c) Reload HAProxy:

```
systemctl reload haproxy
```

3. Update the SSL certificate SANs list with the external domain of the NodePort.

   To establish a trusted SSL connection with Hive Metastore, the SSL certificate's Subject Alternative Names list (SAN) must contain the same domain that you connect with externally. For example, if you want to connect to Hive Metastore using the uri: www.my.cluster.domain.com:60000 then the SANs list must include www.my.cluster.domain.com for a successful connection.

   Warning: Updating the SSL certificate SANs list is a disruptive action. It causes the watsonx.data pods, including engines performing workloads, to restart. To minimize the impact, it is recommended to wait for any long-running workloads to complete before proceeding.

   a) Edit the SSL certificate.

```
oc edit cert/ibm-lh-tls-certificate -n ${PROJECT_CPD_INST_OPERANDS}
```

   b) Add the external domain to the allowed dnsNames.

```
spec:
  commonName: ibm-lh-tls-secret
  dnsNames:
  - '*.zen'
  - '*.zen.svc'
  - '*.zen.svc.cluster.local'
  - '*.svc.cluster.local'
  - localhost
  - ibm-lh-presto-svc
  - <infranode-domain>
```

   c) Save the file, and wait for the certificate to be READY.

```
oc get cert/ibm-lh-tls-certificate -n ${PROJECT_CPD_INST_OPERANDS}
NAME                       READY   SECRET              AGE   EXPIRATION
ibm-lh-tls-certificate     True    ibm-lh-tls-secret   22h   2033-07-23T23:33:06Z
```

Importing self-signed certificates from a Presto (Java) server to a Java truststore

Client applications such as IDEs and utilities such as Presto (Java) CLI must trust the Presto engine if it is configured to use a self-signed certificate. The software installations use self-signed certificates as default setting. The following procedure gives instructions to import a self-signed certificate into a truststore for use by Presto (Java) CLI or other client utilities.

These instructions apply when the clients need to connect to Presto (Java) engines in IBM watsonx.data on Red Hat OpenShift, watsonx.data on Cloud Pak for Data, and watsonx.data Developer Edition.

These instructions do not apply when you connect to the local Presto (Java) engine in the watsonx.data Developer edition with `bin/presto-cli` utility. The `bin/presto-cli` utility has a truststore that is configured by default.

watsonx.data on Red Hat OpenShift

About this task

To import the certificate into your truststore, complete the following steps:

Procedure

1. On a client workstation from where you intend to connect to the Presto (Java) server, get the certificate served by the Presto (Java) server.

   ```
   echo QUIT | openssl s_client -showcerts -connect <presto-engine-host>:<port> | awk '/-----
   BEGIN CERTIFICATE-----/ {p=1}; p; /-----END CERTIFICATE-----/ {p=0}' > presto.cert
   ```

2. Use one of the following methods to add the certificate to the truststore:

   a) Add to your existing Java truststore.

   ```
   keytool -import -trustcacerts -storepass changeit -noprompt -alias presto-cert -file ./
   presto.cert
   ```

   b) Create a new Java truststore.

   ```
   keytool -import -alias presto-cert -file ./presto.cert -keystore ./presto-truststore.jks
   ```

3. Check whether the certificate is imported correctly.

   a) If you added the certificate to existing Java truststore, run:

   ```
   keytool -list -v -storepass changeit -alias presto-cert
   ```

   Note: `changeit` is the default password for the `cacerts` in Java.

   b) If you created a new Java truststore, run:

   ```
   keytool -list -v -keystore ./presto-truststore.jks -alias presto-cert
   ```

4. Install the certificate by using Presto (Java) CLI.

   a) If the certificate is in the existing truststore:

   ```
   export PRESTO_PASSWORD=<your password>; Presto (Java) --password --server https://cpd-lh-
   bart-01.fyre.ibm.com:8443 --user <your username> --catalog "tpch" --execute "select *
   from tiny.customer limit 10;"
   ```

   b) If the certificate is in the new truststore:

```
export PRESTO_PASSWORD=<your password>; ./presto --truststore-path ./presto-
truststore.jks --truststore-password=test123456 --password --server https://cpd-lh-
bart-01.fyre.ibm.com:8443 --user ibmlhadmin --catalog "tpch" --execute "select * from
tiny.customer limit 10;"
```

Note: If the `PRESTO_PASSWORD` environment variable is used, Presto (Java) CLI does not prompt for password.

Note: Provide the `--truststore-path` and `--truststore-password` arguments on the Presto (Java) command line.

Importing self-signed certificates from a Hive metastore server to a Java truststore

Client applications such as IDEs and utilities must trust the Hive metastore (HMS) if it is configured to use a self-signed certificate. The software installations use self-signed certificates as default setting.

watsonx.data on Red Hat OpenShift

About this task

The following procedure gives instructions to import a self-signed certificate into a truststore for use by the client utilities.

Procedure

To import the certificate into your truststore, complete the following steps:

1. On a client workstation from where you intend to connect to HMS, get the certificate served by thrift server.

   ```
   echo QUIT | openssl s_client -showcerts -connect <hive-metastore-host>:<nodeport> | awk
   '/-----BEGIN CERTIFICATE-----/ {p=1}; p; /-----END CERTIFICATE-----/ {p=0}' > hms.cert
   ```

2. Use one of the following methods to add certificate to truststore:

   a) Add to your existing Java truststore.

   ```
   keytool -import -trustcacerts -cacerts -storepass changeit -noprompt -alias hms-cert
   -file ./hms.cert
   ```

   b) Create a Java truststore.

   ```
   keytool -import -alias hms-cert -file ./hms.cert -keystore ./hms-truststore.jks
   ```

3. Check whether the certificate is imported correctly.

   a) If you added the certificate to existing Java truststore, run the following command:

   ```
   keytool -list -v -cacerts -storepass changeit -alias hms-cert
   ```

   Note: `changeit` is the default password for the `cacerts` in Java.

   b) If you created a new Java truststore, run the following command:

   ```
   keytool -list -v -keystore ./hms-truststore.jks -alias hms-cert
   ```

Accessing the MinIO console

For exploratory purposes, you can access the sample storage that comes with watsonx.data through the user interface of MinIO. An edge route is the recommended way to expose the MinIO console.

The out-of-the-box MinIO object storage is provided for exploratory purposes only. It does not have all security features and is not configured to provide high-speed data access. You should register your own s3 storage that meet your security and performance requirements.

About this task

An edge route is a secure way of exposing a service through the Red Hat OpenShift router. You can access the MinIO console from outside the cluster through `https` by using an edge route.

To access the storages through the user interface of MinIO, complete the following steps:

Procedure

1. Copy the S3 credentials from the secret and save it.

   ```
   oc extract secret/ibm-lh-config-secret -n ${PROJECT_CPD_INST_OPERANDS} --to=- --
   keys=env.properties | grep -E "LH_S3_ACCESS_KEY|LH_S3_SECRET_KEY"
   ```

   Example:

   ```
   oc extract secret/ibm-lh-config-secret -n ${PROJECT_CPD_INST_OPERANDS} --to=- --
   keys=env.properties | grep -E "LH_S3_ACCESS_KEY|LH_S3_SECRET_KEY"
   # env.properties
   LH_S3_ACCESS_KEY="mCD2fnzeabC58ipdc7E2a299"
   LH_S3_SECRET_KEY="faE4dnelxi662v31wbsutybz"
   ```

2. Create an edge route.

   ```
   oc create route edge ibm-lh-lakehouse-minio-console --service=ibm-lh-lakehouse-minio-svc --
   port=9001 -n ${PROJECT_CPD_INST_OPERANDS}
   oc get route ibm-lh-lakehouse-minio-console -n ${PROJECT_CPD_INST_OPERANDS}
   ```

   Example:

   ```
   oc create route edge ibm-lh-lakehouse-minio-console --service=ibm-lh-lakehouse-minio-svc --
   port=9001 -n ${PROJECT_CPD_INST_OPERANDS}
   route.route.openshift.io/ibm-lh-lakehouse-minio-console created

   oc get route ibm-lh-lakehouse-minio-console -n ${PROJECT_CPD_INST_OPERANDS}
   NAME                            HOST/
   PORT                                                     PATH   SERVICES
   PORT  TERMINATION  WILDCARD
   ibm-lh-lakehouse-minio-console  ibm-lh-lakehouse-minio-console-
   zen.apps.sprat.cp.fyre.ibm.com        ibm-lh-lakehouse-minio-svc  9001  edge         None
   ```

3. Copy and paste the route hostname into your browser.

   Remember: Ensure that the browser is using `https://` and not `http://`.

   Example:

   `https://ibm-lh-lakehouse-minio-console-zen.apps.mycluster.com`

   Remember: Ensure that the port is either unspecified or 443.

4. Log in to the MinIO console by using the credentials obtained in step 1.

   Tip: The MinIO console uses the storage's S3 access key as username and the S3 secret key as password.

What to do next

- You can use your own certificates to customize the edge route. For more information, see the OpenShift documentation.
- To connect your application to MinIO, see Software Development Kits (SDK).
- To configure object retention, see MinIO Object Locking.
- For security configuration, see Identity and Access Management.

Customization

Based on your requirements, you can customize various components in watsonx.data. You can find the details in the following sections.

watsonx.data on Red Hat OpenShift

Customizing components

A project administrator can specify additional customization of properties for different components in IBM watsonx.data. You can set additional customizations other than the default ones as part of the post-install procedure. The following specifications are optional and it can be altered to change the component-level customizations.

watsonx.data on Red Hat OpenShift

You can customize the watsonx.data components by following the instructions in "Customizing watsonx.data components" on page 266.

The possible custom resource descriptions are listed in the following topics.

Note: watsonx.data supports customization of some of the properties through API. For more information, see API customization.

Presto (Java) coordinator

Properties that can be customized for Presto (Java) coordinator are listed here.

watsonx.data on Red Hat OpenShift

Table 9. watsonx.data component: Presto (Java) coordinator

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_resources_limits_cpu`	Resource CPU limit for Presto (Java) coordinator, container is allowed to use only this much CPU.	Kubernetes CPU Unit	`small`: 12; `small_mincpureq`: 12; `medium`: 12; `large`: 12; `xlarge`: 12; `xxlarge`: 12	`resources.limits.cpu`	N
`presto_coordinator_resources_limits_memory`	Resource Memory limit for Presto (Java) coordinator, container is allowed to use only this much memory.	Units of Bytes **Note:** For more information about the memory unit, see Memory resource units.	`small`: 100G; `small_mincpureq`: 100G; `medium`: 100G; `large`: 100G; `xlarge`: 100G; `xxlarge`: 100G	`resources.limits.memory` **Note:** For more information about the memory unit, see Memory resource units.	N

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_resources_limits_ephemeral_storage`	This parameter sets the maximum amount of local ephemeral storage that a container in a Presto (Java) coordinator Pod can consume.	Units of Bytes	`small`: 10G; `small_mincpureq`: 10G; `medium`: 10G; `large`: 10G; `xlarge`: 10G; `xxlarge`: 10G	`resources.limits.ephemeral-storage`	N
`presto_coordinator_resources_requests_cpu`	Resource CPU request for Presto (Java) coordinator.	Kubernetes CPU Unit	`small`: 12; `small_mincpureq`: 0.005; `medium`: 12; `large`: 12; `xlarge`: 12; `xxlarge`: 12	`resources.requests.cpu`	N
`presto_coordinator_resources_requests_memory`	Resource memory request for Presto (Java) coordinator.	Units of Bytes **Note:** For more information about the memory unit, see Memory resource units.	`small`: 100G; `small_mincpureq`: 100G; `medium`: 100G; `large`: 100G; `xlarge`: 100G; `xxlarge`: 100G	`resources.requests.memory` **Note:** For more information about the memory unit, see Memory resource units.	N
`presto_coordinator_resources_requests_ephemeral_storage`	This parameter sets the minimum/ guaranteed size of local ephemeral storage that a container in a Presto (Java) coordinator Pod requests.	Units of Bytes	`small`: 1G; `small_mincpureq`: 1G; `medium`: 1G; `large`: 1G; `xlarge`: 1G; `xxlarge`: 1G	`resources.request.ephemeral-storage`	N
`presto_coordinator_jvm_Xmx`	Xmx specifies the maximum memory allocation pool for a Java virtual machine (JVM).	-		`jvm.config.Xmx`	Y

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_task_concurrency`	Default local concurrency for parallel operators such as joins and aggregations.	Number(must be the power of two)	16	`config.properties.task.concurrency`	Y
`presto_coordinator_query_max_memory`	The maximum amount of user memory that a query can use across the entire cluster.	Data size	1TB	`config.properties.query.max-memory`	Y
`presto_coordinator_query_max_memory_per_node`	The maximum amount of user memory that a query can use on a worker.	Data size	`presto_coordinator_jvm_Xmx*0.795`	`config.properties.query.max-memory-per-node`	Y
`presto_coordinator_query_max_total_memory_per_node`	The maximum amount of user and system memory that a query can use on a worker.	Data size	`presto_coordinator_jvm_Xmx*0.795`	`config.properties.query.max-total-memory-per-node`	Y
`presto_coordinator_query_max_concurrent_queries`	Describes how many queries can be processed simultaneously in a single cluster node.	Integer	15	`config.properties.query.max-concurrent-queries`	Y
`presto_coordinator_memory_heap_headroom_per_node`	The amount of memory set aside as headroom/ buffer in the JVM heap for allocations that are not tracked by Presto (Java).	Data size	`presto_coordinator_jvm_Xmx*0.2`	`config.properties.query.memory.heap-headeroom-per-node`	Y
`presto_coordinator_query_max_total_memory`	The maximum amount of user and system memory that a query can use across the entire cluster.	Data size	2TB	`config.properties.query.max-total-memory`	Y

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_experimental_optimized_repartitioning`	Improve performance of repartitioning data between stages.	Boolean	`true`	`experimental.optimized-repartitioning`	Y
`presto_coordinator_experimental_pushdown_dereference_enabled`	Add support for pushdown of dereference expressions for querying nested data.	Boolean		`experimental.pushdown-dereference-enabled`	Y
`presto_coordinator_experimental_pushdown_subfields_enabled`	Add support for pushdown of subfields expressions for querying nested data.	Boolean		`experimental.pushdown-subfields-enabled`	Y
`presto_coordinator_join_max_broadcast_table_size`	Add join-max-broadcast-table-size configuration property and`join_max_broadcast_table_size` session property to control the maximum estimated size of a table that can be broadcast when using `AUTOMATIC` join distribution type.	Integer		`join-max-broadcast-table-size`	Y

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_node_scheduler_max_pending_splits_per_task`	The number of outstanding splits with the standard split weight that can be queued for each worker node for a single stage of a query, even when the node is already at the limit for total number of splits. Allowing a minimum number of splits per stage is required to prevent starvation and deadlocks. This value must be smaller than `node-scheduler.max-splits-per-node`, will usually be increased for the same reasons, and has similar drawbacks if set too high.	Integer		`node-scheduler.max-pending-splits-per-task`	Y

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_node_scheduler_max_splits_per_node`	The target value for the total number of splits that can be running for each worker node, assuming all splits have the standard split weight. Using a higher value is recommended, if queries are submitted in large batches (for example, running a large group of reports periodically), or for connectors that produce many splits that complete quickly but do not support assigning split weight values to express that to the split scheduler.	Integer		`node-scheduler.max-splits-per-node`	Y
`presto_coordinator_optimizer_prefer_partial_aggregation`	This property allow users to disable partial aggregations for queries that do not benefit.	Boolean		`optimizer.prefer-partial-aggregation`	Y
`presto_coordinator_query_execution_policy`	Configures the algorithm to organize the processing of all of the stages of a query.	String	phased	`query.execution-policy`	Y

Table 9. watsonx.data component: Presto (Java) coordinator (continued)					
Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_coordinator_query_low_memory_killer_policy`	The policy used for selecting the query to kill when the cluster is out of memory (OOM). This property can have one of the following values: `none`, `total-reservation`, or `total-reservation-on-blocked-nodes`. none disables the cluster OOM killer. The value of total-reservation configures a policy that kills the query with the largest memory reservation across the cluster. The value of `total-reservation-on-blocked-nodes` configures a policy that kills the query using the most memory on the workers that are out of memory (blocked).	String	`total-reservation-on-blocked-nodes`	`query.low-memory-killer.policy`	Y

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_coordinator_query_max_stage_count`	Add a limit on the number of stages in a query. The default is 100 and can be changed with the `query.max-stage-count`configuration property and the `query_max_stage_count` session property.	Integer	200	`query.max-stage-count`	Y
`presto_coordinator_query_min_schedule_split_batch_size`	Add `query.min-schedule-split-batch-size` config flag to set the minimum number of splits to consider for scheduling per batch.	Boolean		`query.min-schedule-split-batch-size`	Y
`presto_coordinator_query_stage_count_warning_threshold`	Add a config option (`query.stage-count-warning-threshold`) to specify a per-query threshold for the number of stages. When this threshold is exceeded, a `TOO_MANY_STAGES` warning is raised.	Integer	150	`query.stage-count-warning-threshold`	Y

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_scale_writers`	Enable writer scaling by dynamically increasing the number of writer tasks on the cluster.	Boolean		`scale-writers`	Y
`presto_coordinator_sink_max_buffer_size`	Buffer size for IO writes while collecting pipeline results. Higher value may increase speed of IO operations with the cost of additional memory. Also higher value may increase number of data lost when presto node will fail effectively slowing down IO in unstable environment.	Integer		`sink.max-buffer-size`	Y
`presto_coordinator_experimental_max_revocable_memory_per_node`	The amount of revocable memory a query can use on each node.	Units of Bytes		`experimental.max-revocable-memory-per-node`	Y
`presto_coordinator_experimental_reserved_pool_enabled`	This property allows users to enable or disable Reserved Pool in Presto (Java). When the General Pool is full, this property uses OOM killer in Presto (Java) to increase the General Pool concurrency and prevent the deadlock.	Boolean	False	`experimental.reserved-pool-enabled`	Y

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_query_min_expire_age`	This property describes the minimum time after which you can remove the query metadata from the server.	String (Duration)	120 minutes	`query.min-expire-age`	Y
`presto_coordinator_enable_dynamic_filtering`	This property improves performance for queries with broadcast or collocated joins by adding dynamic filtering and storage pruning support.	Boolean		`experimental.enable-dynamic-filtering`	Y
`presto_coordinator_com_facebook_presto_governance`	This property sets the minimum log level for the logger `com.facebook.presto.governance`. It helps to customize the logging behavior based on the severity of log messages.	String(log levels) **Note:** There are four levels: **`DEBUG`**, **`INFO`**, **`WARN`** and **`ERROR`**		`com.facebook.presto.governance`	
`presto_coordinator_com_facebook_presto_governance_util`	This property sets the minimum log level for the logger `com.facebook.presto.governance_util`. It helps to customize the logging behavior based on the severity of log messages.	String(log levels) **Note:** There are four levels: **`DEBUG`**, **`INFO`**, **`WARN`** and **`ERROR`**		`com.facebook.presto.governance.util`	

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_com_facebook_presto_dispatcher`	This property sets the minimum log level for the logger `com.facebook.presto.dispatcher`. It helps to customize the logging behavior based on the severity of log messages.	String(log levels) **Note:** There are four levels: **`DEBUG`**, **`INFO`**, **`WARN`** and **`ERROR`**		`com.facebook.presto.dispatcher`	
`presto_coordinator_exchange_client_threads`	This property helps to control the number of threads used by exchange clients in Presto (Java) to fetch data from other Presto (Java) nodes during query execution.	Integer		`exchange.client-threads`	

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_exchange_http_client_max_connections`	This property specifies the maximum number of HTTP connections that the Exchange service can establish concurrently across all servers it interacts with. it helps to regulate the total number of simultaneous connections used by the exchange client for communication between Presto (Java) nodes.	Integer		`exchange.http-client.max-connections`	
`presto_coordinator_exchange_http_client_max_requests_queued_per_destination`	This property determines the maximum number of HTTP requests that can be queued for each destination server by the exchange client.	Integer		`exchange.http-client.max-requests-queued-per-destination`	
`presto_coordinator_http_server_log_max_size`	This property specifies the maximum file size for the log file generated by the HTTP server component.	Units of Bytes		`http-server.log.max-size`	

Table 9. watsonx.data component: Presto (Java) coordinator (continued)					
Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_coordinator_http_server_log_max_history`	The property specifies the maximum number of log files that the HTTP server component will retain before rotating old log content	Integer		`http-server.log.max-history`	
`presto_coordinator_http_server_threads_max`		Integer		`http-server.threads.max`	
`presto_coordinator_join_max_broadcast_table_size`	This property allows to specify a maximum size for replicated tables used in joins.	Units of Bytes		`join-max-broadcast-table-size`	
`presto_coordinator_log_max_history`	This property represents the maximum number of general application log files retained by a logging system before older logs are rotated out.	Integer		`log.max-history`	
`presto_coordinator_log_max_size`	The property `log.max-size` defines the maximum file size allowed for the general application log file.	Units of Bytes		`log.max-size`	

Table 9. watsonx.data component: Presto (Java) coordinator (continued)					
Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_coordinator_node_scheduler_max_splits_per_node`	This property specifies the target maximum number of splits that can concurrently run on each worker node. Splits represent units of work within queries. Adjusting this property allows administrators to optimize resource utilization, especially in scenarios involving large query batches or connectors generating numerous splits. ⚠ **CAUTION:** Setting `presto_coordinator_node_scheduler_max_splits_per_node` at too high value might lead to inefficient memory usage and performance degradation. Set this property such that there is always at least one split waiting to be processed, but not higher.	Integer		`node-scheduler.max-splits-per-node`	

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_optimize_nulls_in_join`	This property when enabled reduces the overhead of processing `NULL` values during `JOIN` operations, particularly beneficial when dealing with columns containing a significant number of `NULL` values.	Boolean		`optimize-nulls-in-join`	
`presto_coordinator_optimizer_default_filter_factor_enabled`	This property enables the use of a default value for estimating the cost of filters in query optimization.	Boolean		`optimizer.default-filter-factor-enabled`	
`presto_coordinator_optimizer_exploit_constraints`	This property enables constraint optimizations for querying catalogs that support table constraints.	Boolean		`optimizer.exploit-constraints`	
`presto_coordinator_query_client_timeout`	This property specifies the duration for which the cluster waits without any communication from the client application, (for example CLI) before abandoning and canceling the ongoing query or task.	String (Duration)		`query.client.timeout`	

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_query_max_execution_time`	This property specifies the maximum allowed time for a query to be actively executing on the cluster before termination.	String (Duration)		`query.max-execution-time`	
`presto_coordinator_query_max_history`	This property refers to the maximum number of queries to keep in the query history to provide statistics and other information. If this value is reached, queries are removed based on age	Integer		`query.max-history`	
`presto_coordinator_query_max_length`	This property specifies the maximum number of characters allowed for the SQL query text. Longer queries are not processed, and are terminated with error.	Integer		`query.max-length`	

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_shutdown_grace_period`	This property specifies the duration of time that the system waits after receiving a shutdown request before initiating the shutdown process. During this grace period, the system continues to operate normally, allowing ongoing active tasks to complete.	String (Duration)		`shutdown.grace-period`	
`presto_coordinator_experimental_max_spill_per_node`	This property refers to the maximum spill space used by all queries on a single node (when the memory allocated for query processing is exceeded).	Units of Bytes		`experimental.max-spill-per-node`	
`presto_coordinator_experimental_query_max_spill_per_node`	This property refers to the maximum spill space used by a single query on a single node.	Units of Bytes		`experimental.query-max-spill-per-node`	

Table 9. watsonx.data component: Presto (Java) coordinator (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_coordinator_experimental_spiller_max_used_space_threshold`	This property sets a threshold disk space usage ratio. If the usage exceeds beyond threshold value, this spill path becomes ineligible for spilling.	Double		`experimental.spiller-max-used-space-threshold`	
`presto_coordinator_experimental_spiller_spill_path`	This property specifies a directory where spilled content is written. It can be a comma separated list to spill simultaneously to multiple directories, which helps to use multiple drives installed in the system. (It is recommended to avoid spilling to system drives and to ensure that spill operations do not interfere with the JVM operation or disk performance.)	String		`experimental.spiller-spill-path`	
`presto_coordinator_httpserver_max_request_header_size`	This property is used to set the maximum size of the request header that `http` supports.	Data size	16kB	`httpserver.max_request_header_size`	Y

Table 9. watsonx.data component: Presto (Java) coordinator (continued)					
Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_coordinator_httpserver_max_response_header_size`	This property is used to set the maximum size of the response header that `http` supports.	Data size	16kB	`httpserver.max_response_header_size`	Y
`presto_coordinator_join_distribution_type`	This property specifies the type of distributed join to use. Allowed values include: `AUTOMATIC`, `PARTITIONED`, `BROADCAST`	String	`AUTOMATIC`	`join-distribution-type`	

Presto (Java) worker

Properties that can be customized for Presto (Java) worker are listed here.

watsonx.data on Red Hat OpenShift

Table 10. watsonx.data component: Presto (Java) worker					
Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_worker_replicas`	The number of replicas for Presto (Java) worker.	Integer	`small`: 3; `small_mincpureq`: 3; `medium`: 9; `large`: 19; `xlarge`: 69; `xxlarge`: 199	`spec.replicas`	N
`presto_worker_resources_limits_cpu`	Resource CPU limit for Presto (Java) worker, container is allowed to use only this much CPU.	Kubernetes CPU Unit	`small`: 12; `small_mincpureq`: 12; `medium`: 12; `large`: 12; `xlarge`: 12; `xxlarge`: 12	`resources.limits.cpu`	N
`presto_worker_resources_limits_memory`	Resource Memory limit for Presto (Java) worker, container is allowed to use only this much memory.	Units of Bytes **Note:** For more information about the memory unit, see Memory resource units.	`small`: 100G; `small_mincpureq`: 100G; `medium`: 100G; `large`: 100G; `xlarge`: 100G; `xxlarge`: 100G	`resources.limits.memory` **Note:** For more information about the memory unit, see Memory resource units.	N

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_worker_resources_limits_ephemeral_storage`	This parameter sets the maximum amount of local ephemeral storage that a container in a Presto (Java) worker Pod can consume.	Units of Bytes	`small`: 10G; `small_mincpureq`: 10G; `medium`: 10G; `large`: 10G; `xlarge`: 10G; `xxlarge`: 10G	`resources.limits.ephemeral-storage`	N
`presto_worker_resources_requests_cpu`	Resource CPU request for Presto (Java) worker.	Kubernetes CPU Unit	`small`: 12; `small_mincpureq`: 0.005; `medium`: 12; `large`: 12; `xlarge`: 12; `xxlarge`: 12	`resources.requests.cpu`	N
`presto_worker_resources_requests_memory`	Resource memory request for Presto (Java) worker.	Units of Bytes **Note:** For more information about the memory unit, see Memory resource units.	`small`: 100G; `small_mincpureq`: 100G; `medium`: 100G; `large`: 100G; `xlarge`: 100G; `xxlarge`: 100G	`resources.requests.memory` **Note:** For more information about the memory unit, see Memory resource units.	N
`presto_worker_jvm_Xmx`	Xmx specifies the maximum memory allocation pool for a Java virtual machine (JVM).	-		`jvm.config.Xmx`	Y
`presto_worker_task_concurrency`	Default local concurrency for parallel operators such as joins and aggregations.	Number(must be the power of two)	16	`config.properties.task.concurrency`	Y
`presto_worker_query_max_memory`	The maximum amount of user memory that a query can use across the entire cluster.	Data size	1TB	`config.properties.query.max-memory`	Y

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_worker_query_max_memory_per_node`	The maximum amount of user memory that a query can use on a worker.	Data size	`presto_worker_jvm_Xmx*0.795`	`config.properties.query.max-memory-per-node`	Y
`presto_worker_query_max_total_memory_per_node`	The maximum amount of user and system memory that a query can use on a worker.	Data size	`presto_worker_jvm_Xmx*0.795`	`config.properties.query.max-total-memory-per-node`	Y
`presto_worker_query_max_concurrent_queries`	Describes how many queries can be processed simultaneously in a single cluster node.	Integer	15	`config.properties.query.max-concurrent-queries`	Y
`presto_worker_memory_heap_headroom_per_node`	This is the amount of memory set aside as headroom/ buffer in the JVM heap for allocations that are not tracked by Presto (Java).	Data size	`presto_worker_jvm_Xmx*0.2`	`config.properties.query.memory.heap-headeroom-per-node`	Y
`presto_worker_query_max_total_memory`	The maximum amount of user and system memory that a query can use across the entire cluster.	Data size	2TB	`config.properties.query.max-total-memory`	Y
`presto_worker_experimental_optimized_repartitioning`	Improve performance of repartitioning data between stages.	Boolean	`true`	`experimental.optimized-repartitioning`	Y
`presto_worker_experimental_pushdown_dereference_enabled`	Add support for pushdown of dereference expressions for querying nested data.	Boolean		`experimental.pushdown-dereference-enabled`	Y

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_experimental_pushdown_subfields_enabled`	Add support for pushdown of subfields expressions for querying nested data.	Boolean		`experimental.pushdown-subfields-enabled`	Y
`presto_worker_join_max_broadcast_table_size`	Add join-max-broadcast-table-size configuration property and `join_max_broadcast_table_size` session property to control the maximum estimated size of a table that can be broadcast when using AUTOMATIC join distribution type.	Integer		`join-max-broadcast-table-size`	Y

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_node_scheduler_max_pending_splits_per_task`	The number of outstanding splits with the standard split weight that can be queued for each worker node for a single stage of a query, even when the node is already at the limit for total number of splits. Allowing a minimum number of splits per stage is required to prevent starvation and deadlocks. This value must be smaller than `node-scheduler.max-splits-per-node`, will usually be increased for the same reasons, and has similar drawbacks if set too high.	Integer		`node-scheduler.max-pending-splits-per-task`	Y

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_node_scheduler_max_splits_per_node`	The target value for the total number of splits that can be running for each worker node, assuming that all splits have the standard split weight. Using a higher value is recommended, if queries are submitted in large batches (for example, running a large group of reports periodically), or for connectors that produce many splits that complete quickly but do not support assigning split weight values to express that to the split scheduler.	Integer		`node-scheduler.max-splits-per-node`	Y
`presto_worker_optimizer_prefer_partial_aggregation`	This property allows users to disable partial aggregations for queries that do not benefit.	Boolean		`optimizer.prefer-partial-aggregation`	Y
`presto_worker_query_execution_policy`	Configures the algorithm to organize the processing of all of the stages of a query.	String	phased	`query.execution-policy`	Y

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_query_low_memory_killer_policy`	The policy that is used for selecting the query to kill when the cluster is out of memory (OOM). This property can have one of the following values: none, total-reservation, or `total-reservation-on-blocked-nodes`. none disables the cluster OOM killer. The value of total-reservation configures a policy that kills the query with the largest memory reservation across the cluster. The value of `total-reservation-on-blocked-nodes` configures a policy that kills the query by using the most memory on the workers that are out of memory (blocked).	String	`total-reservation-on-blocked-nodes`	`query.low-memory-killer.policy`	Y

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_query_max_stage_count`	Add a limit on the number of stages in a query. The default is 100 and can be changed with the `query.max-stage-count` configuration property and the query_max_stage_count session property.	Integer	200	`query.max-stage-count`	Y
`presto_worker_query_min_schedule_split_batch_size`	Add `query.min-schedule-split-batch-size` config flag to set the minimum number of splits to consider for scheduling per batch.	Boolean		`query.min-schedule-split-batch-size`	Y
`presto_worker_query_stage_count_warning_threshold`	Add a config option (`query.stage-count-warning-threshold`) to specify a per-query threshold for the number of stages. When this threshold is exceeded, a `TOO_MANY_STAGES` warning is raised.	Integer	150	`query.stage-count-warning-threshold`	Y

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_scale_writers`	Enable writer scaling by dynamically increasing the number of writer tasks on the cluster.	Boolean		`scale-writers`	Y
`presto_worker_sink_max_buffer_size`	Buffer size for IO writes while collecting pipeline results. Higher value may increase the speed of IO operations with the cost of additional memory. Also higher value may increase the number of data lost when presto node fails effectively slowing down IO in an unstable environment.	Integer		`sink.max-buffer-size`	Y
`presto_worker_experimental_max_revocable_memory_per_node`	The amount of revocable memory a query can use on each node.	Units of Bytes		`experimental.max-revocable-memory-per-node`	Y
`presto_worker_experimental_reserved_pool_enabled`	This property allows users to enable or disable Reserved Pool in Presto (Java). When the General Pool is full, this property uses OOM killer in Presto (Java) to increase the General Pool concurrency and prevent the deadlock.	Boolean	False	`experimental.reserved-pool-enabled`	Y

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_worker_query_min_expire_age`	This property describes the minimum time after which you can remove the query metadata from the server.	String (Duration)	120 minutes	`query.min-expire-age`	Y
`presto_worker_enable_dynamic_filtering`	This property improves performance for queries with broadcast or collocated joins by adding dynamic filtering and storage pruning support.	Boolean		`experimental.enable-dynamic-filtering`	Y
`presto_worker_exchange_client_threads`	This property helps to control the number of threads that are used by exchange clients in Presto (Java) to fetch data from other Presto (Java) nodes during query execution	Integer		`exchange.client-threads`	

Table 10. watsonx.data component: Presto (Java) worker (continued)					
Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_worker_exchange_http_client_max_connections`	This property specifies the maximum number of HTTP connections that the Exchange service can establish concurrently across all servers it interacts with. it helps to regulate the total number of simultaneous connections used by the exchange client for communication between Presto (Java) nodes.	Integer		`exchange.http-client.max-connections`	
`presto_worker_exchange_http_client_max_requests_queued_per_destination`	This property determines the maximum number of HTTP requests that can be queued for each destination server by the exchange client.	Integer		`exchange.http-client.max-requests-queued-per-destination`	
`presto_worker_http_server_log_max_size`	This property specifies the maximum file size for the log file that is generated by the HTTP Server component.	Units of Bytes		`http-server.log.max-size`	

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_http_server_log_max_history`	The property specifies the maximum number of log files that the HTTP Server component retains before rotating old log content	Integer		`http-server.log.max-history`	
`presto_worker_http_server_threads_max`		Integer		`http-server.threads.max`	
`presto_worker_join_max_broadcast_table_size`	This property allows to specify a maximum size for replicated tables used in joins.	Units of Bytes		`join-max-broadcast-table-size`	
`presto_worker_log_max_history`	This property represents the maximum number of general application log files that are retained by a logging system before older logs are rotated out.	Integer		`log.max-history`	
`presto_worker_log_max_size`	The property `log.max-size` defines the maximum file size that is allowed for the general application log file.	Units of Bytes		`log.max-size`	

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
presto_worker_node_scheduler_max_splits_per_node	This property specifies the target maximum number of splits that can concurrently run on each worker node. Splits represent units of work within queries. Adjusting this property allows administrators to optimize resource utilization, especially in scenarios involving large query batches or connectors generating numerous splits. **CAUTION:** Setting presto_worker_node_scheduler_max_splits_per_node at too high value might lead to inefficient memory usage and performance degradation. Ideally, it should be set such that there is always at least one split waiting to be processed, but not higher.	Integer		node-scheduler.max-splits-per-node	

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_optimize_nulls_in_join`	This property when enabled reduces the overhead of processing `NULL` values during `JOIN` operations, particularly beneficial when dealing with columns containing a significant number of `NULL` values.	Boolean		`optimize-nulls-in-join`	
`presto_worker_optimizer_default_filter_factor_enabled`	This property enables the use of a default value for estimating the cost of filters in query optimization.	Boolean		`optimizer.default-filter-factor-enabled`	
`presto_worker_optimizer_exploit_constraints`	This property enables constraint optimizations for querying catalogs that support table constraints.	Boolean		`optimizer.exploit-constraints`	
`presto_worker_query_client_timeout`	This property specifies the duration for which the cluster waits without any communication from the client application, (for example, CLI) before abandoning and canceling the ongoing query or work.	String (Duration)		`query.client.timeout`	

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_query_max_execution_time`	This property specifies the maximum allowed time for a query to be actively executing on the cluster before it is terminated.	String (Duration)		`query.max-execution-time`	
`presto_worker_query_max_history`	This property refers to the maximum number of queries to keep in the query history to provide statistics and other information. If this value is reached, queries are removed based on age	Integer		`query.max-history`	
`presto_worker_query_max_length`	The maximum number of characters allowed for the SQL query text. Longer queries are not processed, and are terminated with error.	Integer		`query.max-length`	

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_shutdown_grace_period`	This property specifies the duration of time that the system waits after receiving a shutdown request before initiating the shutdown process. During this grace period, the system continues to operate normally, allowing ongoing active tasks to be complete.	String (Duration)		`shutdown.grace-period`	
`presto_worker_experimental_max_spill_per_node`	This property refers to the maximum spill space to be used by all queries on a single node (when the memory that is allocated for query processing is exceeded).	Units of Bytes		`experimental.max-spill-per-node`	
`presto_worker_experimental_query_max_spill_per_node`	This property refers to the maximum spill space to be used by a single query on a single node.	Units of Bytes		`experimental.query-max-spill-per-node`	

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_experimental_spiller_max_used_space_threshold`	This property sets a threshold disk space usage ratio, if the usage exceeds beyond this value, this spill path will not be eligible for spilling.	Double		`experimental.spiller-max-used-space-threshold`	
`presto_worker_experimental_spiller_spill_path`	This property specifies a directory where spilled content is written. It can be a comma-separated list to spill simultaneously to multiple directories, which helps to use multiple drives installed in the system. (It is recommended to avoid spilling to system drives and to ensure that spill operations do not interfere with the JVM operation or disk performance.)	String		`experimental.spiller-spill-path`	
`presto_worker_resources_requests_ephemeral_storage`	This parameter sets the minimum/ guaranteed size of local ephemeral storage that a container in a Presto (Java) worker Pod requests.	Units of Bytes	`small`: 1G; `small_mincpureq`: 1G; `medium`: 1G; `large`: 1G; `xlarge`: 1G; `xxlarge`: 1G	`resources.request.ephemeral-storage`	N

Table 10. watsonx.data component: Presto (Java) worker (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_worker_httpserver_max_request_header_size`	This property is used to set the maximum size of the request header that `http` supports.	Data size	16kB	`httpserver.max_request_header_size`	Y
`presto_worker_httpserver_max_response_header_size`	This property is used to set the maximum size of the response header that `http` supports.	Data size	16kB	`httpserver.max_response_header_size`	Y

Presto (Java) singlenode

Properties that can be customized for Presto (Java) singlenode are listed here.

watsonx.data on Red Hat OpenShift

Table 11. watsonx.data component: Presto (Java) singlenode

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_resources_limits_cpu`	Resource CPU limit for Presto (Java) singlenode container is allowed to use only this much CPU.	Kubernetes CPU Unit	`small`: 3; `small_mincpureq`: 3; `medium`: 6; `large`: 9; `xlarge`: 12; `xxlarge`: 12	`resources.limits.cpu`	N
`presto_singlenode_resources_limits_memory`	Resource Memory limit for Presto (Java) singlenode container is allowed to use only this much memory.	Units of Bytes **Note:** For more information about the memory unit, see Memory resource units.	`small`: 24G; `small_mincpureq`: 24G; `medium`: 48G; `large`: 72G; `xlarge`: 96G; `xxlarge`: 96G	`resources.limits.memory`	N
`presto_singlenode_resources_limits_ephemeral_storage`	This parameter sets the maximum amount of local ephemeral storage that a container in a Presto (Java) singlenode Pod can consume.	Units of Bytes	`small`: 10G; `small_mincpureq`: 10G; `medium`: 10G; `large`: 10G; `xlarge`: 10G; `xxlarge`: 10G	`resources.limits.ephemeral-storage`	N

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_resources_requests_cpu`	Resource CPU request for Presto (Java) single node.	Kubernetes CPU Unit	`small`: 3; `small_mincpureq`: 0.005; `medium`: 6; `large`: 9; `xlarge`: 12; `xxlarge`: 12	`resources.requests.cpu`	N
`presto_singlenode_resources_requests_memory`	Resource memory request for Presto (Java) singlenode.	Units of Bytes **Note:** For more information about the memory unit, see Memory resource units.	`small`: 24G; `small_mincpureq`: 24G; `medium`: 48G; `large`: 72G; `xlarge`: 96G; `xxlarge`: 96G	`resources.requests.memory`	N
`presto_singlenode_resources_requests_ephemeral_storage`	This parameter sets the minimum/ guaranteed amount of local ephemeral storage for a container in a Presto (Java) singlenode Pod.	Units of Bytes	`small`: 500Mi; `small_mincpureq`: 500Mi; `medium`: 1G; `large`: 1G; `xlarge`: 1G; `xxlarge`: 1G	`resources.request.ephemeral-storage`	N
`presto_singlenode_jvm_Xmx`	Xmx specifies the maximum memory allocation pool for a Java virtual machine (JVM).	-		`jvm.config.Xmx`	Y
`presto_singlenode_task_concurrency`	Default local concurrency for parallel operators such as joins and aggregations.	Number(must be the power of two)		`config.properties.task.concurrency`	Y
`presto_singlenode_query_max_memory`	The maximum amount of user memory that a query can use across the entire cluster.	Data size	1TB	`config.properties.query.max-memory`	Y
`presto_singlenode_query_max_memory_per_node`	The maximum amount of user memory that a query can use on a worker.	Data size	presto_singlenode_jvm_Xmx*0.795	`config.properties.query.max-memory-per-node`	Y

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_query_max_total_memory_per_node`	The maximum amount of user and system memory that a query can use on a worker.	Data size	presto_singlenode_jvm_Xmx*0.795	`config.properties.query.max-total-memory-per-node`	Y
`presto_singlenode_query_max_concurrent_queries`	Describes how many queries can be processed simultaneously in a single cluster node.	Integer		`config.properties.query.max-concurrent-queries`	Y
`presto_singlenode_memory_heap_headroom_per_node`	This is the amount of memory set aside as headroom/ buffer in the JVM heap for allocations that are not tracked by Presto (Java).	Data size	presto_singlenode_jvm_Xmx*0.2	`config.properties.query.memory.heap-headeroom-per-node`	Y
`presto_singlenode_query_max_total_memory`	The maximum amount of user and system memory that a query can use across the entire cluster.	Data size	2TB	`config.properties.query.max-total-memory`	Y
`presto_singlenode_experimental_optimized_repartitioning`	Improve performance of repartitioning data between stages.	Boolean	**`true`**	`experimental.optimized-repartitioning`	Y
`presto_singlenode_experimental_pushdown_dereference_enabled`	Add support for pushdown of dereference expressions for querying nested data.	Boolean		`experimental.pushdown-dereference-enabled`	Y
`presto_singlenode_experimental_pushdown_subfields_enabled`	Add support for pushdown of subfields expressions for querying nested data.	Boolean		`experimental.pushdown-subfields-enabled`	Y

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_singlenode_join_max_broadcast_table_size`	Add join-max-broadcast-table-size configuration property and `join_max_broadcast_table_size` session property to control the maximum estimated size of a table that can be broadcast when using AUTOMATIC join distribution type.	Integer		`join-max-broadcast-table-size`	Y

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_node_scheduler_max_pending_splits_per_task`	The number of outstanding splits with the standard split weight that can be queued for each singlenode node for a single stage of a query, even when the node is already at the limit for total number of splits. Allowing a minimum number of splits per stage is required to prevent starvation and deadlocks. This value must be smaller than `node-scheduler.max-splits-per-node`, will usually be increased for the same reasons, and has similar drawbacks if set too high.	Integer		`node-scheduler.max-pending-splits-per-task`	Y

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_node_scheduler_max_splits_per_node`	The target value for the total number of splits that can be running for each worker node, assuming that all splits have the standard split weight. Using a higher value is recommended, if queries are submitted in large batches (for example, running a large group of reports periodically), or for connectors that produce many splits that complete quickly but do not support assigning split weight values to express that to the split scheduler.	Integer		`node-scheduler.max-splits-per-node`	Y
`presto_singlenode_optimizer_prefer_partial_aggregation`	This property allows users to disable partial aggregations for queries that do not benefit.	Boolean		`optimizer.prefer-partial-aggregation`	Y
`presto_singlenode_query_execution_policy`	Configures the algorithm to organize the processing of all of the stages of a query.	String	phased	`query.execution-policy`	Y

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_query_low_memory_killer_policy`	The policy that is used for selecting the query to kill when the cluster is out of memory (OOM). This property can have one of the following values: none, `total-reservation`, or `total-reservation-on-blocked-nodes`. none disables the cluster OOM killer. The value of total-reservation configures a policy that kills the query with the largest memory reservation across the cluster. The value of `total-reservation-on-blocked-nodes` configures a policy that kills the query by using the most memory on the workers that are out of memory (blocked).	String	`total-reservation-on-blocked-nodes`	`query.low-memory-killer.policy`	Y

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
presto_singlenode_query_max_stage_count	Add a limit on the number of stages in a query. The default is 100 and can be changed with the query.max-stage-count configuration property and the query_max_stage_count session property.	Integer	200	query.max-stage-count	Y
presto_singlenode_query_min_schedule_split_batch_size	Add query.min-schedule-split-batch-size config flag to set the minimum number of splits to consider for scheduling per batch.	Boolean		query.min-schedule-split-batch-size	Y
presto_singlenode_query_stage_count_warning_threshold	Add a config ox`ption (query.stage-count-warning-threshold) to specify a per-query threshold for the number of stages. When this threshold is exceeded, a TOO_MANY_STAGES warning is raised.	Integer	150	query.stage-count-warning-threshold	Y

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_scale_writers`	Enable writer scaling by dynamically increasing the number of writer tasks on the cluster.	Boolean		`scale-writers`	Y
`presto_singlenode_sink_max_buffer_size`	Buffer size for IO writes while collecting pipeline results. Higher value may increase the speed of IO operations with the cost of additional memory. Also higher value may increase the number of data lost when presto node fails effectively slowing down IO in an unstable environment.	Integer		`sink.max-buffer-size`	Y
`presto_singlenode_experimental_max_revocable_memory_per_node`	The amount of revocable memory a query can use on each node.	Units of Bytes		`experimental.max-revocable-memory-per-node`	Y
`presto_singlenode_experimental_reserved_pool_enabled`	This property allows users to enable or disable Reserved Pool in Presto (Java). When the General Pool is full, this property uses OOM killer in Presto (Java) to increase the General Pool concurrency and prevent the deadlock.	Boolean	False	`experimental.reserved-pool-enabled`	Y

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_query_min_expire_age`	This property describes the minimum time after which you can remove the query metadata from the server.	String	120 minutes	`query.min-expire-age`	Y
`presto_singlenode_enable_dynamic_filtering`	This property improves performance for queries with broadcast or collocated joins by adding dynamic filtering and storage pruning support.	Boolean		`experimental.enable-dynamic-filtering`	Y
`presto_singlenode_exchange_client_threads`	This property helps to control the number of threads that are used by exchange clients in Presto (Java) to fetch data from other Presto (Java) nodes during query execution	Integer		`exchange.client-threads`	
`presto_singlenode_exchange_http_client_max_connections`		Integer		`exchange.http-client.max-connections`	
`presto_singlenode_exchange_http_client_max_requests_queued_per_destination`	This property determines the maximum number of HTTP requests that can be queued for each destination server by the exchange client.	Integer		`exchange.http-client.max-requests-queued-per-destination`	

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_http_server_log_max_size`	This property specifies the maximum file size for the log file that is generated by the HTTP server component.	Units of Bytes		`http-server.log.max-size`	
`presto_singlenode_http_server_log_max_history`	The property specifies the maximum number of log files that the HTTP server component retains before rotating old log content	Integer		`http-server.log.max-history`	
`presto_singlenode_http_server_threads_max`		Integer		`http-server.threads.max`	
`presto_singlenode_join_max_broadcast_table_size`	This property allows to specify a maximum size for replicated tables used in joins.	Units of Bytes		`join-max-broadcast-table-size`	
`presto_singlenode_log_max_history`	This property represents the maximum number of general application log files that are retained by a logging system before older logs are rotated out.	Integer		`log.max-history`	

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_singlenode_log_max_size`	The property `log.max-size` defines the maximum file size that is allowed for the general application log file.	Units of Bytes		`log.max-size`	
`presto_singlenode_node_scheduler_max_splits_per_node`	This property specifies the target maximum number of splits that can concurrently run on each worker node. Splits represent units of work within queries. Adjusting this property allows administrators to optimize resource utilization, especially in scenarios involving large query batches or connectors generating numerous splits. ⚠ **CAUTION:** Setting `presto_singlenode_node_scheduler_max_splits_per_node` at too high value might lead to inefficient memory usage and performance degradation. Ideally, it should be set such that there is always at least one split waiting to be processed, but not higher.	Integer		`node-scheduler.max-splits-per-node`	

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_optimize_nulls_in_join`	This property when enabled reduces the overhead of processing NULL values during JOIN operations, particularly beneficial when dealing with columns containing a significant number of NULLs.	Boolean		`optimize-nulls-in-join`	
`presto_singlenode_optimizer_default_filter_factor_enabled`	This property enables the use of a default value for estimating the cost of filters in query optimization.	Boolean		`optimizer.default-filter-factor-enabled`	
`presto_singlenode_optimizer_exploit_constraints`	This property enables constraint optimizations for querying catalogs that support table constraints.	Boolean		`optimizer.exploit-constraints`	
`presto_singlenode_query_client_timeout`	This property specifies the duration for which the cluster waits without any communication from the client application, such as the CLI, before abandoning and canceling the ongoing query or work.	String (Duration)		`query.client.timeout`	

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_singlenode_query_max_execution_time`	This property specifies the maximum allowed time for a query to be actively executing on the cluster before it is terminated.	String (Duration)		`query.max-execution-time`	
`presto_singlenode_query_max_history`	This property refers to the maximum number of queries to keep in the query history to provide statistics and other information. If this amount is reached, queries are removed based on age	Integer		`query.max-history`	
`presto_singlenode_query_max_length`	The maximum number of characters allowed for the SQL query text. Longer queries are not processed, and are terminated with error.	Integer		`query.max-length`	

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_shutdown_grace_period`	This property specifies the duration of time that the system waits after receiving a shutdown request before initiating the shutdown process. During this grace period, the system continues to operate normally, allowing ongoing active tasks to be complete.	String (Duration)		`shutdown.grace-period`	
`presto_singlenode_experimental_max_spill_per_node`	This property refers to the maximum spill space to be used by all queries on a single node. (when the memory that is allocated for query processing is exceeded.)	Units of Bytes		`experimental.max-spill-per-node`	
`presto_singlenode_experimental_query_max_spill_per_node`	This property refers to the maximum spill space to be used by a single query on a single node.	Units of Bytes		`experimental.query-max-spill-per-node`	

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_experimental_spiller_max_used_space_threshold`	This property sets a threshold disk space usage ratio, if the usgage exceeds beyond this value, this spill path will not be eligible for spilling.	Double		`experimental.spiller-max-used-space-threshold`	
`presto_singlenode_experimental_spiller_spill_path`	This property specifies a directory where spilled content is written. It can be a comma-separated list to spill simultaneously to multiple directories, which helps to utilize multiple drives installed in the system. (It is recommended to avoid spilling to system drives and to ensure that spill operations do not interfere with the JVM operation or disk performance.)	String		`experimental.spiller-spill-path`	
`presto_singlenode_httpserver_max_request_header_size`	This property is used to set the maximum size of the request header that `http` supports.	Data size	16kB	`httpserver.max_request_header_size`	Y

Table 11. watsonx.data component: Presto (Java) singlenode (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_singlenode_httpserver_max_response_header_size`	This property is used to set the maximum size of the response header that `http` supports.	Data size	16kB	`httpserver.max_response_header_size`	Y

Presto (C++) worker

Properties that can be customized for Presto (C++) are listed here.

watsonx.data on Red Hat OpenShift

Table 12. watsonx.data component: Presto (C++)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`prestissimo_worker_replicas`	The number of Presto (C++) worker replicas to deploy for parallel query execution. Increasing the number of replicas can improve query performance by distributing workloads across multiple instances.	Integer	`small`: 3; `small_mincpureq`: 3; `medium`: 9; `large`: 19; `xlarge`: 69; `xxlarge`: 199	`replicas`	Y
`prestissimo_worker_resources_limits_cpu`	The CPU resource limits for each Presto (C++) worker replica. This value determines the maximum CPU usage that is allowed for each worker. **Note:** The value of `prestissimo_worker_resources_limits_cpu` must be greater than or equal to the value of `prestissimo_worker_resources_requests_cpu`.	Integer	12	`resources_limits_cpu`	Y

Table 12. watsonx.data component: Presto (C++) (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`prestissimo_worker_resources_limits_memory`	The memory resource limits for each Presto (C++) worker replica. This value sets the maximum memory allocation for each worker. This also impacts the `config.properties.system-memory-gb`, `config.properties.query-memory-gb`, and `config.properties.query.max-memory-per-node` system properties. • **`query-memory-gb:`** Specifies the total memory capacity that can be used by query execution in GB. The query memory capacity should be configured less than the system memory capacity (`system-memory-gb`) to reserve the memory for system usage such as disk spilling and cache prefetch, which are not counted in query memory usage. • **`system-memory-gb:`** It is the total memory capacity that is used for the system. • **`query.max-memory-per-node:`** It is similar to `query-memory-gb`, but `query-memory-gb` is enforced by memory arbitrator.	Integer	100G	`resources_limits_memory`	Y
`prestissimo_worker_resources_requests_cpu`	The CPU resource requests for each Presto (C++) worker replica. This value specifies the desired amount of CPU resources for each worker. **Note:** The value of `prestissimo_worker_resources_requests_cpu` must be less than or equal to the value of `prestissimo_worker_resources_limits_cpu`.	Integer	12	`resources_requests_cpu`	Y

Table 12. watsonx.data component: Presto (C++) (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`prestissimo_worker_resources_requests_memory`	The memory resource requests for each Presto (C++) worker replica. This value specifies the desired amount of memory for each worker.	Integer	100G	`resources_requests_memory`	Y
`prestissimo_worker_memory`	The amount of memory that is allocated to each Presto (C++) worker replica. This memory is used for query processing and other operations. (The base unit of `prestissimo_worker_memory` is GB.)	Integer	70	`memory`	Y
`prestissimo_worker_httpserver_max_request_header_size`	This property is used to set the maximum size of the request header that `http` supports.	Data size	16kB	`httpserver.max_request_header_size`	
`prestissimo_worker_httpserver_max_response_header_size`	This property is used to set the maximum size of the response header that `http` supports.	Data size	16kB	`httpserver.max_response_header_size`	
`prestissimo_worker_query_max_memory_per_node`	The maximum amount of user memory that a query can use on a worker.	Units of bytes		`query.max-memory-per-node`	
`prestissimo_worker_system_memory_gb`	This property refers to the memory given to the Presto (C++) engine under the Presto (C++) worker pod.	Units of bytes		`system-memory-gb`	
`prestissimo_worker_query_memory_gb`	This property refers to the total memory assigned to query execution for Presto (C++) worker engine.	Units of bytes		`query-memory-gb`	
`prestissimo_worker_async_data_cache_enabled`	This property when enabled prefetches data splits during table scans.	Boolean		`async-data-cache-enabled`	
`prestissimo_worker_task_max_drivers_per_task`	This property help to specify the maximum number of drivers a task can run concurrently.	Integer		`task.max-drivers-per-task`	

Table 12. watsonx.data component: Presto (C++) (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`prestissimo_worker_max_split_preload_per_driver`	This parameter refers to the maximum number of task splits that can be preloaded into memory per driver. Setting this parameter to a value **greater than 0** enables the preloading, while **setting it to 0** disables the preloading.	Integer		`max_split_preload_per_driver`	
`prestissimo_worker_resources_limits_ephemeral_storage`	This parameter sets the maximum amount of local ephemeral storage that a container in a Presto (C++) singlenode pod can consume.	Units of bytes	`small`: 10G; `small_mincpureq`: 10G; `medium`: 10G; `large`: 10G; `xlarge`: 10G; `xxlarge`: 10G	`resources.limits.ephemeral-storage`	No
`prestissimo_worker_resources_requests_ephemeral_storage`	This parameter sets the minimum or guaranteed size of local ephemeral storage for a container in a Presto (C++) singlenode pod.	Units of bytes	`small`: 1G; `small_mincpureq`: 1G; `medium`: 1G; `large`: 1G; `xlarge`: 1G; `xxlarge`: 1G	`resources.request.ephemeral-storage`	No

Postgres

Properties that can be customized for Postgres are listed here.

watsonx.data on Red Hat OpenShift

Table 13. watsonx.data component: Postgres

Property	Description	Type	System property	Restart containers required
`postgres_replicas`	The number of replicas for Postgres.	Integer	`spec.replicas`	N

Table 13. watsonx.data component: Postgres (continued)

Property	Description	Type	System property	Restart containers required
`postgres_resources_limits_cpu`	Resource CPU limit for Postgres. Container is allowed to use only this much CPU.	Kubernetes CPU Unit	`resources.limits.cpu`	N
`postgres_resources_limits_memory`	Resource Memory limit for Postgres. A container is allowed to use only this much memory.	Units of Bytes **Note:** For more information about the memory unit, see Memory resource units.	`resources.limits.memory`	N
`postgres_resources_requests_cpu`	Resource CPU request for Postgres.	Kubernetes CPU Unit	`resources.requests.cpu`	N
`postgres_resources_requests_memory`	Resource memory request for Postgres.	Units of Bytes **Note:** For more information about the memory unit, see Memory resource units.	`resources.requests.memory`	N

MinIO

Properties that can be customized for MinIO are listed here.

watsonx.data on Red Hat OpenShift

Table 14. watsonx.data component: MinIO

Property	Description	Type	System property	Restart containers required
`minio_resources_limits_cpu`	Resource CPU limit for MinIO. Container is allowed to use only this much CPU.	Kubernetes CPU Unit	`resources.limits.cpu`	N
`minio_resources_limits_memory`	Resource Memory limit for MinIO. A container is allowed to use only this much memory.	Units of Bytes **Note:** For more information about the memory unit, see Memory resource units.	`resources.limits.memory`	N
`minio_resources_requests_cpu`	Resource CPU request for MinIO.	Kubernetes CPU Unit	`resources.requests.cpu`	N

Table 14. watsonx.data component: MinIO (continued)

Property	Description	Type	System property	Restart containers required
`minio_resources_requests_memory`	Resource memory request for MinIO.	Units of Bytes **Note:** For more information about the memory unit, see Memory resource units.	`resources.requests.memory`	N

Hive

Properties that can be customized for Hive are listed here.

watsonx.data on Red Hat OpenShift

Table 15. watsonx.data component: Hive

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`hive_replicas`	The number of replicas for Hive.	Integer		`spec.replicas`	N
`hive_resources_limits_cpu`	Resource CPU limit for Hive. The container can use only the specified CPU capacity.	Kubernetes CPU Unit		`resources.limits.cpu`	N
`hive_resources_limits_memory`	Resource memory limit for Hive. The container can use only the specified memory units.	Bytes **Note:** For more information about the memory unit, see Memory resource units.		`resources.limits.memory`	N
`hive_resources_requests_cpu`	Resource CPU request for Hive.	Kubernetes CPU Unit		`resources.requests.cpu`	N
`hive_resources_requests_memory`	Resource memory request for Hive.	Bytes **Note:** For more information about the memory unit, see Memory resource units.		`resources.requests.memory`	N

Table 15. watsonx.data component: Hive (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`hive_connection_pool_max_pool_size`	The size of the connection pool used by Hive metastore. Increasing this value increases the maximum number of Spark jobs and clients that can connect to the metastore at the same	Integer	10	`datanucleus.connectionPool.maxPoolSize`	Y

Catalog

Properties that can be customized for catalog are listed here.

watsonx.data on Red Hat OpenShift

Table 16. watsonx.data component: Presto (Java) catalog

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_hive_max_outstanding_splits`	Limit of number of splits waiting to be served by split source. After reaching this limit writers will stop writing new splits to split source until some of them are used by workers. Higher value will increase memory usage, but will allow to concentrate all IO at one time which may be much faster and increase resources utilization.	Integer		`hive.max-outstanding-splits`	Y

Table 16. watsonx.data component: Presto (Java) catalog (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_hive_max_initial_splits`	This property describes how many splits may be initially created for a single query. The initial splits are created to allow better concurrency for small queries. Hive connector will create first `hive.max-initial-splits` splits with size of `hive.max-initial-split-size` instead of `hive.max-split-size`. Having this value higher will force more splits to have smaller size effectively increasing definition of what is considered small query in database.	Integer		`hive.max-initial-splits`	Y

Table 16. watsonx.data component: Presto (Java) catalog (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_hive_max_initial_split_size`	This property describes max size of each of initially created splits for a single query. The logic of initial splits is described in `hive.max-initial-splits` property. Changing this value changes what is considered small query. Higher value causes smaller parallelism for small queries. Lower value increases concurrency for them. This is max size, as the real size may be lower when end of blocks in single DataNode is reached.	Integer		`hive.max-initial-split-size`	Y

Table 16. watsonx.data component: Presto (Java) catalog (continued)

Property	**Description**	**Type**	**Default value / Default setting**	**System property**	**Restart containers required**
`presto_hive_max_split_size`	This property describes how many splits may be initially created for a single query. The initial splits are created to allow better concurrency for small queries. Hive connector will create first `hive.max-initial-splits` splits with size of `hive.max-initial-split-size` instead of `hive.max-split-size`. Having this value higher will force more splits to have smaller size effectively increasing definition of what is considered small query in database.	Integer		`hive.max-split-size`	Y

Table 16. watsonx.data component: Presto (Java) catalog (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_hive_split_loader_concurrency`	This property specifies the level of concurrency for loading data from Hive tables using the Presto (Java) Hive connector. It controls the number of concurrent split loader threads that Presto (Java) can utilize to fetch data in parallel from the Hive source. A higher concurrency value can improve data retrieval speed and resource utilization, particularly for large Hive queries, but it can also increase system resource consumption.	Integer		`hive.split-loader-concurrency`	Y

Table 16. watsonx.data component: Presto (Java) catalog (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_hive_pushdown_filter_enabled`	This property controls whether filter pushdown is enabled in the Presto (Java) Hive connector. Filter pushdown is a query optimization technique that allows Presto (Java) to push filtering conditions directly to the underlying Hive data source. When enabled, filtering conditions specified in SQL queries are evaluated as close to the data source as possible, reducing the amount of data that needs to be transferred to Presto (Java) for processing.	Integer		`hive.pushdown-filter-enabled`	Y

Table 16. watsonx.data component: Presto (Java) catalog (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
`presto_hive_node_selection_strategy`	Add configuration property `hive.node-selection-strategy` to choose `NodeSelectionStrategy`. When `SOFT_AFFINITY` is selected, scheduler will make the best effort to request the same worker to fetch the same file.	String		`hive.node-selection-strategy`	Y
`presto_hive_max_partitions_per_writers`	The maximum number of partitions permitted for a single writer.	Integer		`hive.max-partitions-per-writers`	
`presto_hive_metastore_timeout`	This parameter specifies the timeout duration for requests made to the Hive metastore. **Note:** It is applicable for version 1.1.4 and later of watsonx.data.	String (Duration)	`hive.metastore-timeout`	10s	Y
`hive_s3_max_error_retries`	This property allows to set the maximum number of retry attempts for S3 client operations in Hive.	Integer	`hive.s3.max-error-retries`	50	Y

Table 16. watsonx.data component: Presto (Java) catalog (continued)

Property	Description	Type	Default value / Default setting	System property	Restart containers required
hive_s3_connect_timeout	This property specifies the TCP connection timeout for S3 operations in Hive.	String (Duration)	hive.s3.connect-timeout	1m	Y
hive_s3_socket_timeout	This property sets the maximum time allowed for reading data from the socket for S3 operations in Hive.	String (Duration)	hive.s3.socket-timeout	2m	Y
hive_s3_max_connections	This property defines the maximum number of concurrent open connections permitted to S3 in Hive.	Integer	hive.s3.max-connections	5000	Y
hive_s3_max_client_retries	This property sets the maximum number of retry attempts for read operations for S3 in Hive.	Integer	hive.s3.max-client-retries	50	Y

Engine

Properties that can be customized for engine are listed here.

watsonx.data on Red Hat OpenShift

Table 17. watsonx.data component: Engine

Property	Description	Type	System property	Restart containers required	
presto_qhmm_enable	This parameter enables (or disables) the Query History Monitoring and Management (QHMM) service in an engine.	Boolean			

Customizing watsonx.data components

A project administrator can customize the properties for different IBM watsonx.data components.

watsonx.data on Red Hat OpenShift

About this task

The following procedure lists the various ways to customize the watsonx.data components.

Note: If a particular property is not run successfully, the default value from the t-shirt size is picked.

Note: The new deployment configuration is implemented in a rolling update.

Procedure

1. Deployment commands to update properties of different watsonx.data components:
   a) To update the **`Hive`**, **`MinIO`**, and **`Postgres`** properties, use one of the following deployment commands (i or ii).
      i) Run the following command to directly update a property and value:

```
oc patch wxd/lakehouse \
  --type=merge \
   -n ${PROJECT_CPD_INST_OPERANDS} \
   -p '{ "spec": { "<property_name>": "<value>" } }'
```

      Example:

```
oc patch wxd/lakehouse \
  --type=merge \
   -n ${PROJECT_CPD_INST_OPERANDS} \
   -p '{ "spec": { "minio_resources_requests_memory": "750Mi" } }'
```

      ii) Run the following command to edit and add new properties to the spec in the custom resource (CR):

```
oc edit wxd lakehouse
```

      Note: Wait for a few minutes for the rolling update after you save the properties.

   b) To update the **`Presto worker`** and **`Presto singlenode`**, use one of the deployment commands (ii or iii).
      i) Run the `oc get` command to get the engine name:

```
oc get wxdengine
```

```
oc get wxdengine
NAME                  AGE
lakehouse-presto-01   75m
```

      ii) Run the following command to directly update a property and value in the engine:

```
oc patch wxdengine/lakehouse-presto-01 \
  --type=merge \
   -n ${PROJECT_CPD_INST_OPERANDS} \
   -p '{ "spec": { "<property_name>": "<value>"  } }'
```

      Example:

```
oc patch wxdengine/lakehouse-presto-01 \
  --type=merge \
   -n ${PROJECT_CPD_INST_OPERANDS} \
   -p '{ "spec": { "presto_worker_replicas": "2"  } }'
```

iii) Run the following command to edit and add new properties to the spec in the custom resource (CR):

```
oc edit wxdengine  lakehouse-presto-01
```

Note: Wait for a few minutes for the rolling update after you save the properties.

iv) Run the following command to change the deployment from single node to multi-node.

```
oc edit -n ${PROJECT_CPD_INST_OPERANDS} wxdengine/lakehouse-presto-01
```

Parameter value:

- `LakehouseDeploymentType:` Edit the value to `multinode`.

- Include the parameter, `presto_worker_replicas:` *<y>*, where *<y>* is the number of worker replicas.

Note: Wait for a few minutes for the rolling update after you save the properties.

2. Restart pods after you update the properties of watsonx.data components:

   a) Run the following command to restart pods for applying the updated properties. This is necessary only for the properties where restart is set as Y. For more information, see the **Restart containers required** column of the table.

```
oc delete -n ${PROJECT_CPD_INST_OPERANDS} $(oc get pod -n ${PROJECT_CPD_INST_OPERANDS} -o
name | grep presto)
```

Delete properties from spec in CR

Procedure

Run the `oc edit` command to delete some properties from the spec in the custom resource (CR).

```
oc edit wxd lakehouse
```

Or

```
oc edit wxdengine  lakehouse-presto-01
```

Customizing max pool size in Hive Metastore

You can specify the maximum number of connections in a connection pool that is used by Hive Metastore.

The `hive_connection_pool_max_pool_size` value is set to 10 by default. The configured size is used by two connection pools (`TxnHandler` and `ObjectStore`). When configuring the maximum connection pool size, consider the number of metastore instances and the number of `HiveServer2` instances that are configured with the embedded metastore. For optimal performance, set the configuration to meet the following condition:

```
  (2 * pool_size * metastore_instances + 2 * pool_size * HS2_instances_with_embedded_metastore)
=
  (2 * physical_core_count + hard_disk_count)
```

Increasing this value increases the maximum number of Spark jobs and clients that can connect to Hive Metastore at the same time.

watsonx.data on Red Hat OpenShift

Procedure

- Overwrite the value of `hive_connection_pool_max_pool_size` using the following command. For more information, see Customizing watsonx.data components.

```
oc patch wxd/lakehouse --type=merge -n cpd-instance -p "{ \"spec\": {
          \"hive_connection_pool_max_pool_size\": "<value>"
} }"
```

Customization through API

You can customize watsonx.data properties through an API for Presto (Java) and Presto (C++). The following sections cover the API customizable properties.

watsonx.data on Red Hat OpenShift

Configuration properties

You can customize configuration properties through an API for Presto (Java) and Presto (C++). The following sections cover the API customizable properties.

watsonx.data on Red Hat OpenShift

Configuration properties for Presto (Java) - coordinator and worker nodes

You can customize the coordinator and worker configuration properties through an API for Presto (Java).

watsonx.data on Red Hat OpenShift

Table 18. Coordinator and worker configuration properties

Property name	Type	Validation added
`experimental.optimized-repartitioning`	Boolean	True or False
`experimental.reserved-pool-enabled`	Boolean	True or False
`fragment-result-cache.enabled`	Boolean	True or False
`fragment-result-cache.max-cached-entries`	Integer	`1000000`
`fragment-result-cache.base-directory`	String	file:///mnt/tmpfs/fragment
`fragment-result-cache.cache-ttl`	String	24h
`heap_dump_on_exceeded_memory_limit.enabled`	Boolean	True or False
`heap_dump_on_exceeded_memory_limit.file_directory`	String	Any string
`heap_dump_on_exceeded_memory_limit.max.number`	Integer	Limit {1, 1000}
`heap_dump_on_exceeded_memory_limit.max.size`	Integer	Limit {1, 1000}
`memory.heap-headroom-per-node`	String	Limit {1, 1e9}; supported values are numbers with or without units TB, MB, GB, B, KB
`node-scheduler.include-coordinator`	Boolean	True or False
`query.execution-policy`	String	Any string

Table 18. Coordinator and worker configuration properties (continued)

Property name	**Type**	**Validation added**
`query.low-memory-killer.policy`	String	Any string
`query.max-memory`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`query.max-memory-per-node`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`query.max-stage-count`	Integer	Limit {1, 1000}
`query.max-total-memory-per-node`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`query.min-expire-age`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`query.stage-count-warning-threshold`	Integer	Limit {1, 1000}
`task.concurrency`	Integer	Limit {1, 1000}
`join-distribution-type`	String	Value should be automatic or broadcast or partitioned
`exchange.client-threads`	Integer	Limit {1, 1000}
`exchange.http-client.max-connections`	Integer	Limit {1, 10000}
`exchange.http-client.max-connections-per-server`	Integer	Limit {1, 100000}
`http-server.log.max-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`http-server.log.max-history`	Integer	Limit {1, 100}
`http-server.threads.max`	Integer	Limit {1, 1000}
`join-max-broadcast-table-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`log.max-history`	Integer	Limit {1, 100}
`log.max-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`node-scheduler.max-pending-splits-per-task`	Integer	Limit {1, 3000}
`node-scheduler.max-splits-per-node`	Integer	Limit {1, 3000}

Table 18. Coordinator and worker configuration properties (continued)

Property name	**Type**	**Validation added**
`optimize-nulls-in-join`	Boolean	True or False
`optimizer.default-filter-factor-enabled`	Boolean	True or False
`optimizer.exploit-constraints`	Boolean	True or False
`optimizer.prefer-partial-aggregation`	Boolean	True or False
`query.client.timeout`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`query.max-execution-time`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`query.max-history`	Integer	Limit {1, 100}
`query.max-total-memory`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`query.min-schedule-split-batch-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`query.max-length`	Integer	Limit {1, 1000000}
`scale-writers`	Boolean	True or False
`shutdown.grace-period`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`sink.max-buffer-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`experimental.max-revocable-memory-per-node`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`experimental.max-spill-per-node`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`experimental.pushdown-dereference-enabled`	Boolean	True or False
`experimental.pushdown-subfields-enabled`	Boolean	True or False
`experimental.query-max-spill-per-node`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`experimental.spiller-max-used-space-threshold`	Float	Float 64

Table 18. Coordinator and worker configuration properties (continued)

Property name	Type	Validation added
`experimental.spiller-spill-path`	String	Any string
`http-server.max-request-header-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`experimental.internal-communication.max-task-update-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB

Configuration properties for Presto (C++) - worker nodes
You can customize the worker configuration properties through an API for Presto (C++).

watsonx.data on Red Hat OpenShift

Table 19. Worker configuration properties

Property name	Type	Validation added
`task.max-drivers-per-task`	Integer	Limit {1, 100}
`system-memory-gb`	Integer	Limit {1, 1000}
`query-memory-gb`	Integer	Limit {1, 10000}
`query.max-memory-per-node`	Integer	Limit {1, 1000}
`async-data-cache-enabled`	Boolean	True or False
`query-reserved-memory-gb`	Integer	Limit {1, 1000}
`system-mem-limit-gb`	Integer	Limit {1, 1e13}
`system-mem-shrink-gb`	Integer	Limit {1, 1e13}
`system-mem-pushback-enabled`	Boolean	True or False

Properties to be customized under support guidance

Though most of the properties can be customized by the watsonx.data administrators, the following properties must be customized under the guidance of the watsonx.data support team.

Table 20. Properties to be customized under support guidance

Property name	Type	Validation added
`runtime-metrics-collection-enabled`	Boolean	True or False
`system-mem-pushback-enabled`	Boolean	True or False
`system-mem-limit-gb`	Integer	Limit{1, 100000}
`system-mem-shrink-gb`	Integer	Limit{1, 100000}

Configuration properties for Presto (C++) - coordinator nodes
You can customize the coordinator configuration properties through an API for Presto (C++).

watsonx.data on Red Hat OpenShift

Table 21. Configuration coordinator properties

Property name	Type	Validation added
`http-server.log.max-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`http-server.log.max-history`	Integer	Limit {1, 1000}
`http-server.threads.max`	Integer	Limit {1, 1000}
`log.max-history`	Integer	Limit {1, 1000}
`log.max-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`node-scheduler.max-pending-splits-per-task`	Integer	Limit {1, 1000}
`node-scheduler.max-splits-per-node`	Integer	Limit {1, 1000}
`optimizer.joins-not-null-inference-strategy`	String	Any string
`optimizer.default-filter-factor-enabled`	Boolean	True or False
`optimizer.exploit-constraints`	Boolean	True or False
`optimizer.in-predicates-as-inner-joins-enabled`	Boolean	True or False
`optimizer.partial-aggregation-strategy`	String	Any string
`optimizer.prefer-partial-aggregation`	Boolean	True or False
`optimizer.infer-inequality-predicates`	Boolean	True or False
`optimizer.handle-complex-equi-joins`	Boolean	True or False
`optimizer.generate-domain-filters`	Boolean	True or False
`optimizer.size-based-join-flipping-enabled`	Boolean	True or False
`join-max-broadcast-table-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`query.client.timeout`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`query.execution-policy`	String	Any string

Table 21. Configuration coordinator properties (continued)

Property name	Type	Validation added
`query.low-memory-killer.policy`	String	Any string
`query.max-execution-time`	String	Limit {1, 1e13}; supported values are numbers with or without units m,s,ms,h
`query.max-history`	Integer	Limit {1, 10000}
`query.max-total-memory-per-node`	String	Limit {1, 1e13}; supported values are numbers with units TB, MB, GB, B, KB
`query.max-total-memory`	String	Limit {1, 1e13}; supported values are numbers with units TB, MB, GB, B, KB
`query.max-memory-per-node`	String	Limit {1, 1e13}; supported values are numbers with units TB, MB, GB, B, KB
`query.max-memory`	String	Limit {1, 1e13} supported values are numbers with units TB, MB, GB, B, KB
`query.max-stage-count`	Integer	Limit {1, 10000}
`query.min-expire-age`	String	supported values are numbers with or without units m, s, ms, h
`query.min-schedule-split-batch-size`	Integer	Limit {1, 10000}
`query.stage-count-warning-threshold`	Integer	Limit {1, 10000}
`query.max-length`	Integer	Limit {1, 10000}
`scale-writers`	Boolean	True or False
`scheduler.http-client.max-requests-queued-per-destination`	Integer	Limit {1, 100000}
`shutdown.grace-period`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h

JVM properties

You can customize JVM properties through an API for Presto (Java).
The following section covers the API customizable properties.

watsonx.data on Red Hat OpenShift

JVM properties for Presto (Java) - coordinator and worker nodes
You can customize the coordinator and worker JVM properties through an API for Presto (Java).

watsonx.data on Red Hat OpenShift

Table 22. Coordinator and worker JVM properties		
Property name	**Type**	**Validation added**
`-XX:G1HeapRegionSize`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB, M, G, B
`-XX:ReservedCodeCacheSize`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB, M, G, B
`-Djdk.nio.maxCachedBufferSize`	Integer	Limit {1, 1e13}
`-Dalluxio.user.app.id`	String	Any string
`-Duser.timezone`	String	Any string
`-xgc`	String	Any string
`-Xmx6G`	String	Any string

Catalog properties

You can customize catalog properties through an API for Presto (Java) and Presto (C++).
The following sections cover the API customizable properties.

watsonx.data on Red Hat OpenShift

Catalog properties for Presto (Java)
You can customize the catalog properties through an API for Presto (Java).

watsonx.data on Red Hat OpenShift

Table 23. Catalog properties		
Property name	**Type**	**Validation added**
`cache.enabled`	Boolean	True or False
`cache.base-directory`	String	Any string
`cache.type`	String	Any string
`cache.alluxio.max-cache-size`	String	Limit {1, 1e13}; supported values are numbers with or without units TB, MB, GB, B, KB
`hive.partition-statistics-based-optimization-enabled`	Boolean	True or False
`hive.metastore-cache-scope`	String	Any string
`hive.metastore-cache-ttl`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`hive.metastore-refreshIntegererval`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h, d
`hive.metastore-cache-maximum-size`	Integer	Limit {1, 1000}

Table 23. Catalog properties (continued)

Property name	**Type**	**Validation added**
`hive.partition-versioning-enabled`	Boolean	True or False
`hive.file-status-cache-expire-time`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`hive.file-status-cache-size`	Integer	Limit {1, 10000000000}
`hive.file-status-cache-tables`	String	Any string
`<catalog-name>.orc.file-tail-cache-enabled`	Boolean	True or False
`<catalog-name>.orc.file-tail-cache-size`	Integer	Limit{1, 1000000}
`<catalog-name>.orc.file-tail-cache-ttl-since-last-access`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`<catalog-name>.orc.stripe-metadata-cache-enabled`	Boolean	True or False
`<catalog-name>.orc.stripe-footer-cache-size`	Integer	Limit {1, 1000}
`<catalog-name>.orc.stripe-footer-cache-ttl-since-last-access`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`<catalog-name>.orc.stripe-stream-cache-size`	Integer	Limit {1, 1000}
`<catalog-name>.orc.stripe-stream-cache-ttl-since-last-access`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`hive.orc.use-column-names`	Boolean	True or False
`<catalog-name>.parquet.metadata-cache-enabled`	Boolean	True or False
`<catalog-name>.parquet.metadata-cache-size`	Integer	Limit {1, 1000}
`<catalog-name>.parquet.metadata-cache-ttl-since-last-access`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h

Table 23. Catalog properties (continued)

Property name	Type	Validation added
`hive.parquet.use-column-names`	Boolean	True or False
`hive.parquet-batch-read-optimization-enabled`	Boolean	True or False
`hive.node-selection-strategy`	String	Any string
`hive.max-outstanding-splits`	Integer	Limit {1, 1000}
`hive.max-initial-splits`	Integer	Limit {1, 1000}
`hive.max-initial-split-size`	Integer	Limit {1, 1000}
`hive.max-split-size`	Integer	Limit {1, 1000}
`hive.split-loader-concurrency`	Integer	Limit {1, 1000}
`hive.pushdown-filter-enabled`	Boolean	True or False
`hive.max-partitions-per-writers`	Integer	Limit {1, 10000}
`hive.s3.max-error-retries`	Integer	Limit {1, 100}
`hive.s3.connect-timeout`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`hive.s3.socket-timeout`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`hive.s3.max-connections`	Integer	Limit {1, 10000}
`hive.s3.max-client-retries`	Integer	Limit {1, 100}
`hive.collect-column-statistics-on-write`	Boolean	True or False
`hive.non-managed-table-creates-enabled`	Boolean	True or False
`hive.s3select-pushdown.enabled`	Boolean	True or False
`hive.recursive-directories`	Boolean	True or False
`hive.allow-rename-table`	Boolean	True or False
`hive.allow-add-column`	Boolean	True or False
`hive.allow-drop-column`	Boolean	True or False
`hive.allow-rename-column`	Boolean	True or False

Table 23. Catalog properties (continued)

Property name	**Type**	**Validation added**
`hive.metastore-timeout`	String	Limit {1, 1e13}; supported values are numbers with or without units m, s, ms, h
`ignore-unsupported-datatypes`	Boolean	True or false values. For more information, see "ignore-unsupported-datatypes" on page 277

ignore-unsupported-datatypes

Presto skips columns with unsupported data types when it retrieves data from a database. The `ignore-unsupported-datatypes` property controls this behavior, which is set to **`true`** by default, causing unsupported columns to be skipped. You can set the property to **`false`** to make Presto raise an error when it encounters unsupported data types instead of omitting the columns. So, tables with unsupported columns do not return incomplete or inaccurate data.

The property is available for the following databases:

- Amazon Redshift
- IBM Db2
- Informix
- MySQL
- Oracle
- PostgreSQL
- SQL Server
- SingleStore
- Snowflake
- Teradata
- SAP HANA

Catalog properties for Presto (C++)
You can customize the catalog properties through an API for Presto (C++).

watsonx.data on Red Hat OpenShift

Table 24. Catalog properties

Property name	**Type**	**Validation added**
`hive.orc.use-column-names`	String	Any string
`max-partitions-per-writers`	Integer	Limit{1, 100000}

Velox properties

You can customize Velox properties through an API for Presto (C++).
The following section covers the API customizable properties.

watsonx.data on Red Hat OpenShift

Velox properties for Presto (C++)
You can customize the Velox properties through an API for Presto (C++).

watsonx.data on Red Hat OpenShift

Table 25. Velox properties		
Property name	**Type**	**Validation added**
`task_writer_count`	Integer	Limit {1, 1000}
`max_split_preload_per_driver`	Integer	Limit {1, 1000}

Resource group properties

You can format the resource group JSON file to match with the sample resource group file that you can download from the console.
The following sections cover the resource group properties that you can configure.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Table 26. Main properties

Property name	Type	Required/Optional	Description
`rootGroups`	Array	Required	Defines the specifications for resource groups.
`selectors`	Array	Required	Specifies the selector configurations for the resource groups.
`cpuQuotaPeriod`	String	Optional	Specifies the CPU quota period. **Pattern**: This property must match with the pattern `^\\d+(\\.\\d+)?[smhd]$`.

Table 27. `rootGroups` properties

Property name	Type	Required/Optional	Description
name	String	Required	Name of the resource group. Following are the naming conditions: • If the name contains `{`, `}`, or `$`, the name must be in the pattern `\$\{([a-zA-Z][a-zA-Z0-9]*)\}`. • Blank spaces are allowed in the name. • The name cannot contain a period (.). • You cannot have the same name for two sibling groups. You can have the same name for a root group and a sub group.
`softMemoryLimit`	String	Required	Specifies the soft memory limit. • Minimum value: 0% • Maximum value: 999%

Table 27. `rootGroups` properties (continued)

Property name	**Type**	**Required/Optional**	**Description**
`maxQueued`	Integer	Optional	Specifies the maximum number of queued requests. • Minimum value: 0 • Maximum value: 2147483647 **Note:** The value must not start with `0`. For example, `01`, `05`. Specify the values as `1`, `5`. `0` alone is a valid value.
`softConcurrencyLimit`	Integer or null	Optional	Specifies the soft concurrency limit. • Minimum value: 0 • Maximum value: 2147483647 **Note:** The value must not start with `0`. For example, `01`, `05`. Specify the values as `1`, `5`. `0` alone is a valid value.
`maxRunning`	Integer or null	Optional	Specifies the maximum running count. • Minimum value: 0 • Maximum value: 2147483647 **Note:** The value must not start with `0`. For example, `01`, `05`. Specify the values as `1`, `5`. `0` alone is a valid value.
`hardConcurrencyLimit`	Integer	Required **Note:** If `maxRunning` value is available, `hardConcurrencyLimit` takes up that value. If `maxRunning` is not set, you must set a value for `hardConcurrencyLimit`.	Specifies the hard concurrency limit. • Minimum value: 0 • Maximum value: 2147483647 **Note:** The value must be greater than or equal to `softConcurrencyLimit`. It must not start with `0`. For example, `01`, `05`. Specify the values as `1`, `5`. `0` alone is a valid value.
`schedulingPolicy`	String Following are the available values: • `fair` • `weighted` • `weighted_fair` • `query_priority` These values are not case-sensitive.	Optional	Specifies the scheduling policy.

Table 27. rootGroups properties (continued)

Property name	Type	Required/Optional	Description
schedulingWeight	Integer or null	Optional	Specifies the scheduling weight. Allowed values (minimum to maximum): 1 to 2147483647. **Note:** The value must not start with 0. For example, 01, 05. Specify the values as 1, 5. 0 alone is a valid value. If a subgroup has schedulingWeight, all of the corresponding siblings in that subgroup must have schedulingWeight.
subGroups	Array or null	Optional	Specifies the subgroups within the resource group. Subgroups have the same rules for different properties as in the resource group.
jmxExport	Boolean or null	Optional	Indicates whether JMX export is enabled.
softCpuLimit	String	Optional	Specifies the soft CPU limit. It must match the pattern ^\\d+(\\.\\d+)?[smhd]$. If hardCpuLimit is defined, the value of softCpuLimit must be less than or equal to hardCpuLimit.
hardCpuLimit	String	Optional	Specifies the hard CPU limit. It must match the pattern ^\d+(\.\d+)?[smhd]$.

Table 27. `rootGroups` properties (continued)			
Property name	**Type**	**Required/Optional**	**Description**

`perQueryLimits`	Object	Optional	Specifies per-query limits.
			
			Example:
			
			`"perQueryLimits":`
			`{ "executionTimeLimit": "30m",`
			`"totalMemoryLimit": "2GB",`
			`"cpuTimeLimit": "1h" }`
			
			The object can have one, two, or three of the following limits:
			
			Note: Negative values are not allowed for any of the following limits.
			
			executionTimeLimit
			• **Type**: String
			• **Pattern**: This property must match with the pattern `^\\d+(\\.\\d+)?[smhd]$`.
			
			totalMemoryLimit
			• **Type**: String
			• **Pattern**: This property must match with the pattern `^\s*(\d+(?:\.\d+)?)\s*([a-zA-Z]+)\s*$`.
			
			cpuTimeLimit
			• **Type**: String
			• **Pattern**: This property must match with the pattern `^\\d+(\\.\\d+)?[smhd]$`.

`workersPerQueryLimit`	Integer or null	Optional	Specifies the workers per query limit. Allowed values (minimum to maximum): -2147483648 to 2147483647. **Note:** The value must be greater than or equal to `softConcurrencyLimit`. It must not start with `0`. For example, `01`, `05`. Specify the values as `1`, `5`. `0` alone is a valid value.

Table 28. `selectors` properties

Property name	**Type**	**Required/Optional**	**Description**
`user`	String or null Strings can have any valid regular expression (`.*`).	Optional	Specifies the user regex pattern.
`source`	String or null Strings can have any valid regular expression (`.*`).	Optional	Specifies the source regex pattern.
`queryType`	String or null Possible values: • `data_definition` • `delete` • `describe` • `explain` • `analyze` • `insert` • `select` • `control` • `update` These values are not case-sensitive.	Optional	Specifies the query type.
`clientTags`	List of strings or null **Example**:`"clientTags": ["resourceGroup1",”resourceGroup2”]` Strings can have any valid regular expression (`.*`).	Optional	Specifies the client tags.

Table 28. `selectors` properties (continued)

Property name	Type	Required/ Optional	Description
`selectorResourceEstimate`	Object or null	Optional	Specifies the selector resource estimate. **Example**: `"selectorResourceEstimate": {"executionTime": {"min": "5m", "max": "10m"}, "cpuTime": {"min": "30m", "max": "1h"}, "peakMemory": {"min": "512MB", "max": "2GB"}}` The object can have one, two, or three of the following limits. You can also use the `min`, `max`, or both of the parameters for all of the three limits. For example, `"selectorResourceEstimate": {"executionTime": {"min": "5m"} }`. **Note:** Negative values are not allowed for any of the following limits. **executionTime** **min** • **Type**: String • **Pattern**: This property must match with the pattern `^\\d+(\\.\\d+)?[smhd]$`. **max** • **Type**: String • **Pattern**: This property must match with the pattern `^\\d+(\\.\\d+)?[smhd]$`. **peakMemory** **min** • **Type**: String • **Pattern**: This property must match with the pattern `^\s*(\d+(?:\.\d+)?)\s*([a-zA-Z]+)\s*$`. **max** • **Type**: String • **Pattern**: This property must match with the pattern `^\s*(\d+(?:\.\d+)?)\s*([a-zA-Z]+)\s*$`. **cpuTime** **min** • **Type**: String • **Pattern**: This property must match with the pattern `^\\d+(\\.\\d+)?[smhd]$`. **max** • **Type**: String • **Pattern**: This property must match with the pattern `^\\d+(\\.\\d+)?[smhd]$`.

Table 28. selectors properties (continued)

Property name	Type	Required/ Optional	Description
clientInfo	String or null Strings can have any valid regular expression (.*).	Optional	Specifies the client info regex pattern.
schema	String or null Strings can have any valid regular expression (.*).	Optional	Specifies the schema.
principal	String or null Strings can have any valid regular expression (.*).	Optional	Specifies the principal regex pattern.
group	String	Required	The group name must be from the available names in the resource group. For redirecting to a subgroup, use `"group": "groupname.subgroupname"`. You can also use dynamic values like ${SOURCE}, ${USER}, and ${SCHEMA} as used in root group names. You cannot have `null` in `group`.

Note: In the `source` and `user` regex, you can use the provided name in the format `(?<sampleName>.*)` as a dynamic group name. For example:

```
{
            "source": "(?<sampleName>.*)",
            "clientTags": [
                "hipri"
            ],
            "group": "bi-${sampleName}"
}and there is a group as ,{
            "name": "bi-${sampleName}",
            "softMemoryLimit": "80%",
            "hardConcurrencyLimit": 100,
            "maxQueued": 1000,
            "schedulingPolicy": "weighted",
            "jmxExport": true
}
```

In this example, *sampleName* is a dynamic value. Special characters are not allowed in the name. You can add other values like ${SOURCE}, ${USER}, or ${SCHEMA}. The `group` name is case-sensitive. You can have values before and after the dynamic variable. For example, `abc-${SOURCE}` or `${toolname}-xyz`.

Table 29. cpuQuotaPeriod properties

Property name	Type	Required/Optional	Description
cpuQuotaPeriod	String	Optional	Specifies the CPU quota period. **Pattern**: This property must match with the pattern `^\\d+(\\.\\d+)?[smhd]$`.

Chapter 4. watsonx.data client package

The client package consists of utilities and pre-packaged libraries to access and develop applications to work with IBM watsonx.data.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Installing `ibm-lh-client`

The `ibm-lh-client` package includes convenient utilities and pre-packaged libraries to access the IBM watsonx.data, and to develop applications that work with the watsonx.data.

Before you begin

- Ensure that you have your entitlement key to access the IBM Entitled Registry. After you obtain the entitlement key from the container software library, you can log in to the registry with the key and pull the runtime images to your local system.
- Make sure to meet the following system requirements and install the most recent version of `Docker` or `Podman` on your system.

Operating system	x86-64	Docker / Podman / Colima installation instructions
Linux	√	Docker Podman
Windows	√	Docker Podman
Mac OS x86	√	Docker Podman
Mac with Apple Silicon with Rosetta Emulation		Docker Colima

  Note: Make sure to meet the prerequisites for installing the watsonx.data client package on Mac with Apple Silicon with Rosetta Emulation.
- If you are using SUSE Linux, `podman` is only available for version SLES 15.4. Upgrade the system first (`zypper dist-upgrade`) before installing the dependencies that are not provided but are needed for the installation of `docker` or `podman`.
  - For Docker: `sysuser-shadow` in SLES 12.5 and `catatonit` in SLES 15.5.
  - For Podman: `fuse-overlayfs` in SLES 15.4.

  Note: Packages can be downloaded from the official site.
- Ensure to add the `podman-plugins` for the DNS server of the Podman network to work properly.

  ```
  yum install -y podman-plugins
  ```

  For SUSE, install cni-plugin-dnsname instead.

  If you add the `podman-plugins` after you start the watsonx.data service, delete the Podman network `ibm-lh-network` and restart the watsonx.data service.

Tip: Podman provides a Docker-compatible command-line front end. You can alias the Docker CLI with the `alias docker=podman` shell command.

Important: It is important to set up and run the client utilities as a non-root user. For more information about granting access to non-root users to work with the container engines, see Manage Docker as a non-root user and Rootless containers using Podman. To ensure user-specific access, each user needs to extract and set up the `ibm-lh-client` package in a private subdirectory within their own home directory.

Procedure

1. Set up the installation directory and environment variables

   a) Set up the work directory.

   ```
   mkdir <install_directory>
   cd <install_directory>
   ```

   b) Set the environment variables:

   ```
   export LH_ROOT_DIR=<install_directory>
   export LH_RELEASE_TAG=latest
   export IBM_LH_TOOLBOX=cp.icr.io/cpopen/watsonx-data/ibm-lakehouse-toolbox:$LH_RELEASE_TAG
   export LH_REGISTRY=cp.icr.io/cp/watsonx-data
   export PROD_USER=cp
   export IBM_ENTITLEMENT_KEY=<ibm_entitlement_key>
   export IBM_ICR_IO=cp.icr.io
   ```

   Note: The environment variable LANG is passed into the containers to support Full width and MBCS characters. If the LANG is not set, the default is considered to be **`en_US.UTF-8`**. Update the `LH_CLIENT_LANG` in `etc/local_config.env` to change the LANG chosen during installation.

   Note: Use the following table to identify the `LH_RELEASE_TAG` value:

   Table 30. Release tags with Cloud Pak for Data versions

Cloud Pak for Data version	Service instance version
5.0.2	`v2.0.2` (`latest`)
5.0.1	`v2.0.1`
5.0.0	`v2.0.0`
4.8.5	`v1.1.4`
4.8.4	`v1.1.3`
4.8.3	`v1.1.2`
4.8.1	`v1.1.1`
4.8.0	`v1.1.0`

   The `latest` tag is set to `v2.0.2`

   If you are using Docker, run the following command:

   ```
   export DOCKER_EXE=docker
   ```

   If you are using Podman, run the following command:

   ```
   export DOCKER_EXE=podman
   ```

2. Pull the watsonx.data client package and copy it to the host system:

   ```
   $DOCKER_EXE pull $IBM_LH_TOOLBOX
   id=$($DOCKER_EXE create $IBM_LH_TOOLBOX)
   $DOCKER_EXE cp $id:/opt - > /tmp/pkg.tar
   ```

```
$DOCKER_EXE rm $id
id=
```

3. Extract the watsonx.dataclient `pkg.tar` file in to the `/tmp` directory. Verify that the checksum is correct by comparing the checksum in `bom.txt` and the `cksum` command output. For example, run the following command:

   ```
   tar -xf /tmp/pkg.tar -C /tmp
   cat /tmp/opt/bom.txt
   cksum /tmp/opt/*/*
   tar -xf /tmp/opt/client/ibm-lh-client-*.tgz -C $LH_ROOT_DIR
   ```

4. Authenticate with the registry:

   ```
   $DOCKER_EXE login ${IBM_ICR_IO} \
   --username=${PROD_USER} \
   --password=${IBM_ENTITLEMENT_KEY}
   ```

5. Run the setup script.

   ```
   $LH_ROOT_DIR/ibm-lh-client/bin/setup --license_acceptance=y --runtime=$DOCKER_EXE
   ```

Working with client utilities

The watsonx.data client package includes utilities for SSL certificate management and engine management. These utilities are automatically configured to work with the installed Presto engine.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

`ibm-lh-client` package utilities

The `ibm-lh-client` package includes utilities for SSL certificate management and engine management.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

To use the utilities, the `ibm-lh-client` package must be installed on your system. For more information, see Installing `ibm-lh-client` package

The `ibm-lh-client` supports the following utilities:

- `cert-mgmt`
- `presto-cli`
- `presto-run`
- `python-run`
- `dev-sandbox`
- `ibm-lh`
- `connect-lh`

`cert-mgmt`

Add or remove Presto engine connection certificates to `ibm-lh-client` truststore.

Syntax

Add a Presto engine connection certificate:

```
./cert-mgmt --op=add --host=<host> --port=<port>
```

Remove a Presto engine connection certificate:

```
./cert-mgmt --op=remove --host=<host> --port=<port>
```

`presto-cli`

Start an interactive session with the Presto engine.

Syntax

Use the following command to start an interactive session:

```
bin/presto-cli --engine=<engine name> --catalog=tpch
```

For more information on how to use `presto-cli`, refer Chapter 7, "Connecting to a Presto server," on page 450

`presto-run`

Run SQL in a non-interactive manner.

Syntax

Use the bin/presto-run utility to run SQL in a noninteractive manner:

```
bin/presto-run --engine=<engine-name> --catalog=tpch <<< "select * from tiny.customer limit 10;"
```

Run SQL from a file from your LH_SANDBOX_DIR mount.

```
bin/presto-run --engine=<engine-name> -f <path of sql file directory>
```

For more information on how to use `presto-run`, refer "Running SQL queries by using ibm-lh-client" on page 291.

`python-run`

It provides a way to run python scripts.

Syntax

Start an interactive python session.

```
bin/python-run --engine=<engine-name>
```

Note: The engine details are available as environment variables (ENG_HOST, ENG_PORT, ENG_USERNAME, ENG_PASSWORD). For example,`os.environ['ENG_PASSWORD']` provides the saved password for the selected engine.

Run a python script from your LH_SANDBOX_DIR mount.

```
bin/python-run --engine=<lakehouse-engine-name> <python-script-in-sandbox>
```

For more information on how to use `python-run`, refer "Running Python scripts by using ibm-lh-client " on page 292

`dev-sandbox`

Provides a containerized environment with useful utilities and python modules to help explore the watsonx.data.

Syntax

Set the sandbox that starts an interactive bash shell from which you can run other commands:

```
bin/dev-sandbox --engine=<engine-name>
```

For more information on how to use `dev-sandbox`, refer "Running Python scripts in sandbox container by using ibm-lh-client" on page 292

ibm-lh

It is a terminal-based interface that is designed to facilitate interaction with watsonx.data resources. For more information on how to use `ibm-lh`, refer `ibm-lh commands and usage`

connect-lh

Manage the connections for your watsonx.data engine. Use this command to add, remove, list, or show details of a Presto engine connection.

Syntax

Add a Presto engine connection:

```
connect-lh --op=add \
--name=<alias> \
--host=<host> \
--port=<port> \
--username=<username> \
--password=<password>
```

Remove a Presto engine connection:

```
connect-lh --op=remove --name=<alias>
```

Show the connection details for a specific connection:

```
connect-lh --op=details --name=<alias>
```

List all configured engine connections:

```
connect-lh --op=list
```

where:

- `name`: The given name or alias to identify the connection.
- `host` for connecting `presto-cli`:
  - The hostname of the Presto server in watsonx.data Developer edition.
  - The exposed secure route for the Presto server in watsonx.data software or watsonx.data on Cloud Pak for Data (CPD). For more information about exposing secure route, see Exposing secure route to Presto server.
  - The hostname of the Presto server from the web console in watsonx.data on IBM cloud. From **Infrastructure Manager**, select the engine you want to connect and copy the host. The copied host contains both the host and port details. Use the host details as the host name of the Presto server.
- `host` for connecting `ibm-lh`:
  - The hostname of the Presto server in watsonx.data Developer edition.
  - The secure route for the Presto server in watsonx.data software or watsonx.data on Cloud Pak for Data (CPD).
  - The hostname of the cloud server in watsonx.data on IBM cloud. (For example : us-south.lakehouse.dev.cloud.ibm.com).
- `port` for connecting `presto-cli`:
  - The default port is 9443 for watsonx.data Developer edition.
  - The default port is 443 for watsonx.data software and watsonx.data on Cloud Pak for Data (CPD).

  - The port number of the Presto server from the web console in watsonx.data on IBM cloud. From **Infrastructure Manager**, select the engine you want to connect and copy the host. The copied host contains both the host and port details. Use the port details as the port number of the Presto server.
- `port` for connecting `ibm-lh`:
  - The default port is 9443 for watsonx.data Developer edition.
  - The default port is 443 for watsonx.data software, watsonx.data on Cloud Pak for Data (CPD), and watsonx.data on IBM cloud
- `username`: The username for authentication.
  - The default `username` is `ibmlhadmin` for watsonx.data Developer edition.
  - Username is the user created for watsonx.data software or watsonx.data on Cloud Pak for Data (CPD). The default `username` is `admin`. If IAM is enabled, the default user is `cpadmin`.
  - Username can either be `ibmlhapikey` or `ibmlhtoken` in watsonx.data on IBM cloud. For more information on how to use `ibmlhapikey` or `ibmlhtoken`, see Connect to Presto server.
- `password`: The password for authentication.
  - The default `password` is `password` for watsonx.data Developer edition.
  - To get the default password for default `admin` or `cpadmin` user in watsonx.data software, run the following command:

    ```
    ibm-lakehouse-manage get-cpd-instance-details
    ```

  - To get the default password for default `admin` or `cpadmin` user in watsonx.data on Cloud Pak for Data (CPD), run the following command:

    ```
    cpd-cli manage get-cpd-instance-details \
    --cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
    --get_admin_initial_credentials=true
    ```

    Note: If the `--password` option is not specified, password is prompted.
  - Password can either be `apikey` or `apitoken` in watsonx.data on IBM cloud. For more information see, Get API key and Get API token.

Sandbox directory

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

After installing `ibm-lh` client, you can introduce a directory for your own use with the containerized utilities. Such a 'sandbox' directory can include your SQL files, python scripts and other files that can be referenced when you use the utilities.

To mount your own directory, set the `LH_SANDBOX_DIR` environment variable to the sandbox directory path and create a sample SQL file or python script to be run in that directory. The following steps describe the setup of sandbox directory.

- Create the sandbox directory. For example:

  ```
  mkdir /home/jmouse/mysandbox
  ```

- Set the environment variable to the sandbox path created in the previous step. For example:

  ```
  export LH_SANDBOX_DIR=/home/jmouse/mysandbox
  ```

- You can then run the python script or SQL file in that directory.

  For more information, see:

  - "Running SQL queries by using ibm-lh-client" on page 291
  - "Running Python scripts by using ibm-lh-client " on page 292

– "Running Python scripts in sandbox container by using ibm-lh-client" on page 292

Note: It is advised that you create a new directory that is solely meant for sharing files inside these containers. Avoid selecting directories such as your HOME as a Sandbox directory. Mounting directories such as your HOME can cause file permission changes on existing files on some operating systems or with specific container runtime.

Running SQL queries by using `ibm-lh-client`

You can run SQL queries by using Presto engine. The `ibm-lh-client` interacts with Presto engine in the following three ways. You can use any of the following methods to run the SQL query.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

About this task

Complete the following steps to work with the Presto engine host:

Procedure

1. Establish connection with Presto engine by using the `ibm-lh-client`. For more information about installation, see "Installing ibm-lh-client" on page 285
2. Now, you can run SQL queries by using Presto engine. The `ibm-lh-client` interacts with Presto engine in the following three ways. You can use any of the following methods to run the SQL query.

   Using an interactive session

   Use the following command to run SQL query in an interactive manner:

   ```
   bin/presto-cli --engine=ENGINE_NAME --catalog=tpch
   ```

   Parameter value:

   - ENGINE_NAME: Enter the engine name that you added.
   - tpch: Specify the SQL query that you need to run.

   Using a noninteractive session

   Use the following command to run SQL query in a noninteractive manner:

   ```
   bin/presto-run --engine=ENGINE_NAME --catalog=tpch <<< "select * from tiny.customer
   limit 10;"
   ```

   Parameter value:

   - ENGINE_NAME: Enter the engine name that you added.
   - tpch: Specify the SQL query that you need to run.

   Using an SQL file

   Set the LH_SANDBOX_DIR mount by referring "Sandbox directory" on page 290

   Use the following command to run an SQL file from your LH_SANDBOX_DIR mount:

   ```
   bin/presto-run --engine=ENGINE_NAME -f PATH_SANDBOX_DIR_SQL_FILE
   ```

   Parameter value:

   - ENGINE_NAME: Enter the engine name that you added.
   - PATH_SANDBOX_DIR_SQL_FILE: Specify the SQL query file location available in your sandbox directory. For example: /home/jmouse/mysandbox/sample.sql

Running Python scripts by using `ibm-lh-client`

The `ibm-lh-client` supports the execution of Python scripts.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

About this task

Perform the following steps to work with the Presto engine host:

Procedure

The `ibm-lh-client` also supports the Python scripts. You can run Python scripts by using the following command.

Start an interactive Python session

Use the following command to start a session to run Python scripts:

```
bin/python-run --engine=<ENGINE_NAME>
```

Parameter value:

- ENGINE_NAME: Enter the engine name that you added.

Run a Python script

Set the LH_SANDBOX_DIR mount by referring "Sandbox directory" on page 290

Use the following command to run python file from your LH_SANDBOX_DIR mount:

```
bin/python-run --engine=<ENGINE_NAME> <PATH_SANDBOX_DIR_PYTHON_FILE>
```

Parameter value:

- ENGINE_NAME: Enter the engine name that you added.
- PATH_SANDBOX_DIR_PYTHON_FILE: Specify the Python query location available in your sandbox directory. For example: /home/jmouse/mysandbox/sample.py

Running Python scripts in sandbox container by using `ibm-lh-client`

You can run Python files from inside a sandbox container to develop various functions by using `ibm-lh-client` and connecting to a Presto (Java) engine.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

About this task

To work with the sandbox container, complete the following steps:

Procedure

1. Set the variable "LH_SANDBOX_DIR" with the folder path. For example: `export LH_SANDBOX_DIR=/home/jmouse/mysandbox`. For more information, see on how to setup sandbox directory, see "Sandbox directory" on page 290

```
export LH_SANDBOX_DIR=<folder_path>
```

2. Run the following command to start the sandbox environment.

```
 bin/dev-sandbox --engine=<ENGINE_NAME>
```

Parameter value:

- ENGINE_NAME: Enter the engine name that you added.

Note: The environment variables are set inside the sandbox container to help you with scripting and to avoid hardcoding.

3. Run the following commands to get the details of the engine host and engine password:

   a. In the sandbox environment, run the following command:

   ```
   ./python-run --engine=<ENGINE_NAME>
   ```

   b. In the python environment, import os and check for the engine host and engine password from the following commands:

   ```
   import os
   os.environ['ENG_HOST']
   os.environ['ENG_PASSWORD']
   ```

4. Use sandbox to run python file from the sandbox directory by using the command `python3 <path of python file to be run>` as shown in the following example where 'sample.py' is the python file:

   ```
   python3 /home/jmouse/mysandbox/sample.py
   ```

   Note: Type 'exit' to exit from the sandbox container.

Setting up the `ibm-lh` utility

The `ibm-lh` utility provides a terminal interface for issuing the IBM watsonx.data component management and ingestion commands from a client system where the utility is installed.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Before you begin

The `ibm-lh` utility is included in the `ibm-lh-client` package. Make sure that `ibm-lh-client` is installed on your system. For more information, see Installing `ibm-lh-client` package.

About this task

The `ibm-lh` utility communicates with the Presto server over an SSL connection. To facilitate SSL communication, you must import the SSL certificate into the utility's truststore and configure the `ibm-lh` utility.

Procedure

To set up `ibm-lh` utility, complete the following steps:

1. Add the Presto engine connection details.

   ```
   connect-lh --op=add \
   --name=<alias> \
   --host=<host> \
   --port=<port> \
   --username=<username> \
   --password=<password>
   ```

   For more information, see "connect-lh" on page 289.

2. Add the SSL certificate to your host and port.

   ```
   ./cert-mgmt --op=add --host=<hostname> --port=<port>
   ```

   Note: If you are connecting to IBM watsonx.data on IBM Cloud, you can skip this step.

For more information, see "cert-mgmt" on page 287.

3. Configure connectivity to watsonx.data.

   IBM watsonx.data developer edition Run the following command:

   ```
   ./ibm-lh config add_dev --name <instance_name> --host <hostname> --port <port>
   ```

   IBM watsonx.data stand-alone Run the following command:

   ```
   ./ibm-lh config add_cpd --name <instance_name> --host <hostname> --port <port>
   ```

   IBM watsonx.data on Cloud Pak for Data Run the following command:

   ```
   ./ibm-lh config add_cpd --name <instance_name> --host <hostname> --port <port>
   ```

   IBM watsonx.data on IBM Cloud Run the following command:

   ```
   ./ibm-lh config add_saas --name <instance_name> --crn <crn> --host <hostname> --port <port>
   ```

   where, `crn` is the CRN of your watsonx.data instance on IBM Cloud.

`ibm-lh` commands and usage

The `ibm-lh` command-line utility is a terminal-based interface that is designed to facilitate interaction with watsonx.data resources.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

The `ibm-lh` CLI utility supports the following commands:

- `ibm-lh engine`
- `ibm-lh config`
- `ibm-lh database`
- `ibm-lh bucket`
- `ibm-lh data-copy`
- `ibm-lh table-maint`

`ibm-lh engine`

Work with engine resources.

Syntax

```
./ibm-lh engine
```

Command	Description
`ibm-lh engine ls`	List all engines.
`ibm-lh engine info`	Get information about a specific engine.
`ibm-lh engine add`	Create an engine.
`ibm-lh engine rm`	Delete the specified engine.
`ibm-lh engine attach`	Associate a catalog with an engine.
`ibm-lh engine detach`	Dissociate a catalog from an engine.

`ibm-lh config`

Configure connectivity to the watsonx.data API endpoints.

Syntax

```
./ibm-lh config
```

Command	Description
`ibm-lh config add_dev`	Add a new hostname and port. **watsonx.data Developer edition**
`ibm-lh config add_cpd`	Add a new hostname and port. **watsonx.data on Red Hat OpenShift** **watsonx.data on Cloud Pak for Data**
`ibm-lh config add_saas`	Add a new hostname and port. **watsonx.data on IBM Cloud**
`ibm-lh config rm`	Remove an existing hostname and port.
`ibm-lh config ls`	List all instances.
`ibm-lh config select`	Select an instance.
`ibm-lh config current`	Show the currently selected instance.

`ibm-lh database`

Register and work with external databases.

Syntax

```
./ibm-lh database
```

Command	Description
`ibm-lh database add`	Add a database.
`ibm-lh database ls`	List all databases.
`ibm-lh database info`	Get information about a specified database.
`ibm-lh database rm`	Delete a database.

`ibm-lh bucket`

Register and work with object storage buckets.

Syntax

```
./ibm-lh bucket
```

Command	Description
`ibm-lh bucket add`	Add a storage bucket.
`ibm-lh bucket ls`	List all storage buckets.
`ibm-lh bucket info`	Get information about a specified bucket.

Command	Description
`ibm-lh bucket rm`	Delete a storage bucket.

`ibm-lh data-copy`

Interactive Mode

You can use interactive command line interface to run ingestion commands from inside the `ibm-lh-tool` container.

Note: To ingest any type of data files from a local file system, data files are needed to be copied to `~ / ibm-lh-client/localstorage/volumes/ibm-lh` directory. Now, you can access data files from `/ ibmlhdata/` directory by using the `ibm-lh data-copy` command.

Run the ingestion job using the following command:

Syntax

```
./ibm-lh data-copy --interactive
```

Note:

There are sample scripts and config file templates available for CLI ingestion commands in the `/ Ingestion-Examples/` folder inside the `ibm-lh-tool` container.

1. Run `cd /Ingestion-Examples/` command to access the example scripts and config files inside.

Non-interactive Mode

You can use non-interactive command line interface to run ingestion commands from outside the `ibm-lh-tool` container. You can execute the ingestion commands directly from the local environment where `ibm-client-package` is installed.

Note: To ingest any type of data files from a local file system, data files are needed to be copied to `~ / ibm-lh-client/localstorage/volumes/ibm-lh` directory. Now, you can access data files from `/ ibmlhdata/` directory by using the `ibm-lh data-copy` command.

Run the ingestion job using the following command:

Syntax

```
./ibm-lh data-copy <ingestion_parameters>
```

Where `<ingestion_parameters>` for `ibm-lh data-copy` command is listed in `ibm-lh data-copy` command options.

`ibm-lh table-maint`

Maintain tables with stored procedures.

Syntax

```
./ibm-lh table-maint --file <file-name>
```

You can use stored procedures to run the following table operations on Iceberg tables in Spark.

Snapshot management

Procedure	Description
`rollback_to_snapshot`	Roll back a table to a specific snapshot ID.
`rollback_to_timestamp`	Roll back to a specific time.
`set_current_snapshot`	Set the current snapshot ID for a table.

Procedure	Description
`cherrypick_snapshot`	Cherry-pick changes from a snapshot into the current table state.

Metadata management

Procedure	Description
`expire_snapshots`	Remove older snapshots and their files, which are no longer needed.
`remove_orphan_files`	Remove files that are not referenced in any metadata files of an Iceberg table.
`rewrite_data_files`	Combine small files into larger files to reduce metadata overhead and runtime file open cost.
`rewrite_manifests`	Rewrite manifests for a table to optimize scan planning.

Table migration

Procedure	Description
`register_table`	Create a catalog entry for a `metadata.json` file that exists but does not have a corresponding catalog identifier.

To generally use the utility:

```
$ ibm-lh [argument] [options]
```

To see a list of available commands:

```
$ ibm-lh --help
```

To see the description and options for a specific command:

```
$ ibm-lh [argument] --help
```

Chapter 5. Configuring user interface (UI) components

You can configure different components by using the UI of IBM watsonx.data.

Provisioning a Presto (Java) engine

An engine in watsonx.data runs SQL queries on your data source and fetches the queried data. Presto (Java) is one of the engines supported in watsonx.data.

Note: The versions of Presto (Java) supported depends on the watsonx.data versions. The following is the list of supported versions:

watsonx.data v1.0.0, v1.0.1 and v1.0.2: Presto (Java) version 0.279.

watsonx.data v1.0.3, v1.1.1 and v1.1.3: Presto (Java) version 0.282.

watsonx.data v1.1.4: Presto (Java) version 0.285.1.

watsonx.data v2.0.0: Presto (Java) version 0.286.

watsonx.data on Red Hat OpenShift

About this task

To provision a Presto (Java) engine, complete the following steps.

Procedure

1. Log in to watsonx.data console.
2. From the navigation menu, select **Infrastructure Manager**.
3. Click **Add component**, select **IBM Presto**, and click **Next**.
4. In the **Add component - IBM Presto** window, provide the following details to sign up new compute to work with your data:

Field	Description
Type	Select the **Presto (Java) <version>** engine from the list.
Display name	Enter your compute engine name.
Size	Select the engine size. **Custom** is selected by default. Custom size indicates that the Presto (Java) engine has a single-node baseline deployment. For more information about customization, see Specifying additional customization.
Associated catalogs (optional)	Associate the available catalogs with the engine if necessary.

5. Click **Create**.

Provisioning a Presto (C++) engine

An engine in watsonx.data runs SQL queries on your data source and fetches the queried data. Presto (C++) is one of the engines supported in watsonx.data.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

You can provision a Presto (C++) engine in IBM watsonx.data to run SQL queries on your data source and fetch the queried data.

Procedure

1. Log in to the watsonx.data console.
2. From the navigation menu, select **Infrastructure Manager**.
3. Click **Add component**, select **IBM Presto**, and click **Next**.
4. In the **Add component - IBM Presto** window, provide the following details to sign up new compute to work with your data:

Field	**Description**
Type	Select the engine type **Presto (C++) <version>** from the list.
Display name	Enter your compute engine name.
Size	Select the engine size. **Custom** is selected by default. **Custom** size includes 1 coordinator node and 3 worker nodes.
Associated catalogs (optional)	Associate the available catalogs with the engine if necessary.

5. Click **Create**.

Adding a Spark engine

Using IBM watsonx.data, you can add Spark engines.

You can either provision native Spark engine or register external Spark engine. Native Spark engine is a compute engine that resides within watsonx.data. External Spark engines are engines that exist in a different environment from where watsonx.data is available.

watsonx.data on Red Hat OpenShift

About this task

To add a Spark engine, complete the following steps.

Procedure

1. Log in to watsonx.data console.
2. From the navigation menu, select **Infrastructure Manager**.
3. To add a Spark engine, click **Add component** and click **Next**.
4. In the **Add component** page, from the **Engines** section, select **IBM Spark**.
5. In the **Add component - IBM Spark** page, configure the following details:

   Important: Co-located and self-managed Spark engines are deprecated in the 2.0.2 release and will not be available from the 2.0.3 release onwards. Use native Spark engine for Spark use cases. To start using native Spark engine, see "Native Spark engine" on page 490.

Field	**Description**
Display name	Enter your compute engine name.

Field	Description
Registration mode	Based on your requirement, you can select one of the following options: • **Create a native Spark engine** : The native Spark engine is a compute engine that resides within watsonx.data. If you select this option, see "Provisioning native Spark engine" on page 490 to provision the native Spark engine. • **Register an external Spark engine** : The Spark and watsonx.data instances are located in different clusters. For example, your Spark instance is provisioned on IBM Cloud, and watsonx.data is installed on your computer. • **Register a co-located Spark engine** (deprecated) : The Spark and watsonx.data instances are located in the same cluster.
Instance	If you selected the **Register a co-located Spark engine** (deprecated) as the **Registration mode** , select the **Spark instance** (that is colocated with watsonx.data) from the list. Click **Create one** to create an instance if you do not have one.
Management method	If you selected **Register an external Spark engine** as the **Registration mode** , select the appropriate management method: • **Fully-managed**: Indicates that the Spark instance is owned and managed by IBM Cloud. • **Self-managed**(deprecated): Indicates that the instance is an IBM Analytics Engine Spark on Cloud Pak for Data cluster.
Instance API endpoint	If you selected the **Registration mode** as **Register an external Spark engine** and **Management method** as **Fully-managed**, enter the IBM Analytics engine instance endpoint. For more information, see Retrieving service endpoints.
API key	If you selected the **Registration mode** as **Register an external Spark engine** and **Management method** as **Fully-managed**, enter the API key.
Spark jobs V4 endpoint	If you selected the **Registration mode** as **Register an external Spark engine** and **Management method** as **Self-managed** (deprecated), enter the self-managed IBM Analytics engine endpoint details.
ZenApiKey	If you selected the **Registration mode** as **Register an external Spark engine** and **Management method** as **Self-managed** (deprecated), enter the self-managed API details.

6. Click **Create**. The engine is provisioned and is displayed in the **Infrastructure Manager** page.

Registering an external engine

Some engines cannot be directly provisioned in IBM watsonx.data. Such engines can be registered and used for querying data.

You can register the following engines:

- IBM Spark

  Note: For more information about registering Spark as an external engine, see Adding a Spark engine.
- IBM Db2 Warehouse
- IBM Netezza
- Other

watsonx.data on Red Hat OpenShift

About this task

To register external engines, complete the following steps.

Procedure

1. Log in to watsonx.data console.
2. From the navigation menu, select **Infrastructure Manager**.
3. Click **Add component**. Select the required from the **Engines** section and click **Next**.
4. In the **Add component** window, enter the details based on the engine that you selected.
   - IBM Db2 Warehouse
   - IBM Netezza
   - Other

   IBM Db2 Warehouse

   For **IBM Db2 Warehouse**, configure the following details:

Field	Description
Display name	Enter your compute engine name.
Instance URL	Enter the console URL for **IBM Db2 Warehouse**.
Complete watsonx.data configuration	External engines require extra watsonx.data configuration. For more information, see How to configure watsonx.data in IBM Db2 Warehouse.
Confirmation checkbox	Select the confirmation checkbox to confirm complete configuration.

   IBM Netezza

   For **IBM Netezza**, configure the following details:

Field	Description
Display name	Enter your compute engine name.
Instance URL	Enter the console URL for **IBM Netezza**.
Complete watsonx.data configuration	External engines require extra watsonx.data configuration. For more information, see How to configure watsonx.data in IBM Netezza.
Confirmation checkbox	Select the confirmation checkbox to confirm complete configuration.

   Other

   For **Other**, configure the following details:

Field	Description
Type	Enter your engine type.
Display name	Enter your engine name.
Instance URL	Enter your instance URL.
Complete watsonx.data configuration	External engines require extra watsonx.data configuration to query watsonx.data catalogs. For more information, see your engine's documentation.
Confirmation checkbox	Select the confirmation checkbox to confirm complete configuration.

5. Click **Create**.

Editing engine details

To edit the display name, description and tags of an engine, use one of the following methods:

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Edit the engine display name, description, and tags in list view.
   a) Click the name of engine. The engine information window opens.
   b) In **Details** tab, click **Edit**.
   c) In the **Display name** field, enter the display name for the engine.
   d) In the **Description** field, enter the description of the engine or edit the existing description.
   e) In the **Tags** field, select the tags from list or type a tag name to define a new tag.
   f) Click **Save**.
2. Edit the engine display name, description, and tags in topology view.
   a) Click the engine display name. The engine information window opens.
   b) In **Details** tab, click **Edit**.
   c) In the **Display name** field, enter the display name for the engine.
   d) In the **Description** field, enter the description of the engine or edit the existing description.
   e) In the **Tags** field, select the tags from list or type a tag name to define a new tag.
   f) Click **Save**.

Associating a catalog with an engine

To associate a catalog or catalogs with an engine, use one of the following methods:

watsonx.data on Red Hat OpenShift

Procedure

1. Associate a catalog or catalogs with an engine in list view.
   a) Click the overflow menu and then click **Manage associations**.
   b) In **Manage associations** window, select the engine with which you want to associate the catalog or catalogs.
   c) Click **Save and restart engine**.
2. Associate a catalog or catalogs with an engine in topology view.

a) Hover over the catalog that you want to associate with an engine and click the **Manage associations** icon.

b) In **Manage associations** window, select the engine with which you want to associate the catalog or catalogs.

c) Click **Save and restart engine**.

Exploring the catalog objects

To explore the objects in a catalog, use one of the following methods:

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Explore the catalog objects in list view.
   a) Click the name of catalog that you want to explore. Catalog information window opens.
   b) Click **Data objects**.
2. Explore the catalog objects in topology view.
   a) Click the catalog that you want to explore. Catalog information window opens.
   b) Click **Data objects**.

Dissociating a catalog from an engine

To dissociate a catalog or catalogs with an engine, use one of the following methods:

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Dissociate a catalog or catalogs from an engine in list view.
   a) Click the overflow menu icon and then click **Manage associations**.
   b) In the **Manage associations** window, clear the checkbox in the **Engine** column.
   c) Click **Save and restart engine**.
2. Dissociate a catalog or catalogs from an engine in topology view.
   a) Hover over the catalog that you want to dissociate from an engine and click the **Manage associations** icon.
   b) In the **Manage associations** window, clear the checkbox in the **Engine** column.
   c) Click **Save and restart engine**.

Deleting an engine

To delete an engine, use one of the following methods:

watsonx.data on Red Hat OpenShift

Procedure

1. Delete an engine in list view.
   a) Click the overflow menu icon at the end of the row and click **Delete**. A delete confirmation dialog appears.
   b) Click **Delete**.

2. Deleting a database in topology view.
    a) Hover over the engine that you want to delete and click the **Delete** icon. A delete confirmation dialog appears.
    b) Click **Delete**.

Configuring Presto resource groups

You can configure one or more Presto resource groups in watsonx.data.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Procedure

1. Log in to watsonx.data console.
2. From the navigation menu, go to **Configurations**.
3. Click the **Presto resource groups** tile.

   The **Presto resource groups** window opens.
4. Click **Add Resource Group**.
5. From the **Add Resource Group** window, browse and select a resource group JSON file.

   Alternatively, you can drag and drop a resource group file.

   Important: The uploaded JSON file structure must match with the sample resource group file structure. To download the sample file, click **Download sample resource group**. The maximum allowed size of a file is 2 MB and the only file format supported is `.json`.

   For more information about the resource group properties that you can define in the JSON file, see Resource group properties.
6. Click **Add** to add the resource group.

   Note: To delete a resource group, go to the overflow menu of the resource group and click **Delete**. A confirmation box opens. Click **Delete** to confirm deletion. You cannot delete a resource group if it is assigned to an engine.
7. To assign the resource group to engines, click the overflow menu of the resource group and select **Assign**.
8. Select one or more Presto engines from the **Available engines** section and click **Assign**.

   A confirmation box opens.

   Note: If you click **Confirm**, the engines restart, and any ongoing queries of the data sources that are associated to the engines are terminated. You can have only one resource group assigned to an engine.
9. Click **Confirm** to complete the configuration.

Unassigning resource group from engines

You can unassign a resource group from engines.

Procedure

1. Click the overflow menu of the resource group and select **Unassign**.
2. From the **Currently assigned engines** section, select one or more engines that you want to unassign the resource group from.
3. Click **Unassign**.

   A confirmation box opens.

   Note: If you click **Confirm**, the engines restart, and any ongoing queries of the data sources that are associated to the engines are terminated.

4. Click **Confirm** to unassign.

Managing the Spark engine details

IBM watsonx.data allows you to view and edit the details of a Spark engine. You can also monitor the status of the applications that are submitted in the instance.

watsonx.data on Red Hat OpenShift

About this task

Viewing Spark details

You can view the Spark details in list and topology views.

1. Click the name of Spark engine (either from list or topology view). Engine information window opens.
2. In the **Details** tab, you can view the following details:

Field	Description
Display name	The Spark engine name.
Engine ID	The unique identifier of the Spark instance.
Description	The description of the engine.
Tags	The tag that is specified at the time of registering an engine.
Type	The engine type. Here, IBM Analytics Engine (Spark).
Instance URL	The IBM Analytics Engine (Spark) URL.
watsonx.data application endpoint	The application submission endpoint. To submit an application by using API, see API Docs.
Instance API endpoint	The IBM Analytics Engine (Spark) API endpoint.
History server endpoint	The IBM Analytics Engine (Spark) history server endpoint.
History server	The history server URL, where you can view the history details of the applications that are run.

Editing Spark details

You can edit the Spark details in list and topology views.

1. Click the name of Spark engine (either from list or topology view). Engine information window opens.
2. In the **Details** tab, click **Edit**.
3. In the **Display name** field, enter the display name for the Spark engine.
4. In the **Description** field, enter the description of the engine or edit the existing description.
5. In the **Tags** field, select the tags from the list or start typing to define a new tag.
6. Click **Save**.Click the name of Spark engine (either from list or topology view). Engine information window opens.

Viewing the submitted Spark applications in watsonx.data

You can view the status of the submitted Spark applications in list and topology views.

1. Click the name of the Spark engine (either from list or topology view). Engine information window opens.

2. In **Applications** tab, you can view the list of all applications that are submitted to watsonx.data. The tab also displays the details such as the status of each application, the Spark version that is used for running the application, creation date, and run date.

Adding a storage-catalog pair

You can organize your data in watsonx.data by associating storage.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

About this task

In watsonx.data, the data is stored either in an internal storage created during instance provisioning or in an externally managed storage. You can associate a catalog with a storage. A catalog defines the schemas and metadata for a storage.

Note: The out-of-the-box MinIO object storage is provided for exploratory purposes only. It does not have all the security features and is not configured to provide high-speed data access. Register your own s3 bucket that meets your security and performance requirements.

Procedure

To add a storage, complete the following steps:

1. Log in to watsonx.data console.
2. From the navigation menu, select **Infrastructure manager**.
3. To define and connect a storage, click **Add component**.
4. In the **Add component** window, select a storage from the **Storage** section and provide the details to connect to existing externally managed storage.

   Note: A catalog defines the schemas and metadata for a data source. Depending on storage type, Iceberg, Hive, Hudi, and Delta lake catalogs are supported.

   Note: You can modify the access key and secret key of a user-registered bucket for a storage. This feature is only available for user-registered buckets and is not applicable to default buckets, ADLS, or Google Cloud Storage. This feature can only be used if the new credentials successfully pass the test connection.

   Features

   For **Iceberg** connector:

   a. You can delete data from tables by using `DELETE FROM` statement for **Iceberg** connector.
   b. You can specify the table property `delete_mode` for new tables by using either `copy-on-write` mode or `merge-on-read` mode (default).

   For `DELETE FROM` statement for **Iceberg** connector:

   a. Filtered columns only support comparison operators, such as `EQUALS`, `LESS THAN`, or `LESS THAN EQUALS`.
   b. Deletes must only occur on the latest snapshot.
   c. For V1 tables, the **Iceberg** connector can delete data only in one or more entire partitions. Columns in the filter must all be identity-transformed partition columns of the target table.

   For the **Iceberg** connector, `ALTER TABLE` operations on a column support the following data type conversions:

   a. `INT` to `BIGINT`
   b. `FLOAT` to `DOUBLE`
   c. `DECIMAL`(num1, dec_digits) to `DECIMAL`(num2, dec_digits), where num2>num1

Limitations for SQL statements

a. For **Iceberg**, `UPDATE` query with sub-query is not supported.

b. For **Iceberg**, `UPDATE` query with mixed-case column is not supported.

c. For **Iceberg**, **Memory** and **Hive** connectors, `DROP SCHEMA` can do `RESTRICT` by default.

d. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

a. For the **Iceberg** connector, the maximum number of digits that can be accommodated in a column of data type `FLOAT` and `DOUBLE` is 37. Trying to insert anything larger ends up in a decimal overflow error.

b. When the fields of data type `REAL` have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

For more information on mixed-case feature flag behavior, supported SQL statements, and supported data types matrices, see Support content.

IBM Cloud Object Storage

IBM Cloud® Object Storage stores encrypted and dispersed data across multiple geographic locations.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

You can create an IBM Cloud Object Storage by creating an instance. For more information, see Create an instance. If you select **IBM Cloud Object Storage** from the **Storage** section, configure the following details:

Field	Description
Display name	Enter the name to be displayed. (The following special characters are not allowed: ! @ # $ % ^ & * ( ) = + : { } < > ? ' \ ; `).
Bucket name	Enter the name of your existing bucket. If you do not have an existing bucket then you can create a new bucket. For more information, see Create bucket.
Region	Select the region where the storage is available.
Endpoint	Enter the **Endpoint** URL. Test connection is enabled when the endpoint is provided. **Fetch the Endpoint by the following steps:** • Go to **IBM Cloud Console**. • Select the service as **IBM Cloud Object** storage**>** Select your bucket. • Go to the **Configuration** tab**>** Click **Endpoints**. • Copy the **Public** endpoint address. **Create the Endpoint by the following steps:** • Endpoints are created when bucket is configured. For more information, see Configure Endpoint.

Field	Description
Access key	Enter your **Access key**. **Fetch the credentials for access key by the following steps:** • From the **Resource list** page, select the name of the service to open the service details page. • Click **Service credentials**. • Expand the row for the required access key credential. For more information, see Viewing a credential. **Create the credentials for access key by the following steps if you do not have it:** • Create credentials for bucket. For more information, see Service credentials. • Enable the HMAC toggle switch to get the Access Key and Secret Key paired for use with S3-compatible tools. For more information, see Using HMAC credentials.
Secret key	Enter your **Secret key**. **Fetch the credentials for secret key by the following steps:** • From the **Resource list** page, select the name of the service to open the service details page. • Click **Service credentials**. • Expand the row for the required secret key credential. For more information, see Viewing a credential. **Create the credentials for secret key by the following steps if you do not have it:** • Create credentials for bucket. For more information, see Service credentials. • Enable the HMAC toggle switch to get the Access Key and Secret Key paired for use with S3-compatible tools. For more information, see Using HMAC credentials.
Connection Status	Click the **Test connection** link to test the bucket connection. If the bucket connection is successful, a success message appears.
Associate catalog	Select the checkbox to add a catalog for your storage. This catalog is associated with your storage and serves as your query interface with the data stored within.
Activate now	Select the checkbox to terminate in-flight queries against data in any bucket.
Catalog type	Select the catalog type from the list. The recommended catalog is Apache Iceberg. The other options for catalog are Apache Hive, Apache Hudi, and Delta Lake.
Catalog name	Enter the name of the associated catalog. (The following special characters are not allowed: ! @ # $ % ^ & * ( ) = + : { } < > ? ' \ ; `).
Create	Click **Create** to associate the storage.

Amazon S3

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

If you select **Amazon S3** from the **Storage** section, configure the following details:

Field	Description
Display name	Enter the name to be displayed.
Bucket name	Enter the name of your existing bucket.
Region	Select the region where the storage is available.
Endpoint	Enter the **Endpoint** URL.
Access key	Enter your **Access key**.
Secret key	Enter your **Secret key**.
Connection Status	Click the **Test connection** link to test the bucket connection. If the bucket connection is successful, a success message appears.
Associate catalog	Select the checkbox to add a catalog for your storage. This catalog is associated with your storage and serves as your query interface with the data stored within.
Catalog type	Select the catalog type from the list. The recommended catalog is Apache Iceberg. The other options for catalog are Apache Hive, Apache Hudi, and Delta Lake.
Catalog name	Enter the name of the associated catalog.
Create	Click **Create** to create the storage.

IBM Storage Ceph

IBM Storage Ceph is a scalable, open, software-defined storage platform. It combines an enterprise-hardened version of the Ceph storage system, with a Ceph management platform, deployment utilities, and support services.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

If you select **IBM Storage Ceph** from the **Storage** section, configure the following details:

Field	Description
Display name	Enter the name to be displayed.
Bucket name	Enter the name of your existing bucket.
Endpoint	Enter the **Endpoint** URL. Test connection is enabled when the endpoint is provided.
Access key	Enter your **Access key**.
Secret key	Enter your **Secret key**.
Connection Status	Click the **Test connection** link to test the bucket connection. If the bucket connection is successful, a success message appears.

Field	Description
Associate catalog	Select the checkbox to add a catalog for your storage. This catalog is associated with your storage and serves as your query interface with the data stored within.
Catalog type	Select the catalog type from the list. The recommended catalog is Apache Iceberg. The other options for catalog are Apache Hive, Apache Hudi, and Delta Lake.
Catalog name	Enter the name of the associated catalog.
Create	Click **Create** to create the storage.

MinIO

MinIO is a high-performance, S3 compatible object store. It is built for large scale data lake and database workloads.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

If you select **MinIO** from the **Storage** section, configure the following details:

Field	Description
Display name	Enter the name to be displayed.
Bucket name	Enter the name of your existing bucket.
Endpoint	Enter the **Endpoint** URL. Test connection is enabled when the endpoint is provided.
Access key	Enter your **Access key**.
Secret key	Enter your **Secret key**.
Connection Status	Click the **Test connection** link to test the bucket connection. If the bucket connection is successful, a success message appears.
Associate catalog	Select the checkbox to add a catalog for your storage. This catalog is associated with your storage and serves as your query interface with the data stored within.
Catalog type	Select the catalog type from the list. The recommended catalog is Apache Iceberg. The other options for catalog are Apache Hive, Apache Hudi, and Delta Lake.
Catalog name	Enter the name of the associated catalog.
Create	Click **Create** to create the storage.

Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) is a file system that manages large data sets that can run on commodity hardware.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

If you select **Hadoop Distributed File System (HDFS)** from the **Storage** section, configure the following details:

Field	Description
Display name	Enter the name to be displayed.
Thrift URI	Enter the **Thrift URI**.
Thrift port	Enter **Thrift port**.
Kerberos authentication	Use the toggle switch to enable or disable **Kerberos authentication**. If enabled, enter the following information: 1. HDFS principal 2. Hive client principal 3. Hive server principal Upload the following files: 1. Kerberos config file (.config) 2. HDFS keytab file (.keytab) 3. Hive keytab file (.keytab)
Upload core site file (.xml)	Upload core site file (.xml).
Upload HDFS site file (.xml)	Upload HDFS site file (.xml).
Associated catalog	Add a catalog for your storage. This catalog is associated with your storage and serves as your query interface with the data stored within.
Catalog type	Select the catalog type from the list. The recommended catalog is Apache Iceberg. The other options for catalog are Apache Hive, Apache Hudi, and Delta Lake.
Catalog name	Enter the name of the associated catalog.
Create	Click **Create** to create the storage.

Google Cloud Storage

Google Cloud Storage is a service for storing objects in Google Cloud. An object is an immutable piece of data consisting of a file of any format.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

If you select **Google Cloud Storage** from the **Storage** section, configure the following details:

Field	Description
Display name	Enter the name to be displayed.
Bucket name	Enter the name of your existing bucket.
Upload JSON key file (.json)	Upload the JSON key file. JSON key file is used to authenticate a Google Cloud service account with Google Cloud Storage.
Associate catalog	Select the checkbox to add a catalog for your storage. This catalog is associated with your storage and serves as your query interface with the data stored within.
Catalog type	Select the catalog type from the list. The recommended catalog is Apache Iceberg. The other options for catalog are Apache Hive, Apache Hudi, and Delta Lake.
Catalog name	Enter the name of the associated catalog.

Field	Description
Create	Click **Create** to create the storage.

Note: For Google Cloud Storage, multiple buckets of different service accounts cannot be configured.

Azure Data Lake Storage Gen1 Blob

Azure Data Lake Storage (ADLS) is a scalable data storage and analytics service that is hosted in Azure, Microsoft's public cloud. The Microsoft Azure Data Lake Storage connection supports access to both Gen1 and Gen2 Azure Data Lake Storage repositories.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

If you select **Azure Data Lake Storage Gen1 Blob** from the **Storage** section, configure the following details:

Field	Description
Display name	Enter the name to be displayed.
Container name	Enter the **Container name**.
Storage account name	Enter the **Storage account name**.
Endpoint	Upload the JSON key file. JSON key file is used to authenticate a Google Cloud service account with Google Cloud Storage.
Authentication mode	Select the **Authentication mode**. • **SAS**: Enter your SAS token. • **Account key**: Enter your access key.
Associate catalog	Select the checkbox to add a catalog for your storage. This catalog is associated with your storage and serves as your query interface with the data stored within.
Catalog type	Select the catalog type from the list. The recommended catalog is Apache Iceberg. The other options for catalog are Apache Hive, Apache Hudi, and Delta Lake.
Catalog name	Enter the name of the associated catalog.
Create	Click **Create** to create the storage.

Note: Azure Data Lake Storage Gen2 and Azure Data Lake Storage Gen1 Blob storage do not support SAS authentication mode for Presto (Java) engine.

Limitations

- Use separate containers and storage accounts for ADLS Gen1 and ADLS Gen2 storage for complete metadata synchronization, including tables. Otherwise, a PARTIAL SUCCESS message appears in the sync logs when SYNC finishes.

Azure Data Lake Storage Gen2

Azure Data Lake Storage (ADLS) is a scalable data storage and analytics service that is hosted in Azure, Microsoft's public cloud. The Microsoft Azure Data Lake Storage connection supports access to both Gen1 and Gen2 Azure Data Lake Storage repositories.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

If you select **Azure Data Lake Storage Gen2** from the **Storage** section, configure the following details:

Field	Description
Display name	Enter the name to be displayed.
Container name	Enter the **Container name**.
Storage account name	Enter the **Storage account name**.
Endpoint	Enter the **Endpoint** URL.
Authentication mode	Select the **Authentication mode**. • **SAS**: Enter your SAS token. • **Service Principle**: Enter the Application id, Directory id and Secret key.
Associate catalog	Select the checkbox to add a catalog for your storage. This catalog is associated with your storage and serves as your query interface with the data stored within.
Catalog type	Select the catalog type from the list. The recommended catalog is Apache Iceberg. The other options for catalog are Apache Hive, Apache Hudi, and Delta Lake.
Catalog name	Enter the name of the associated catalog.
Create	Click **Create** to create the storage.

Note: Azure Data Lake Storage Gen2 and Azure Data Lake Storage Gen1 Blob storage do not support SAS authentication mode for Presto (Java) engine.

Limitations

- Use separate containers and storage accounts for ADLS Gen1 and ADLS Gen2 storage for complete metadata synchronization, including tables. Otherwise, a PARTIAL SUCCESS message appears in the sync logs when SYNC finishes.

IBM Storage Scale

IBM Storage Scale is software-defined file and object storage that is built over a highly performant and scalable clustered file system. It is widely used for high-performance computing, data-intensive technical computing, big data, and analytics. It is complemented with many new machine learning and AI workloads.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

If you select **IBM Storage Scale** from the **Storage** section, configure the following details:

Field	Description
Display name	Enter the name to be displayed.
Bucket name	Enter the name of your existing bucket.
Endpoint	Enter the **Endpoint** URL. Test connection is enabled when the endpoint is provided.
Access key	Enter your **Access key**.
Secret key	Enter your **Secret key**.
Connection Status	Click the **Test connection** link to test the bucket connection. If the bucket connection is successful, a success message appears.

Field	Description
Associate catalog	Select the checkbox to add a catalog for your storage. This catalog is associated with your storage and serves as your query interface with the data stored within.
Catalog type	Select the catalog type from the list. The recommended catalog is Apache Iceberg. The other options for catalog are Apache Hive, Apache Hudi, and Delta Lake.
Catalog name	Enter the name of the associated catalog.
Create	Click **Create** to create the catalog.

Exploring the storage objects

To explore the objects in a storage, use one of the following methods:

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Explore the storage objects in list view.
   a) Click the name of storage that you want to explore. Storage information window opens.
   b) Click **Objects**.
2. Explore the storage objects in topology view.
   a) Click the storage that you want to explore. Storage information window opens.
   b) Click **Objects**.

Editing storage details

To edit the display name, description and tags of a storage, use one of the following methods:

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Edit the storage display name, display name, description and tags in list view.
   a) Click the name of storage. Storage information window opens.
   b) In the **Details** tab, click **Edit**.
   c) In the **Display name** field, enter the name to be displayed.
   d) In the **Description** field, enter the description of the engine or edit the existing description.
   e) In the **Tags** field, select the tags from list or start typing to define a new tag.
   f) Click **Save**.
2. Edit the storage display name, description and tags in topology view.
   a) Click the storage. Storage information window opens.
   b) In the **Details** tab, click **Edit**.
   c) In the **Display name** field, enter the name to be displayed.
   d) In the **Description** field, enter the description of the engine or edit the existing description.
   e) In the **Tags** field, select the tags from list or start typing to define a new tag.
   f) Click **Save**.

Deleting a storage-catalog pair

To delete a storage-catalog pair, complete the following steps:

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Dissociate the catalog that is associated with the storage from the engine. For instructions, see "Dissociating a catalog from an engine" on page 303.
2. In **Infrastructure manager**, go to the **Storage** tab.
3. Click the overflow menu and then click **Deactivate**.
4. In the **Confirm deactivation** window, click **Deactivate**.
5. Click the overflow menu and then click **Remove**.
6. In the **Confirm removal** window, click **Remove**.

Adding a data source-catalog pair

You can connect to various data sources through IBM watsonx.data for Presto.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

About this task

You can register and use data sources in IBM watsonx.data. A catalog defines the schemas and metadata for a data source.

Procedure

1. Log in to watsonx.data console.
2. From the navigation menu, select **Infrastructure manager**.
3. To define and connect a database, click **Add component**.
4. In the **Add component** window, select a database from the **Data source** section.

   Note: Two databases with the same name cannot be added.

Apache Druid

Apache Druid is a real-time analytics database designed for fast slice-and-dice analytics ("OLAP" queries) on large data sets.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Apache Druid

Configure the following details for Apache Druid:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Coordinator host	Enter the **Coordinator host**.
Coordinator port	Enter the **Coordinator port**.

Field	Description
Broker host	Enter the **Broker host**.
Broker port	Enter the **Broker port**.
Schema name	Enter the **Schema name**.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Apache Kafka

Apache Kafka is a distributed event streaming platform. Connect to an Apache Kafka real-time processing server to write and to read Streams of events from and into topics.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Apache Kafka

Configure the following details for Apache Kafka:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Hostname	Enter the **Hostname**. You can add multiple host information. To add, click the **Add** icon. A new row appears for adding hostname and port number. Enter the details.
Port	Enter the **Port** number.

Field	Description
SASL connection	Use the toggle switch to enable or disable the Simple Authentication Security Layer (SASL) to include an authentication mechanism. If enabled, specify the **Username** and **API key/ Password**.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Add topics	You can add topics after you create the database. i. Go to the **Infrastructure manager**. ii. Select the **Apache Kafka** database. iii. Click **Add topics** option. iv. Upload `.json` definition files. You can either drag the files or use the **Click to upload** option. Topic names are determined from the definition files. v. Use the **Edit** option to view and edit the topic files.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting

Apache Pinot

Apache Pinot is an open source-distributed database that is designed for real-time, user-facing analytics.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Apache Pinot

Configure the following details for Apache Pinot:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Hostname	Enter the hostname.
Port	Enter the port number.

Field	Description
Controller authentication	Use the toggle switch to enable or disable **Controller authentication**. If enabled, enter the controller username and password.
Broker authentication	Use the toggle switch to enable or disable **Broker authentication**. If enabled, enter the broker username and password.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. When Presto (Java) engine tries to contact Apache Pinot server directly, the queries do not work in the following scenarios:

   i. Nonlimit and Nonaggregate queries do not work with an SSL connection.

   ii. Limit queries that depend on an inner query that does not work with an SSL connection. For example,

   ```
   SELECT playerstint, teamid
   FROM pinot.default.baseballstats
   WHERE playerstint IN (
     SELECT playerstint
     FROM pinot.default.baseballstats
     LIMIT 2
   )
   LIMIT 5;
   ```

2. Queries to **Apache Pinot** fail if the broker's instance ID doesn't have a valid hostname or IP address.
3. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Amazon Redshift

Amazon Redshift uses SQL to analyze structured and semi-structured data across data warehouses, operational databases, and data lakes. It uses AWS-designed hardware and machine learning to deliver the best price performance at any scale.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Amazon Redshift

Configure the following details for Amazon Redshift:

Field	**Description**
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
SSL connection	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt , .cert or .cer)** link. iii. Browse the SSL certificate and upload.
Validate certificate	Use the toggle switch to validate whether the SSL certificate that the host returns is trusted or not.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

BigQuery

BigQuery database allows querying the data stored in BigQuery. This can be used to join data between different systems like BigQuery and Hive.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

BigQuery

Configure the following details for BigQuery:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Project id	Enter the **Project id**.
Authentication type	Select the type of authentication: • JSON key (Base 64 encoded) : Enter the Base 64 encoded Google service account key. • JSON key (JSON) : Enter the Google service account JSON key.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Cassandra

Cassandra is an open-source NoSQL distributed database that handles a huge amount of data across commodity servers.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Cassandra

Configure the following details for Cassandra:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.

Field	Description
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. Standard `INSERT INTO` statement is not supported.
2. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

ClickHouse

ClickHouse is a high-performance, column-oriented SQL database management system (DBMS) for online analytical processing (OLAP).

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

ClickHouse

Configure the following details for ClickHouse:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.

Field	Description
Validate certificate	Use the toggle switch to validate that the SSL certificate returned by the host is trusted or not.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Elasticsearch

Elasticsearch is a NoSQL data source that stores data in a semi-structured manner using a JSON-based document format.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Elasticsearch

Configure the following details for Elasticsearch:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
SSL connection	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert or .cer)** link. iii. Browse the SSL certificate and upload.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.

Field	Description
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `BINARY` data type supports only `SELECT` statement.
2. When the fields of data type `REAL` have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

IBM Data Virtualization Manager

IBM Data Virtualization Manager for z/OS provides virtual, integrated views of data residing on IBM Z. It enables users and applications to have read or write access to IBM Z data in place, without having to move, replicate, or transform the data.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

IBM Data Virtualization Manager

Configure the following details for IBM Data Virtualization Manager:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert or .cer)** link. iii. Browse the SSL certificate and upload.
Validate certificate	Use the toggle switch to validate whether the SSL certificate that the host returns is trusted or not.

Field	Description
Hostname in SSL certificate	Provide the hostname in SSL certificate.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

IBM Db2

IBM Db2 is a database that contains relational data.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

IBM Db2

Configure the following details for IBM Db2:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Authentication type	Choose and enter the **Authentication type** details: • Username and password: Enter the database username and database password. • Authentication value: Enter the API key.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.

Field	Description
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Note: Select IBM Db2 from the **Database Type** drop-down list to add IBM Watson Query.

You can now query the nicknames that are created in Db2 and the virtualized tables from Watson Query instances.

Features

You can perform the following operations for `BLOB` and `CLOB` data types for **Db2** data source:

- INSERT
- CREATE
- CTAS
- ALTER
- DROP

To insert `CLOB` data type, provide the value directly or cast it explicitly to `CLOB` data type.

```
INSERT INTO <table_name> VALUES ('<clob value>', '<other values>');
```

```
INSERT INTO <table_name> VALUES (CAST('<clob text>' AS CLOB));
```

To insert `BLOB` data, use the cast function with `BLOB` data type. The corresponding hexadecimal value is inserted into the **Db2** data source:

```
INSERT INTO <table_name> VALUES (CAST('<blob text>' AS BLOB));
```

Limitations for SQL statements

1. `ALTER TABLE DROP COLUMN` operation is not supported for column-organized tables.
2. `DROP TABLE` statement is supported only when enabled in the catalog.
3. `CREATE VIEW` can be used for a table only if that table is in the same catalog and the same schema.
4. `DROP SCHEMA` can do `RESTRICT` by default.
5. `CREATE VIEW` with `JOINS` is not supported.
6. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `BLOB` and `CLOB` data types support `SELECT` statement but do not support operations such as `equal`, `like`, and `in`.
2. `BINARY` data type supports only `SELECT` statement.
3. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

IBM Netezza

Netezza Performance Server is a platform for high-performance data warehousing and analytics.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

IBM Netezza

Configure the following details for IBM Netezza:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Note: For a database type as Netezza, select the version 11.2.2.x.

Limitations for SQL statements

1. **IBM Netezza** connector partially supports `ALTER TABLE` and `CREATE VIEW` statements.
2. `DROP TABLE` statement is supported only when enabled in the catalog.
3. You can use `CREATE VIEW` for a table only if that table is in the same catalog and the same schema.
4. `DROP SCHEMA` can do `RESTRICT` by default.
5. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `NUMERIC` data type is not supported. You can use `DECIMAL` data type as an equivalent alternative to `NUMERIC` data type.
2. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and

10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Note: IBM Netezza supports the `unicode` characters from the Presto (Java) engine with `varchar` and `char` data types.

IBM Informix

IBM Informix is an enterprise class Object Relational Database Management System (ORDBMS). It is extensible and provides extreme transaction performance and features to support both OLTP and Data Warehouse database services.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

IBM Informix

Configure the following details for IBM Informix:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Informix server	Enter the **Informix server** name.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert or .cer)** link. iii. Browse the SSL certificate and upload.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

MongoDB

MongoDB is a distributed database that stores data in JSON-like documents.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

MongoDB

Configure the following details for MongoDB:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Authentication database name	Enter the name of your authentication database.
Hostname	Enter the **Hostname.** You can add multiple host information. To add, click the **Add** icon. A new row appears for adding hostname and port. Enter the details.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert or .cer)** link. iii. Browse the SSL certificate and upload.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. **MongoDB** connector does not support complex deletes involving `OR` statements with different columns for `DELETE` statement.

2. `DROP TABLE` statement is supported only when enabled in the catalog.
3. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

MySQL

MySQL is an open source relational database management system.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

MySQL

Configure the following details for MySQL:

Field	**Description**
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Validate certificate	Use the toggle switch to enable or disable **Validate certificate** option. Enable it to validate whether the SSL certificate that is returned by the host is trusted or not.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.

Field	Description
Create	Click **Create** to create the database.

Limitations for SQL statements

1. `DROP TABLE` statement is supported only when enabled in catalog.
2. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `BLOB` and `CLOB` data types support `SELECT` statement but do not support operations such as `equal`, `like`, and `in`.
2. `BINARY` data type supports only `SELECT` statement.
3. The data that is shown for the `BLOB` data type from the UI is in Base64 format, while the result from presto-cli is in hexadecimal format.
4. The data that is shown for the `BINARY` data type from the UI is in Base64 format, while the result from presto-cli is in hexadecimal format.
5. You can use `CLOB` data type as an equivalent alternative to `LONGTEXT`.
6. When the fields of data type `REAL` have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Oracle

Oracle allows querying and creating tables in an external Oracle database.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Oracle

Configure the following details for Oracle:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Connection mode	Select the **Connection mode** as either Service name or SID.
Connection mode value	Enter the **Connection mode value**.

Field	Description
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `BINARY`, `BLOB`, and `CLOB` data types support only `SELECT` statement.
2. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

PostgreSQL

PostgreSQL is an open source and customizable object-relational database.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

PostgreSQL

Configure the following details for PostgreSQL:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.

Field	Description
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Validate certificate	The toggle switch must be enabled to **Validate certificate**. If disabled, the test connection feature does not work and the PostgreSQL database cannot be added to the engine.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Note: Enable the toggle switch for **Validate certificate** option to create the PostgreSQL database. If disabled, the **Test connection** feature does not work and the database cannot be added to the engine.

Limitations for SQL statements

1. `DROP TABLE` statement is supported only when enabled in catalog.
2. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `BLOB` and `CLOB` data types can be used as an equivalent alternative to BYTEA and TEXT respectively.
2. `BLOB` and `CLOB` data types support SELECT statement but do not support operations such as `equal`, `like`, and `in`.
3. The data that is shown for the `BLOB` data type from the UI is in Base64 format, while the result from presto-cli is in hexadecimal format.
4. `BINARY` data type supports only `SELECT` statement.
5. BYTEA is the `BINARY` alternative data type.
6. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Prometheus

Prometheus allows reading Prometheus metrics as tables in Trino by using the Prometheus HTTP API mechanism.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Prometheus

Configure the following details for Prometheus:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert or .cer)** link. iii. Browse the SSL certificate and upload.
Verify hostname	Use the toggle switch to enable hostname verification in the SSL certificate.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Add	Click **Add** to add the database.

Limitations

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Redis

Redis is a fastest in-memory database. It provides solutions for caching, vector search, and NoSQL databases.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Redis

Configure the following details for Redis:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Note: You can add tables after creating the database from the View database page.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

SAP HANA

SAP HANA is a column-oriented, in-memory relational database.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

SAP HANA

Configure the following details for SAP HANA:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
SSL connection	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Validate certificate	Use the toggle switch to validate whether the SSL certificate that the host returns is trusted or not.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Bring Your Own JAR (BYOJ) Process

The following is the procedure to add your own JAR to the SAP HANA database:

1. Log in to the IBM watsonx.data console.
2. From the navigation menu, go to the **Configurations** page and click the **Driver manager** tile.
3. Click **Add driver**.
4. Upload the SAP HANA JAR and specify the driver version. Currently, only one JAR(ngdbc-2.17.12.jar) is supported for SAP HANA database.
5. Click **Add**. Once the driver is successfully added, it undergoes a series of validation. If the validation is successful, it is set to 'inactive' status otherwise it is set to 'failed' status.
6. Click the vertical ellipsis icon to assign or delete the driver.
7. To assign the driver to an engine:
   - Click **Assign**.
   - Select one or more engines to assign the driver. Once assigned, the driver is set to 'active' status.

   Note: You can link the SAP HANA database to the engine only when a driver is associated to that engine. Only one SAP HANA driver can be associated to an engine at a time.
8. To unassign a driver from an engine, users must first introduce another driver.
9. Click **Save and restart engine**.
10. In the **Infrastructure manager**, hover over the SAP HANA database and click the **Manage associations** icon.
11. Select the engine to modify the catalog's association with it. All in-flight queries on the modified engines are stopped.
12. Click **Save and restart engine**.

Limitations for SQL statements

1. `DROP TABLE` statement is supported only when enabled in the catalog.
2. By default, SAP HANA creates VARCHAR columns with a size of 1. So if a base table column is defined with just VARCHAR without specifying a size (e.g., VARCHAR(size)), then `CTAS (Create Table As Select)` operations will not work.
3. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `BLOB` and `CLOB` data types support only `CREATE` and `SELECT` statements.
2. `BINARY` data type supports only `SELECT` statement.
3. The data that is shown for the `BLOB` and `BINARY` data types from the UI is in Base64 format, while the result from presto-cli is in hexadecimal format.
4. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

SingleStore

SingleStore is a relational database management system that is designed for data-intensive applications.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

SingleStore

Configure the following details for SingleStore:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert, or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert, or .cer) link.** iii. Browse the SSL certificate and upload.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.

Field	Description
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `BLOB` and `CLOB` data types support only `SELECT` statement.
2. When the fields of data type `REAL` have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Snowflake

Snowflake is a cloud-hosted relational database for building data warehouse.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Snowflake

Configure the following details for Snowflake:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Warehouse name	Enter your **Warehouse name**.
Username	Enter the **Username**.
Password	Enter the **Password**.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. `CREATE TABLE AS` is also supported for `CREATE TABLE` statement.
2. `DROP TABLE` statement is supported only when enabled in the catalog.
3. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `BLOB` and `CLOB` data types support `SELECT` statement but do not support operations such as `equal`, `like`, and `in`.
2. The data that is shown for the `BLOB` data type from the UI is in Base64 format, while the result from presto-cli is in hexadecimal format.
3. `BINARY` data type supports only `SELECT` statement.
4. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

SQL Server

SQL Server is a relational database management system.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

SQL Server

Configure the following details for SQL Server:

Field	**Description**
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert or .cer)** link. iii. Browse the SSL certificate and upload.

Field	Description
Validate certificate	Use the toggle switch to validate whether the SSL certificate that the host returns is trusted or not.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. `DROP TABLE` statement is supported only when enabled in the catalog.
2. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `BLOB` and `CLOB` data types support `SELECT` statement but do not support operations such as `equal`, `like`, and `in`.
2. The data that is shown for the `BLOB` data type from the UI is in Base64 format, while the result from presto-cli is in hexadecimal format.
3. The data that is shown for the `BINARY` data type from the UI is in Base64 format, while the result from presto-cli is in hexadecimal format.
4. When the fields of data type `REAL` have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Teradata

Teradata is a relational database management system.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Teradata

Configure the following details for Teradata:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.

Field	Description
Authentication type	Choose and enter the **Authentication type** details: • TD2: Enter the username and password. • LDAP: Enter the username and password.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection. If enabled, i. The **Upload SSL certificate (.pem, .crt, .cert or .cer)** link is enabled. ii. Click the **Upload SSL certificate (.pem, .crt, .cert or .cer)** link. iii. Browse the SSL certificate and upload.
Test connection	Click the **Test connection** link to test the database connection. If the database connection is successful, a success message appears.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. `ALTER TABLE` does not drop the first column for `DROP COLUMN` statement.
2. `DROP TABLE` statement is supported only when enabled in the catalog.
3. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. `BLOB` and `CLOB` data types support only `CREATE` and `SELECT` statements.
2. The data that is shown for the `BLOB` data type from the UI is in Base64 format, while the result from presto-cli is in hexadecimal format.
3. `BINARY` data type supports only `SELECT` statement.
4. `VARBYTE` is the `BINARY` alternative data type.
5. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Custom data source

You can now use the Custom data source to create data source connections that are not provided by the built-in data sources. Custom data source can be used for connectors that are supported by Presto as in the Presto documentation but not listed in IBM watsonx.data supported connectors. This feature is applicable for Presto (Java) and Presto (C++) engines. For Presto (C++) engine, only Hive, Apache Iceberg, Arrow Flight service, and Custom data sources can be associated.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

To add a Custom data source, complete the following steps:

1. Log in to watsonx.data console.
2. From the navigation menu, select **Infrastructure manager**.
3. To define and connect a data source, click **Add component**.
4. In the **Data sources** section, select **Custom data source**.
5. Configure the following details:

   Note: Use of this feature may crash your engine if configured incorrectly. IBM does not provide support for use of this feature.

   Note: Two databases with the same name cannot be added. Custom databases do not support SSL configuration properties.

Field	**Description**
Display name	Enter the data source name to be displayed on the screen.
Property	Enter the properties and their values to be configured for the database. Enter the property name:value pair as specified in Presto documentation. You can add multiple properties.
connector.name=	Enter the name of the database connector that you want to add as specified in Presto documentation.
Encryption	Encrypting values of the keys are stored.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the datasource.

You can use the Custom data source for the following connectors in IBM watsonx.data for Presto (Java) engine:

- **Local File connector:** The Local File connector is used to display the http request logs of a worker. Use the custom data source option with the following properties. For more information, see Local File connector.
  - `connector.name=localfile`
  - `presto-logs.http-request-log.location=var/log`
  - `presto-logs.http-request-log.pattern=http-request.log*`
- **Black Hole connector:** The Black Hole connector is designed for high-performance testing of other components. Use the custom data source option with the following property. For more information, see Black Hole connector.
  - `connector.name=blackhole`

Arrow Flight service

You can directly connect to the databases in IBM watsonx.data through the Arrow Flight service in the same CPD cluster. However, for IBM Cloud and IBM Developer edition, Arrow Flight service that is deployed on IBM Cloud is used where you need to enter the details that are related to API key and Flight service URL. This feature is applicable for both Presto (Java) and Presto (C++) engines.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

About this task

Prerequisites:

- The Arrow Flight connector supports only the IBM Arrow Flight service, either on the same CPD cluster that has IBM watsonx.data or on an IBM Cloud VM.
- In CPD, Arrow Flight service should be installed in the same CPD cluster for the data sources to work.
- You need to install Watson Studio in the CPD to get an instance of the Arrow Flight service.
- The Arrow Flight service should be running for databases to work with.

Features and capabilities:

- The databases that are supported through Arrow Flight service have better performance as compared to JDBC databases.
- Basic query pushdown is available for queries across the tables from the same database and across the tables from different flight databases.
- The Arrow Flight service databases support only `SELECT` and `DESCRIBE` queries.

Procedure

1. Log in to IBM watsonx.data console.
2. From the navigation menu, select **Infrastructure manager**.
3. To define and connect a database, click **Add component** and select **Add database**.
4. In the **Add database** window, select a database from the **Database type** drop-down list. The list includes the following database types:
   - "Apache Derby" on page 342.
   - "Greenplum" on page 343.
   - "MariaDB" on page 344.
   - "Salesforce" on page 346.

Apache Derby

Apache Derby is a relational database management system (RDBMS) that can be embedded in Java programs and used for online transaction processing.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Apache Derby

Configure the following details for Apache Derby:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection.

Field	Description
watsonx.data Developer edition Connection details: Arrow Flight	Enter the following details for **Arrow Flight service** connection: **Service hostname**: Enter the service hostname. **Port**: Enter the port number. **API key**: Enter the API key. **Token URL**: Enter the token URL. **Port is SSL enabled**: Use the toggle switch to enable or disable SSL connection. **Validate server certificate**: This option can be used when the host certificate is not signed by a known certificate authority. Toggle the switch to enable or disable the server certificate validation. If enabled, i. The **Upload SSL certificate (.pem, or .crt)** link is enabled. ii. Click the **Upload SSL certificate (.pem, or .crt) link.** iii. Browse the SSL certificate and upload.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Greenplum

Greenplum is a massively parallel processing (MPP) SQL database. It provides access to data in powerful server clusters within a single SQL interface.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Greenplum

Configure the following details for Greenplum:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.

Field	Description
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection.
watsonx.data Developer edition Connection details: Arrow Flight	Enter the following details for **Arrow Flight service** connection: **Service hostname**: Enter the service hostname. **Port**: Enter the port number. **API key**: Enter the API key. **Token URL**: Enter the token URL. **Port is SSL enabled**: Use the toggle switch to enable SSL connection. **Validate server certificate**: This option can be used when the host certificate is not signed by a known certificate authority. To validate that the SSL certificate returned by host is trusted, do the following steps: i. Use the toggle switch to validate the server certificate. ii. The **Upload SSL certificate (.pem, or .crt)** link is enabled. ii. Click the **Upload SSL certificate (.pem, or .crt) link.** iii. Browse the SSL certificate and upload.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

MariaDB

MariaDB is an open-source, multi-threaded, relational database management system (RDBMS). It can be used for data warehousing, e-commerce, enterprise-level features, and logging applications.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

MariaDB

Configure the following details for MariaDB:

Field	**Description**
Display name	Enter the database name to be displayed on the screen.
Database name	Enter the name of your database.
Hostname	Enter the **Hostname**.
Port	Enter the **Port** number.
Username	Enter the **Username**.
Password	Enter the **Password**.
watsonx.data Developer edition Connection details: Arrow Flight	Enter the following details for **Arrow Flight service** connection: **Service hostname**: Enter the service hostname. **Port**: Enter the port number. **API key**: Enter the API key. **Token URL**: Enter the token URL. **Port is SSL enabled**: Use the toggle switch to enable SSL connection. **Validate server certificate**: This option can be used when the host certificate is not signed by a known certificate authority. To validate that the SSL certificate that is returned by the host is trusted, do the following steps: i. Use the toggle switch to validate the server certificate. ii. The **Upload SSL certificate (.pem, or .crt)** link is enabled. ii. Click the **Upload SSL certificate (.pem, or .crt) link.** iii. Browse the SSL certificate and upload.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and 10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Salesforce

Salesforce database is used to securely sync your data in systems that are part of the Salesforce ecosystem.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Salesforce

Configure the following details for Salesforce:

Field	Description
Display name	Enter the database name to be displayed on the screen.
Hostname	Enter the **Hostname**.
Username	Enter the **Username**.
Password	Enter the **Password**.
Port is SSL enabled	Use the toggle switch to enable or disable SSL connection.
watsonx.data Developer edition Connection details: Arrow Flight	Enter the following details for **Arrow Flight service** connection: **Service hostname**: Enter the service hostname. **Port**: Enter the port number. **API key**: Enter the API key. **Token URL**: Enter the token URL. **Port is SSL enabled**: Use the toggle switch to enable or disable SSL connection. **Validate server certificate**: This option can be used when the host certificate is not signed by a known certificate authority. To validate that the SSL certificate returned by host is trusted, do the following steps: i. Use the toggle switch to validate the server certificate. ii. The **Upload SSL certificate (.pem, or .crt)** link is enabled. ii. Click the **Upload SSL certificate (.pem, or .crt) link.** iii. Browse the SSL certificate and upload.
Catalog name	Enter the name of the catalog. This catalog is automatically associated with your database.
Create	Click **Create** to create the database.

Limitations for SQL statements

1. For the database-based catalogs, `CREATE SCHEMA`, `CREATE TABLE`, `DROP SCHEMA`, `DROP TABLE`, `DELETE`, `DROP VIEW`, `ALTER TABLE`, and `ALTER SCHEMA` statements are not available in the **Data manager** UI.

Limitations for data types

1. When the fields of data type REAL have 6 digits or more in the decimal part with the digits being predominately zero, the values when queried are rounded off. It is observed that the rounding off occurs differently based on the precision of the values. For example, a decimal number 1.654 when rounded to 3-digits after the decimal point are the same. Another example is 10.890009 and

10.89000. It is noticed that 10.89000 is rounded to 10.89, whereas 10.89009 is not rounded off. This is an inherent issue because of the representational limitations of binary floating point formats. This might have a significant impact when querying involves sorting.

Updating database credentials

To update the database credentials, use one of the following methods:

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Update the database credentials in list view.
   a) In **Databases** tab, click the overflow menu and then click **Update credentials**.
   b) In the **Update credentials** window, enter your database username and your database password.
   c) Click **Update**.
2. Update the database credentials in topology view.
   a) Hover over the database for which you want to update the credentials.
   b) Click the **Update credentials** icon.
   c) In the **Update credentials** window, enter your database username and your database password.
   d) Click **Update**.

Editing database details

To edit the display name, description, and tags of a database, use one of the following methods:

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Edit the display name, database description and tags in list view.
   a) Click the name of database. Database information window opens.
   b) In the **Details** tab, click **Edit**.
   c) In the **Display name** field, enter the name to be displayed.
   d) In the **Description** field, enter the description of the engine or edit the existing description.
   e) In the **Tags** field, select the tags from list or start typing to define a new tag.
   f) Click **Save**.
2. Edit the display name, database description and tags in topology view.
   a) Click the database. Database information window opens.
   b) In the **Details** tab, click **Edit**.
   c) In the **Display name** field, enter the name to be displayed.
   d) In the **Description** field, enter the description of the engine or edit the existing description.
   e) In the **Tags** field, select the tags from list or start typing to define a new tag.
   f) Click **Save**.

Deleting a database-catalog pair

To delete a database-catalog pair, complete the following steps:

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Dissociate the catalog that is associated with the database from the engine. For instructions, see "Dissociating a catalog from an engine" on page 303.
2. In **Infrastructure manager**, go to the **Databases** tab.
3. Click the overflow menu then click **Remove**.
4. In the **Confirm removal** window, click **Remove**.

Chapter 6. Working with data

You can use watsonx.data to collect, store, query, and analyze all your enterprise data with a single unified data platform.

Note: The out-of-the-box MinIO object storage is provided for exploratory purposes only. It does not have all security features and is not configured to provide high-speed data access. You should register your own s3 bucket that meet your security and performance requirements.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Exploring data

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

About Data manager

The **Data manager** page in IBM watsonx.data is the entry point for you to browse the schemas and tables by engine. You can select an **Engine** by selecting the **Data objects** tab in the **Data manager** page to view the associated catalogs, schemas, and tables. You can search for tables that are loaded under the engine.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

From the **Data manager** page, you can create new schemas and tables by using the **`Create`** option. You can also select a catalog or schema, click the overflow menu, and use the corresponding **`Create`** option to create a schema or table. **`Create table from file`** option in the overflow menu of schema is also used to ingest a data file into watsonx.data. Similarly, schemas and tables can be dropped from the catalogs.

Note: Wait for a few minutes to view the changes after a schema or table is dropped.

You can create **Ingestion jobs** from the **Data manager** page.

Other tasks that can be performed in the **Data manager** page include adding, renaming, or dropping a column.

Note: You must add PostgreSQL and MySQL to watsonx.data network to browse the PostgreSQL database from your own PostgreSQL container (not IBM-LH-PostgreSQL) using watsonx.data **Data Manager**.

You can browse the **Table schema** and up to 25 rows of **Data sample** for some tables. You can view the **Time travel** snapshots and use the **Rollback** feature to rollback or rollforward to any snapshots for Iceberg tables.

Creating schemas

You can create schema from the **Data manager** page by using the web console.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Log in to IBM watsonx.data console.
2. From the navigation menu, select **Data manager**, click **Browse data**.

3. Select the engine from the **Engine** drop-down. Catalogs that are associated with the selected engine are listed.
4. There are two ways to create a schema. Select the required option:

   Option 1: To create a schema under any catalog, do the following steps:

   a) Click **Create**.

   b) Click **Create schema**. The **Create schema** page opens.

   c) Go to step 5.

   Option 2: To create a schema under a particular catalog, do the following steps:

   a) Select a catalog where you want to create a schema.

   b) Click the overflow menu of the selected catalog and select **Create schema**. The **Create schema** page opens.

   c) Go to step 5.
5. In the **Create schema** form, select the catalog. Enter schema name.
6. Click **Create**. The schema is created under the selected catalog.
7. When you name a schema:

   a) Do not wrap the schema name in quotation marks. (Example: Use **`test`** instead of **`“test”`**.)

   b) Do not use semicolon `[;]`, colon `[:]`, single quotation mark `[']`, or double quotation mark `["]` in the schema name.

   c) Do not use leading space in the schema name.

   d) Do not use special character such as question mark (?) or asterisk (*) in schema name.

Creating tables

You can generate, configure, and run DDL from the **Data manager** page by using the web console.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Log in to IBM watsonx.data console.
2. From the navigation menu, select **Data manager**, click **Browse data**.
3. Select engine from the **Engine** menu. Catalogs that are associated with the selected engine are listed.
4. There are two ways to import a file into a table. Select the required option:

   Option 1: To import file to any available schema under a catalog, do the following steps:

   a) Click **Create**.

   b) Click **Create table from file**. The **Create table from a file** page opens.

   c) Go to step 5.

   Option 2: To import file to a particular schema under the catalog, do the following steps:

   a) Select a schema under a catalog where you want to import a file to create table.

   b) Click the overflow menu of the selected schema and select **Create table from a file**. The **Create table from a file** page opens.

   c) Go to step 5.
5. In the **Create table from file** form, drag a file to the box or click to upload.

   Note: Following are the requirements for selecting a file:

   - .CSV, .Parquet, .JSON, .TXT are the supported data file formats
   - Creating a table from a file is only supported by iceberg catalogs.
   - The default file format for Iceberg is Parquet.

- For JSON file, you must enclose the content in `[]`.

  Note: You can apply the configuration for **Encoding**, **Escape character**, **Field delimiter**, and **Line delimiter** prior to uploading any .CSV and .TXT files. Default values are as follows:

  - Encoding value: UTF-8
  - Escape character: \\
  - Field delimiter: ,
  - Line delimiter: \n

6. In the **Target** form, select the **Catalog**, and **Schema** in which the table is created.
7. Enter a name for the table in the **Table name** field, and click **Next**. Do not use special character such as question mark (?) or asterisk (*) in table or column name.
8. Select the **Data format version**.

   Note: Parquet v2 tables are not readable by Presto (C++). To ensure compatibility with Presto (C++), choose Parquet v1.
9. Verify the details in the **Summary** page and scroll down to view the **DDL preview**.
10. Click **Create**.
11. Go to the **Data manager** page and select the schema under which you created the table and click the refresh icon. The newly created table is listed.

    Note: You can customize, add, rename or drop columns from an existing table. You can also rollback your tables.

Ingesting data

Overview of data ingestion in watsonx.data

Data ingestion is the process of importing and loading data into IBM watsonx.data. You can use the **Ingestion jobs** tab from the **Data manager** page to securely and easily load data into watsonx.data console. Alternatively, you can also load or ingest local data files to create tables using the **Create table** option.

watsonx.data on Red Hat OpenShift

When you ingest a data file into the watsonx.data, the table schema is generated and inferred when a query is run.

Data ingestion in watsonx.data supports CSV and Parquet formats. The files to be ingested must be of the same format type and same schema. watsonx.data auto discovers the schema based on the source file being ingested.

Following are some of the requirements or behavior of the ingestion tool:

- Schema evolution is not supported.
- Target table must be in Iceberg format.
- IBM Storage Ceph, IBM Cloud Object Storage (COS), AWS S3, and MinIO object storage are supported.
- `pathStyleAccess` property for object storage is not supported.
- Only Parquet and CSV file formats are supported as source data files.

Loading or ingesting data through CLI

An ingestion job in watsonx.data can be run with the **`ibm-lh`** tool. The tool must be pulled from the ibm-lh-client and installed in the local system to run the ingestion job through the CLI. For more details and instructions to install ibm-lh-client package and use the **`ibm-lh`** tool for ingestion, see Installing ibm-lh-client and Setting up the ibm-lh command-line utility.

The **`ibm-lh`** tool supports the following features:

- Auto-discovery of schema based on the source file or target table.
- Advanced table configuration options for the CSV files:
  - Delimiter
  - Header
  - File encoding
  - Line delimiter
  - Escape characters
- Ingestion of single, multiple files, or single folder (no sub folders) of S3 and local Parquet files.
- Ingestion of single, multiple files, or single folder (no sub folders) of S3 and local CSV files.

Ingesting data from object storage bucket

In this tutorial, you learn to move data into a data lake or an object storage bucket and load the data files to Presto. You learn to optimize the file format to choose the table format and run complex SQL query in IBM watsonx.data.

watsonx.data on Red Hat OpenShift

Before you begin

This tutorial requires:

- watsonx.data must be installed.
- The configuration details of data bucket that you bring in. This is required for establishing connection with the watsonx.data.
- Ensure that the data bucket has data.

About this task

Scenario: You need to run SQL query on data files that is in your object storage bucket. For this, you must attach the data files in your object storage bucket to Presto. You can also convert data into an optimized analytical format in Parquet or ORC to enhance query performance and reduce server and storage resource consumption. Now, you can run SQL query against the table you created.

The objectives of this tutorial are listed as follows:

- Creating infrastructure within the watsonx.data service.
- Establishing connection with the customer data bucket.
- Querying from the bucket

Procedure

1. **Uploading data into an object storage bucket and attaching to Presto**

   In this section of the tutorial, you are going to manage data in an object storage bucket and attach the bucket to HMS and associate with Presto (Java) engine.

   a. Access any one of the object storage access tools like S3 Browser, AWS S3 console, direct S3 APIs, and various CLI/UI object storage tools.

   b. Load data files to your object storage bucket by using the tool.

   c. Register and attach the object storage bucket to HMS and associate with Presto (Java) engine by using watsonx.data UI.

   d. Alternatively, you can also register and attach an object storage bucket with pre-existing data to HMS.

2. **Load data files into Presto**

   After you attach the object storage bucket to HMS, you need to load data files into Presto (Java) by creating schema and external tables through the Hive connector.

   a. Run the following command to create schema for the data you want to access.

      ```
      CREATE SCHEMA <SCHEMA_NAME> WITH ( location = '<SCHEMA_LOCATION>' );
      ```

      For example:

      ```
      CREATE SCHEMA hive.gosales WITH ( location = 's3a://lhbeta/gosales' );
      ```

   b. Run the following command to create table by using an external location by pointing to an uploaded data file.

      ```
      CREATE TABLE IF NOT EXISTS <TABLE_NAME> ("<COLUMN_NAMES>" <DATA_TYPE>) WITH ( format =
      '<DATA_FORMAT>', external_location = '<DATA_FILE_LOCATION>' );
      ```

      For example:

      ```
      CREATE TABLE IF NOT EXISTS hive.gosales.branch ("BRANCH_CODE" int, "ADDRESS1" varchar,
      "ADDRESS1_MB" varchar, "ADDRESS2" varchar, "ADDRESS2_MB" varchar, "CITY" varchar,
      "CITY_MB" varchar, "PROV_STATE" varchar, "PROV_STATE_MB" varchar, "POSTAL_ZONE" varchar,
      "COUNTRY_CODE" int, "ORGANIZATION_CODE" varchar, "WAREHOUSE_BRANCH_CODE" int) WITH ( format
      = 'CSV', external_location = 's3a://lhbeta/gosales/branch' );
      ```

3. **Generate statistics with analyze table.**

   If you want to use the data without creating a new copy for a different table format or more table optimizations, you can generate statistics alone with analyze table.

   a. To generate statistics with analyze table, run the following command:

      ```
      analyze <TABLE_NAME>;
      ```

      For example:

      ```
      analyze hive.gosales.branch;
      ```

4. **Convert data to analytics optimized formats (Optional)**

   You can use the data for creating different table format and more table optimizations. It is recommended to convert the data files to analytics optimized format in Parquet or ORC to improve query performance, reduce server and storage resource consumption. Table format like Iceberg can provide more performance improvements and features like snapshots, time travel, and transactional support for insert, update, and delete.

   a. To create table for a data in CSV format to Parquet format, run Create table as command:

      ```
      CREATE TABLE IF NOT EXISTS
      <TABLE_NAME>
      WITH ( format = 'PARQUET')
      AS
      SELECT *
      FROM <TABLE_NAME>;
      ```

      For example:

      ```
      CREATE TABLE IF NOT EXISTS
      hive.default.branch
      WITH ( format = 'PARQUET')
      AS
      SELECT *
      FROM hive.gosales.branch;
      ```

b. To change the table format to Iceberg, run `Create table as` command:

```
CREATE TABLE IF NOT EXISTS
<TABLE_NAME>
WITH ( format = 'PARQUET')
AS
SELECT *
FROM <TABLE_NAME>;
```

For example:

```
CREATE TABLE IF NOT EXISTS
iceberg-beta.default.branch
WITH ( format = 'PARQUET')
AS
SELECT *
FROM hive.gosales.branch;
```

Note:

You can also include any additional SQL into the select clause for any transformations or conversion business logic or sort the data for optimized access. You can also add column partitions for more performance improvements.

Note: Statistics are automatically generated as part of the ingest of the new table.

Preparing for data ingestion

This topic guides you through efficiently ingesting data manually from an external object storage into your IBM watsonx.data for querying. We support IBM Storage Ceph, IBM Cloud Object Storage (COS), AWS S3, and MinIO as the object storage buckets.

Parquet and CSV are the supported file types.

You can ingest Parquet files directly for optimal performance and CSV files require a staging directory for conversion to Parquet format.

watsonx.data on Red Hat OpenShift

Before you begin

- S3 folder must be created with data files in it for ingesting. The best way to create an S3 folder is by using AWS CLI. The source folder must contain either all parquet file or all CSV files. Use AWS CLI to avoid hidden "0-byte" files that can cause ingestion issues. For detailed information on S3 folder creation, refer to https://docs.aws.amazon.com/AmazonS3/latest/userguide/using-folders.html.
- Staging folder must be specified for CSV files, individual file ingestion (Parquet or CSV) and local Parquet folders. Staging folder is not required for all files in an S3 folder (source folder ingestion). The exception for this case is when there are type differences between different types of parquet files in the S3 folder or when TIME data type is involved.
- For ingestion job through CLI, the staging bucket must be the same bucket that is associated with the Hive catalog. Staging is possible only in the Hive catalog.

About this task

Scenario: You have a collection of data files in an S3 folder that you need to ingest into your IBM database. You need to run SQL query on data files that is in your object storage bucket.

The objectives of this tutorial are listed as follows:

- Creating infrastructure within the watsonx.data service.
- Establishing connection with the customer data bucket.
- Querying from the bucket

Note: You can use the Spark ingestion through the web console.

Note: For detailed information on the usage of different parameters, see "Options and parameters supported in ibm-lh tool" on page 359, and for ingesting data files into watsonx.data by using Spark CLI, commands and configuration file, see "Spark ingestion through ibm-lh tool command line" on page 377, "Creating an ingestion job by using commands" on page 370, and "Creating an ingestion job by using the configuration file" on page 373.

Procedure

Ingesting Parquet or CSV files from an S3 Folder

In this section, you have a collection of Parquet/CSV files in an S3 folder that you need to ingest into your IBM database.

a. Prepare the source S3 folder:
   - Use AWS CLI to copy the Parquet /CSV files into a common S3 folder. Avoid creating empty folders through the console to prevent hidden 0-byte files.

b. Specify staging directory (For CLI ingestion):
   - Provide the `staging-location` parameter to designate a staging directory for CSV or specific Parquet files to Parquet conversion. The ingest tool will create it if it does not exist.

     Note: See "Staging location" on page 364 for more details.

c. Create schema file to specify CSV file properties:
   - Provide the `schema` parameter to specify CSV file properties such as field delimiter, line delimiter, escape character, encoding and whether header exists in the CSV file.

     Note: See "Schema file specifications" on page 365 for more details.

d. Initiate server-mode ingestion:
   - Employ the CLI (server-mode) to start the ingestion process.

e. CSV or specific Parquet to Parquet conversion:
   - The ingest tool converts the specific Parquet or CSV files into Parquet format and stores them in the staging directory.

Results

- Optimizes data transfer performance.
- Simplifies the ingestion process.
- Provides clear troubleshooting in case of errors.

Ingesting data by using web console

You can ingest data into IBM watsonx.data by using the web console. Ingestion through web console is supported only by using a Spark engine.

watsonx.data on Red Hat OpenShift

Ingesting data by using Spark

You can ingest data into IBM watsonx.data through the web console. Ingestion through web console is supported only by using IBM Analytics Engine (Spark).

watsonx.data on Red Hat OpenShift

Before you begin

- You must have the **Administrator** role and privileges in the catalog to do ingestion through the web console.

- Register an IBM Analytics Engine (Spark). See “Registering an external engine” on page 300.
- Add storage for the source data files and target catalog. See “Adding a storage-catalog pair” on page 306.
- Optionally, you can create a schema in the catalog for the target table. See “Creating schemas” on page 349.
- Optionally, you can also create a target table in the schema. See “Creating tables” on page 350.
- To enable your Spark application to work with the watsonx.data catalog and storage, you must have `Metastore admin` role. Without `Metastore admin` privilege, you cannot ingest data to storage using Native Spark engine. To enable your Spark application to work with the watsonx.data catalog and storage, add the following configuration to your application payload:

```
spark.hive.metastore.client.plain.username=ibmlhapikey
spark.hive.metastore.client.plain.password=<api-key-of-the-user-which-has-metastore-admin-
role>
spark.hadoop.wxd.apiKey=Basic base64(ibmlhapikey_ibmcloudid:apikey)
```

Procedure

1. Log in to the watsonx.data console.
2. From the navigation menu, select **Data manager** and open the **Ingest data** window in one of the following ways:
   a. Select the **Ingest data** tab and click **Create ingestion job**. The **Ingest data** window opens with an auto-generated job ID.
   b. From the **Data objects** tab, select a table from the list of tables under the schema and click **Ingest data**. The **Ingest data** window opens with an auto-generated job ID.

      Note: You can ingest data to an existing table. The existing table and the new data to be ingested must be of the same format type. For example, you can ingest a CSV file to an existing table created from a CSV file and can ingest Parquet files to a table created from Parquet file.
3. If required, modify the auto-generated ingestion job ID in the **Enter job ID** field.
4. Select the IBM Analytics Engine (Spark) from the **Select engine** list. The registered Spark engines are listed here.
5. Select a pre-defined resource size or customize your own from the options listed:
   a. Small: File data is under 100 GB. Driver memory (GB): 2GB, Driver cores: 1 vCPU, Number of executors: 1, Executor cores: 1 vCPU, and Executor memory (GB): 2GB.
   b. Medium: File data is between 100 GB - 500 GB. Driver memory (GB): 4GB, Driver cores: 2 vCPU, Number of executors: 2, Executor cores: 2 vCPU, and Executor memory (GB): 4GB.
   c. Large: File data is above 500 GB. Driver memory (GB): 8GB, Driver cores: 4 vCPU, Number of executors: 4, Executor cores: 4 vCPU, and Executor memory (GB): 8GB.
   d. Custom: Configure the five (5) resources for your job based on your needs.
6. Click **Next**.
7. In the **Select file(s)** tab, browse and select your remote file(s) from your connected bucket(s) by selecting **Select data from bucket** and to browse and select your file(s) from your local system, select **Select data from system**.
   a. From **Select data from bucket** option, click **Bucket** drop-down and select the bucket from where you want to ingest the data. Go to step 10.
   b. From **Select data from system** option, drag a file to the box or click to upload.
8. Select the required file type based on the source data. The available options are CSV and Parquet.
9. From the source directory, select the source data files to be ingested and click **Next**.

   Note: You can apply the configuration for **Header**, **Encoding**, **Escape character**, **Field delimiter**, and **Line delimiter** for the CSV files.

10. View the selected files and the corresponding file previews in the **File(s) selected** and **File preview** tabs. File preview enables to preview first 10 rows of the selected source file.
11. In the **Target** tab, select the target catalog from the **Select catalog** list.
12. Select one of the schema options:

    a. **Existing schema**: To ingest source data into an existing schema. Corresponding target schemas are listed in the **Select schema** drop-down.

    b. **New schema**: Enter the target schema name in **Schema name** to create a new schema from the source data.
13. Select the corresponding **Target table** options based on the selection in step 12.
    - **Existing table**:To ingest source data into an existing table. Corresponding target tables are listed in the **Select table** drop-down.
    - **New table**: Enter the target table name in **Table name** to create a new table from the source data.
14. Click **Next**.
15. Validate the details in the summary page. Click **Ingest**.

    Note: To cancel an ingestion job, click the overflow menu of the **Job ID** from the **Ingest data** page and click **Cancel**. Some canceled ingestion jobs might show `Failed` status even if the cancellation is successful at the backend.

Limitations

The following are some of the limitations of Spark ingestion:

- Spark ingestion through UI can ingest files from local system with a maximum size of 500 MB only.
- The default buckets in watsonx.data are not exposed to the Spark engine. Hence, **`iceberg-bucket`** and **`hive-bucket`** are not supported for source or target table. Users can use their own MinIo or S3 compatible buckets that are exposed and accessible by the Spark engine.

Advanced options to customize configmap for ingestion

Users can customize the configmap settings to adjust the CPU and memory value. When you run an ingestion job, parameters that control the number of CPUs and the size of memory that is allocated can be specified.

watsonx.data on Red Hat OpenShift

About this task

When huge files are ingested, it requires more CPU and memory to load the files. By default, 4 CPUs and 4 GB memory is allocated for bulk ingestion jobs through the web console. To override the default value based on your requirement, complete the following steps.

Note: Updating the configmap does not affect the in-progress ingestion jobs. The resource allocation changes are picked in the new jobs.

Procedure

1. Log in to `ocp` by using one of the following options:

   a) Run the following command to log in to the cluster by providing a username and password:

   ```
   ibm-lakehouse-manage login-to-ocp \
   --user=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

   b) Run the following command to log in to the cluster by providing a token:

```
ibm-lakehouse-manage login-to-ocp \
--server=${OCP_URL} \
--token=${OCP_TOKEN}
```

2. To update the CPU and memory values, enter the required values for the `INGEST_CPU_RESOURCE` and `INGEST_MEMORY_SIZE` fields in the following command.

   ```
   oc patch cm ibm-lh-<INSTANCE_NAME>-ingest-config-cm -n <PROJECT_CPD_INST_OPERANDS > --type 
   merge -p '{"data":{
     "INGEST_JOB_CONFIG": "{\"INGEST_SYSTEM_CFG\":{\"INGEST_CPU_RESOURCE\": \"8\", 
   \"INGEST_MEMORY_SIZE\": \"16G\"}}"}
   }'

   INGEST_CPU_RESOURCE: <new value>
   INGEST_MEMORY_SIZE: <new value>
   ```

   `INSTANCE_NAME` is the name of the watsonx.data instance that is installed. The instance name can be obtained from the console under `Instances` menu.

Using a certification authority (CA) certificate of external object storage for ingestion

You can use a custom-signed CA certificate to establish a secure connection between Cloud Pak for Data pods and services for watsonx.data ingestion.

watsonx.data on Red Hat OpenShift

The pods are injected with a secret mount (`cpd-custom-ca-certs`) that contains the CA certificate and environment variables that points to the mount path. The CA certificate is merged with the built-in CA certificate and injected to the pods.

Procedure

1. Connect to the external object store over `https:` Cloud Pak for Data.

   For more information, see Connecting to external object stores over https: Cloud Pak for Data.
2. Install the Cloud Pak for Data configuration admission controller for the Cloud Pak for Data namespace by using the **`manage install-cpd-config-ac`** command.

   ```
   cpd-cli manage install-cpd-config-ac \
   --cpd_instance_ns=<project-name> \
   [--cpd_config_ac_image=<image-location-and-name>] \
   [--preview=true|false] \
   [-v][-vv][-vvv]
   ```

   For more information, see **`manage install-cpd-config-ac`**.
3. Enable the Cloud Pak for Data configuration admission controller in the specified namespace and other tethered namespaces (if any) using the **`manage enable-cpd-config-ac`** command.

   ```
   cpd-cli manage enable-cpd-config-ac \
   --cpd_instance_ns=<project-name> \
   [-v][-vv][-vvv]
   ```

   For more information, see **`manage enable-cpd-config-ac`**.
4. Run the **`oc cli`** command to create the `cpd-custom-ca-certs` secret. Include a `--from-file` entry for each certificate that you want to include in the secret. For example:

   ```
   oc create secret generic cpd-custom-ca-certs \
   --namespace=${PROJECT_CPD_INST_OPERANDS} \
   --from-file=<file-name-1>.crt=<fully-qualified-cert-file-name-1> \
   --from-file=<file-name-2>.crt=<fully-qualified-cert-file-name-2>
   ```
5. Update the `cpd-custom-ca-certs` secret with the contents of custom certificates.

```
cpd-cli manage gen-platform-ca-certs \
--cpd_instance_ns=${PROJECT_CPD_INST_OPERANDS} \
--apply=true
```

Important: Injecting the secret into the pods might take some time. Complete this action during a maintenance window or before you give users access to this instance of Cloud Pak for Data. However, services cannot use the certificates in the secret until you inject the secret into the Cloud Pak for Data pods.

6. To verify that the secrets are injected successfully, run the **`manage list-platform-ca-certs-pods`** command.

   ```
   cpd-cli manage list-platform-ca-certs-pods \
   --cpd_instance_ns=<project-name>
   ```

   For more information, see **`manage list-platform-ca-certs-pods`**.

What to do next

Proceed with ingesting data by using Spark.

Ingesting data by using command line interface (CLI)

You can run the **`ibm-lh`** tool to ingest data into IBM watsonx.data through the command line interface (CLI).

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Options and parameters supported in ibm-lh tool

Ingesting data files from S3 or local location into IBM watsonx.data is done by using the **`ibm-lh`** tool. The parameters supported in the **`ibm-lh`** tool are described in this topic.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Command options

Before you begin:

Set the mandatory environment variable for `ENABLED_INGEST_MODE` before starting an ingestion job. Based on the ingestion mode, specify appropriate value for the variable. Following is an example for Spark.

```
export ENABLED_INGEST_MODE=SPARK
```

The following ingestion modes are supported:

- `PRESTO`
- `SPARK_LEGACY`
- `SPARK`

Note: SPARK is the default mode.

Sample scripts and config file templates are available for CLI ingestion commands in the `/Ingestion-Examples/` folder in the `ibm-lh` tool container. To access the example scripts and config files, run the following command:

1. Run `./ibm-lh data-copy --interactive` to enter the container in interactive mode.
2. Run `cd /Ingestion-Examples/` to access the example scripts and config files.

Options and parameters

Parameter	Description	Declaration	Modes of ingestion
`cert-file-path`	To verify CPD certificate.	`--cert-file-path <path for certificate>`	PRESTO, SPARK_LEGACY and SPARK
`create-if-not-exist`	Create target table if it does not exist.	`--create-if-not-exist`	PRESTO, SPARK_LEGACY, and SPARK
`dbpassword`	Database password that is used to do ingestion. This is a mandatory parameter to run an ingestion job unless the default user is used.	`--dbpassword <DBPASSWORD>`	PRESTO
`dbuser`	Database username that is used to do ingestion. This is a mandatory parameter to run an ingestion job unless the default user is used.	`--dbuser <DBUSER>`	PRESTO
`debug`	Debug the logs of ingestion jobs. The short command for this parameter is `-d`.	`--debug`	PRESTO, SPARK_LEGACY, and SPARK
`engine-id`	Engine ID of the Spark engine when doing REST API based SPARK ingestion. The short command for this parameter is `-e`.	`--engine-id <spark-enginename>`	SPARK
`ingest-config`	Configuration file for data migration.	`--ingest-config <INGEST_CONFIGFILE>`	PRESTO and SPARK_LEGACY
`ingestion-engine-endpoint`	Endpoint of ingestion engine. hostname=`<hostname>`, port=`<port>`. This is a mandatory parameter to run an ingestion job.	`--ingestion-engine-endpoint <INGESTION_ENGINE_ENDPOINT>`	PRESTO and SPARK_LEGACY
`instance-id`	Identify unique instances. The instance ID is available in the URL of the instance or from the instance details page (Click the information icon on the watsonx.data instance UI screen). The short command for this parameter is `-i`.	`--instance-id <instance ID>`	SPARK

Parameter	Description	Declaration	Modes of ingestion
`job-id`	Job ID that is generated when REST API or UI based ingestion is initiated. The job ID is used to get the status of an ingestion job. This parameter is used only with `ibm-lh get-status` command in the interactive mode of ingestion. The short command for this parameter is `-j`	`ibm-lh get-status --job-id <Job id>`	SPARK
`all-jobs`	Get history of all the ingestion jobs. This parameter is used only with `ibm-lh get-status` command in the interactive mode of ingestion.	`ibm-lh get-status --all-jobs`	SPARK
`log-directory`	Specify the location of the log files. For more details, see "Log directory" on page 365.	`--ingest-config <ingest_config_file> --log-directory <directory_path>`	PRESTO, SPARK_LEGACY, and SPARK
`partition-by`	Supports the functions for year, month, day, and hour for timestamp in the `partition-by` list. If a target table already exists or the `create-if-not-exist` parameter is not specified, partition-by does not effect the data.	`ibm-lh data-copy --partition-by "<columnname1>, <columnname2>"`	SPARK_LEGACY and SPARK
`password`	Password of the user connecting to the instance. In SaaS, API key of the instance is used. The short command for this parameter is `-pw`.	`--password <apikey>`	SPARK
`schema`	Schema file that includes CSV specifications, and more. For more details, see "Schema file specifications" on page 365.	`--schema </path/to/schemaconfig/file>`	PRESTO, SPARK_LEGACY, and SPARK

Parameter	Description	Declaration	Modes of ingestion
`source-data-files`	Data files or folders for data migration. File name ending with / is considered a folder. Single or multiple files can be used. This is a mandatory parameter to run an ingestion job. Example: `<file1_path>,<file2_path>,<folder1_path>`. File names are case sensitive. The short command for this parameter is `-s`.	`--source-data-files <SOURCE_DATA_FILE>`	PRESTO, SPARK_LEGACY, and SPARK
`staging-location`	Location where CSV files and in some circumstances Parquet files are staged. For more information, see "Staging location" on page 364. This is a mandatory parameter to run an ingestion job.	`--staging-location <STAGING_LOCATION>`	PRESTO
`staging-hive-catalog`	If the default catalog for staging is not used, use this parameter to specify the name of the Hive catalog that is configured in watsonx.data. The default catalog is `hive_data`.	`--staging-hive-catalog <catalog_name>`	PRESTO
`staging-hive-schema`	The schema name associated with the staging hive catalog for ingestion. Create and pass in a custom schema name by using this parameter. Default schema: `lhingest_staging_schema`. If schema is created as default, this parameter is not required.	`--staging-hive-schema <schema_name>`	PRESTO

Parameter	Description	Declaration	Modes of ingestion
`sync-status`	This parameter is used in REST API based ingestion. Default value is `false`. When this parameter is set to `true`, `ibm-lh data-copy` tool waits and polls to get continuous status after an ingestion job is submitted.	`--sync-status <IS THERE ANY ENTRY?>`	SPARK
`system-config`	This parameter is used to specify system related parameters. For more information see "System config" on page 364.	`--system-config <path/to/system/configfile>`	PRESTO, SPARK_LEGACY, and SPARK
`target-catalog-uri`	Target catalog uri	`--target-catalog-uri <TARGET_CATALOG_URI>`	SPARK_LEGACY
`target-table`	Data migration target table. `<catalog>.<schema>.<table1>`. This is a mandatory parameter to run an ingestion job. Example: `<iceberg.demo.customer1>`. The short command for this parameter is `-t`. For more information see "Target table" on page 366.	`--target-table <TARGET_TABLE>`	PRESTO, SPARK_LEGACY, and SPARK
`trust-store-path`	Path of the truststore to access the ingestion engine. This is used to establish SSL connections. This parameter is mandatory for non-root user.	`--trust-store-path <TRUST_STORE_PATH>`	PRESTO and SPARK_LEGACY
`trust-store-password`	Password of truststore to access the ingestion engine. This is used to establish SSL connections. This parameter is mandatory for non-root user.	`--trust-store-password <TRUST_STORE_PASSWORD>`	PRESTO and SPARK_LEGACY

Parameter	Description	Declaration	Modes of ingestion
`user`	User name of the user connecting to the instance. The short command for this parameter is `-u`.	`--user <username>`	SPARK
`url`	Base url of the location of watsonx.data cluster. The short command for this parameter is `-w`.	`--url <url>`	SPARK

System config

The `system-config` parameter refers to a file and is used to specify system related parameters.

For the command line, the parameter is declared as follows:

```
--system-config /path/to/systemconfig/file
```

The format of the system config parameter is as follows:

```
[system-config]
<param_name1>:<param_val>
<param_name2>:<param_val>
<param_name3>:<param_val>
...
```

Currently, only the memory-limit parameter is supported. This parameter specifies the maximum memory in watsonx.data that an ingestion job can use. Default value for memory-limit is 500M. The limit can be in bytes, K, M or G. The `system-config` is applicable for PRESTO, SPARK_LEGACY, and SPARK ingestion modes.

Following are some examples of how the `memory-limit` parameter can be specified in the `system-config` file.

```
[system-config]
memory-limit:500M

[system-config]
memory-limit:5000K

[system-config]
memory-limit:1G

[system-config]
memory-limit:10000000 #This is in bytes
```

Note: The `memory-limit` parameter is applicable for PRESTO ingestion mode.

Staging location

Staging location parameter is applicable for PRESTO ingestion mode. The staging location is used for:

- CSV file or folder ingestion
- Local Parquet file or folder ingestion.
- S3 Parquet file ingestion
- In some circumstances, when the source file or files in the S3 Parquet folder contains special column types, such as TIME or are associated with different column types.

Important: For ingestion job through CLI, the staging bucket must be the same bucket that is associated with the Hive catalog. Staging is possible only in the Hive catalog.

Note: The internal MinIO buckets in CPD deployments (iceberg-data, hive-data, wxd-milvus, wxd-system) and their associated catalogs cannot be used for staging, as their endpoints are not externally accessible. Users can use their own storage buckets that are exposed and accessible by external connections.

Schema file specifications

The schema parameter points to the schema file. The schema file can be used to specify CSV file properties such as field delimiter, line delimiter, escape character, encoding and whether header exists in the CSV file. This parameter is applicable for PRESTO, SPARK_LEGACY, and SPARK ingestion modes.

The following is the schema file specification:

```
[CSV]
DELIMITER:<delim> #default ','

#LINE_DELIMITER:
#A single char delimiter other than ' '(blank), need not be enclosed in quotes.
#Must be enclosed in quotes if it is one of:  '\n' for newline, '\t' for TAB, ' ' for space.
LINE_DELIMITER:<line_delim> #default '\n'

HEADER:<true|false> #default 'true'
#HEADER is a mandatory entry within schema file.

#single character value
ESCAPECHAR:<escape_char>   #default '\\'

#Encoding (Example:"utf-8")
ENCODING:<encoding>    #default None
```

The encoding values supported by Presto and Spark ingestion are directly dependent on encoding value supported by Python.

Note: The encoding value must be enclosed in single quotation marks.

Note: The delimiter value must be enclosed in single quotation marks.

The following is an example of schema specification:

```
$ more /tmp/schema.cfg
[CSV]
DELIMITER:','
HEADER:false
LINE_DELIMITER:'\n'
```

Log directory

The ingest log files are generated in the log directory. By default, the ingested log file is generated as `/tmp/ingest.log`. By using the `--log-directory` parameter, you can specify a new location for ingest log files. A separate log file is created for each ingest command invocation. The new log file name is in the format `ingest_<timestamp)_<pid>.log`. The log directory must exist before invocation of the `ibm-lh` ingest tool.

This parameter is applicable only in the command line option for PRESTO, SPARK_LEGACY, and SPARK ingestion modes.

Example by using command line:

```
ibm-lh data-copy --source-data-files s3://cust-bucket/warehouse/a_source_file1.csv,s3://cust-
bucket/warehouse/a_source_file2.csv
--staging-location s3://cust-bucket/warehouse/staging/
--target-tables iceberg_target_catalog.ice_schema.cust_tab1
--ingestion-engine-endpoint "hostname=localhost,port=8080"
--create-if-not-exist
--log-directory /tmp/mylogs
```

Example when using a config file:

```
ibm-lh data-copy --ingest-config ext.cfg --log-directory /tmp/mylogs
```

Target table

The ability to handle special characters in table and schema names for ingestion is constrained by the underlying engines (Presto, Legacy Spark, Spark) and the special characters they support. When using schema or table names with special characters, not all special characters will be accepted or handled by Spark, Presto, Legacy Spark. Consult the documentation for the special characters support.

The SQL identifier of the target table for data migration is `<catalog>.<schema>.<table>`. Use double quotes " or backticks ` to escape parts with special characters.

Examples:

```
ibm-lh data-copy --target-table 'catalog."schema 2.0"."my table!"'
```

```
ibm-lh data-copy --target-table 'catalog.schema 2.0.my table!'
```

```
ibm-lh data-copy --target-table catalog.'"schema 2.0"'.'"my table!"'
```

```
ibm-lh data-copy --target-table "catalog.\schema 2.0`.`my table!`"`
```

```
ibm-lh data-copy --target-table catalog.\"schema\ 2.0\".\"my\ table!\"
```

Note: Both double quotes " and backticks ` are accepted, but quote styles cannot be mixed. In order to include a literal quote inside an identifier, double the quoting character (example, "" or ``.

Presto ingestion through `ibm-lh` tool command line

Ingesting data files from S3 or local location into IBM watsonx.data is done by using two options that are supported in the **`ibm-lh`** tool.

- **`Command line`** option
- **`Configuration file`** option

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Before you begin:

1. Set the mandatory environment variable `ENABLED_INGEST_MODE` to `PRESTO` before starting an ingestion job by running the following command:

   ```
   export ENABLED_INGEST_MODE=PRESTO
   ```

2. Set the environment variables for `SOURCE_S3_CREDS` and `STAGING_S3_CREDS` based on the requirements before starting an ingestion job by running the following commands:

   ```
   export
   SOURCE_S3_CREDS="AWS_ACCESS_KEY_ID=,AWS_SECRET_ACCESS_KEY=,ENDPOINT_URL=,AWS_REGION=,BUCKET_N
   AME="
   ```

   ```
   export
   STAGING_S3_CREDS="AWS_ACCESS_KEY_ID=,AWS_SECRET_ACCESS_KEY=,ENDPOINT_URL=,AWS_REGION=,BUCKET_
   NAME="
   ```

Different options and variables that are supported in a command line and configuration file are listed as follows:

- **`Command line`** option

Parameter	Description	Declaration
create-if-not-exist	Create target table if it does not exist.	`--create-if-not-exist`
dbpassword	Database password that is used to do ingestion. This is a mandatory parameter to run an ingestion job unless the default user is used.	`--dbpassword <DBPASSWORD>`
dbuser	Database username that is used to do ingestion. This is a mandatory parameter to run an ingestion job unless the default user is used.	`--dbuser <DBUSER>`
ingest-config	Configuration file for data migration	`--ingest-config <INGEST_CONFIGFILE>`
ingestion-engine-endpoint	Endpoint of ingestion engine. hostname=`<hostname>`, port=`<port>`. This is a mandatory parameter to run an ingestion job.	`--ingestion-engine-endpoint <INGESTION_ENGINE_ENDPOINT>`
log-directory	This option is used to specify the location of log files.	`--ingest-config <ingest_config_file> --log-directory <directory_path>`
schema	Schema file that includes CSV specifications, and more.	`--schema </path/to/schemaconfig/file>`
source-data-files	Data files or folders for data migration. File name ending with / is considered a folder. Single or multiple files can be used. This is a mandatory parameter to run an ingestion job. File names are case sensitive. Example: `<file1_path>,<file2_path>,<folder1_path>`	`--source-data-files <SOURCE_DATA_FILE>`
staging-location	Location where CSV files and in some circumstances Parquet files are staged. This is a mandatory parameter to run an ingestion job.	`--staging-location <STAGING_LOCATION>`
staging-hive-catalog	The Hive catalog name configured in the watsonx.data, if not using the default catalog for staging. Default catalog: hive_data.	`--staging-hive-catalog <catalog_name>`

Parameter	Description	Declaration
staging-hive-schema	The schema name associated with the staging Hive catalog for ingestion. Create and pass in a custom schema name by using this parameter. Default schema: `lhingest_staging_schema`. If schema is created as default, you do not have need to specify this parameter.	`--staging-hive-schema <schema_name>`
system-config	This parameter is used to specify system related parameters.	`--system-config <path/to/system/configfile>`
target-table	Data migration target table. `<catalog>.<schema>.<table1>`. This is a mandatory parameter to run an ingestion job. Example: `<iceberg.demo.customer1>`	`--target-table <TARGET_TABLES>`
trust-store-path	Path of the truststore to access the ingestion engine. This is used to establish SSL connections. This parameter is optional and deprecated.	`--trust-store-path <TRUST_STORE_PATH>`
trust-store-password	Password of truststore to access the ingestion engine. This is used to establish SSL connections. This parameter is optional and deprecated.	`--trust-store-password <TRUST_STORE_PASSWORD>`

- **`Configuration file` option**

  The **Configuration file** contains a global ingest configuration section and multiple individual ingest configuration sections to run the ingestion job. The specifications of the individual ingestion sections override the specifications of the global ingestion section.

  – Global ingest config section

Parameter	Description	Declaration
create-if-not-exist	Create a target table if not existed.	`create-if-not-exist:<true>`
ingestion-engine-endpoint	Specifies connection parameters of the ingestion engine. Endpoint of ingestion engine. hostname=`<hostname>`,port=`<port>`	`ingestion-engine:hostname=<hostname>, port=<port>`
target-table	Data migration target table. Only one target table can be specified. `<catalog>.<schema>.<table1>`	`target-table:<table_name>`

  – Individual ingest config section

Parameter	Description	Declaration
create-if-not-exist	Create target table if it does not exist.	`create-if-not-exist`

Parameter	Description	Declaration
dbpassword	Database password that is used to do ingestion. This is a mandatory parameter to run an ingestion job unless the default user is used.	`dbpassword:<DBPASSWORD>`
dbuser	Database username that is used to do ingestion. This is a mandatory parameter to run an ingestion job unless the default user is used.	`dbuser:<DBUSER>`
ingestion-engine-endpoint	Endpoint of ingestion engine. hostname=`<hostname>`, port=`<port>`. This is a mandatory parameter to run an ingestion job.	`ingestion-engine-endpoint:<INGESTION_ENGINE_ENDPOINT>`
schema	Schema file that includes CSV specifications, and more.	`schema:/path/to/schemaconfig/file`
source-files	Data files or folders for data migration. File name ending with / is considered a folder. This is a mandatory parameter to run an ingestion job.	`source-files:<SOURCE_DATA_FILE>`
staging-location	Location where CSV files and in some circumstances Parquet files are staged. This is a mandatory parameter to run an ingestion job.	`staging-location:<STAGING_LOCATION>`
staging-hive-catalog	The Hive catalog name configured in the watsonx.data, if not using the default catalog for staging. Default catalog: hive_data.	`--staging-hive-catalog <catalog_name>`
staging-hive-schema	The schema name associated with the staging Hive catalog for ingestion. Create and pass in a custom schema name by using this parameter. Default schema: `lhingest_staging_schema`. If schema is created as default, you do not have need to specify this parameter.	`--staging-hive-schema <schema_name>`
system-config	This parameter is used to specify system related parameters.	`--system-config <path/to/system/configfile>`
target-catalog-uri	Target catalog uri	`target-catalog-uri:<TARGET_CATALOG_URI>`

Parameter	Description	Declaration
target-table	Data migration target table. `<catalog>.<schema>.<table1>`. This is a mandatory parameter to run an ingestion job. Example: `<iceberg.demo.customer1>`	`target-table:<TARGET_TABLES>`
target-table-storage	Target table file storage location	`target-table-storage:<TARGET_TABLE_STORAGE>`
trust-store-path	Path of truststore to access ingestion engine. This is used to establish SSL connections. This parameter is optional and deprecated.	`trust-store-path:<TRUST_STORE_PATH>`
trust-store-password	Password of truststore to access the ingestion engine. This is used to establish SSL connections. This parameter is optional and deprecated.	`trust-store-password:<TRUST_STORE_PASSWORD>`

Creating an ingestion job by using commands

You can run the **`ibm-lh`** tool to ingest data into IBM watsonx.data through the command line interface (CLI). This topic provides details of using the command line for ingestion in the Presto ingestion mode.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Before you begin

Set the mandatory environment variable `ENABLED_INGEST_MODE` to `PRESTO` before starting an ingestion job by running the following command:

```
export ENABLED_INGEST_MODE=PRESTO
```

Set the environment variables for `SOURCE_S3_CREDS` and `STAGING_S3_CREDS` based on the requirements before starting an ingestion job by running the following commands:

```
export
SOURCE_S3_CREDS="AWS_ACCESS_KEY_ID=xxxxxxx,AWS_SECRET_ACCESS_KEY=yyyyyyyy,ENDPOINT_URL=https://
s3.jp-tok.cloud-object-storage.appdomain.cloud,AWS_REGION=us-east-1,BUCKET_NAME=cust-bucket"
```

```
export
STAGING_S3_CREDS="AWS_ACCESS_KEY_ID=xxxxxxx,AWS_SECRET_ACCESS_KEY=yyyyyyyy,ENDPOINT_URL=https://
s3.jp-tok.cloud-object-storage.appdomain.cloud,AWS_REGION=us-east-1,BUCKET_NAME=cust-bucket"
```

About this task

Following are the details of the command line option to ingest data files from S3 or local location to watsonx.data Iceberg table:

Note: The commands must run within the **`ibm-lh`** container. For more details and instructions to install the **`ibm-lh-client`** package and use the **`ibm-lh`** tool for ingestion, see Installing ibm-lh-client and Setting up the ibm-lh command-line utility.

Note: To access IBM Cloud Object Storage (COS) and MinIO object storage, specify the ENDPOINT_URL to pass the corresponding url to the tool. For more information about IBM COS, see https://cloud.ibm.com/docs/cloud-object-storage?topic=cloud-object-storage-endpoints.

Note: Replace the absolute values in the command examples with the values applicable to your environment. See "Options and parameters supported in ibm-lh tool" on page 359.

1. **Ingest a single CSV/Parquet file from S3 location by using command.**

   To ingest a single CSV/Parquet file from a S3 location, run the following command:

   ```
   ibm-lh data-copy --source-data-files SOURCE_DATA_FILE \
   --staging-location s3://lh-target/staging \
   --target-table TARGET_TABLES \
   --ingestion-engine-endpoint INGESTION_ENGINE_ENDPOINT \
   --dbuser DBUSER \
   --dbpassword DBPASSWORD \
   --create-if-not-exist
   ```

   For example:

   ```
   ibm-lh data-copy --source-data-files s3://cust-bucket/warehouse/a_source_file.parquet \
   --staging-location s3://cust-bucket/warehouse/staging/ \
   --target-table iceberg_target_catalog.ice_schema.cust_tab1 \
   --ingestion-engine-endpoint "hostname=localhost,port=8080" \
   --create-if-not-exist
   ```

2. **Ingest multiple CSV/Parquet files and CSV folders from S3 location by using command.**

   To ingest multiple Parquet files from a S3 location, run the following command:

   ```
   ibm-lh data-copy --source-data-files SOURCE_DATA_FILE \
   --staging-location s3://lh-target/staging \
   --target-table TARGET_TABLES \
   --ingestion-engine-endpoint INGESTION_ENGINE_ENDPOINT \
   --dbuser DBUSER \
   --dbpassword DBPASSWORD \
   --create-if-not-exist
   ```

   For examples:

   ```
   ibm-lh data-copy --source-data-files s3://cust-bucket/warehouse/a_source_file1.csv,s3://cust-
   bucket/warehouse/a_source_file2.csv \
   --staging-location s3://cust-bucket/warehouse/staging/ \
   --target-table iceberg_target_catalog.ice_schema.cust_tab1 \
   --ingestion-engine-endpoint "hostname=localhost,port=8080" \
   --create-if-not-exist
   ```

   ```
   ibm-lh data-copy --source-data-files s3://cust-bucket/warehouse/ \
   --staging-location s3://cust-bucket/warehouse/staging/ \
   --target-table iceberg_target_catalog.ice_schema.cust_tab1 \
   --ingestion-engine-endpoint "hostname=localhost,port=8080" \
   --create-if-not-exist
   ```

3. **Ingest all Parquet files in a folder from S3 location by using command.**

   To ingest all Parquet files in a folder from a S3 location, run the following command:

   ```
   ibm-lh data-copy --source-data-files SOURCE_DATA_FILE \
   --target-table TARGET_TABLES \
   --ingestion-engine-endpoint INGESTION_ENGINE_ENDPOINT \
   --dbuser DBUSER \
   --dbpassword DBPASSWORD \
   --create-if-not-exist
   ```

   For example:

```
ibm-lh data-copy --source-data-files s3://cust-bucket/warehouse/ \
--target-table iceberg_target_catalog.ice_schema.cust_tab1 \
--ingestion-engine-endpoint "hostname=localhost,port=8080" \
--create-if-not-exist
```

Note: In general, this option does not require a staging location. However, a few exceptional scenarios are there when a staging location must be specified. When the staging location is not used, make sure that the hive catalog configured with Presto can be used with source-data-files location. The following are the exceptional cases where a staging location is required:

- Any or all parquet files in the folder are huge.
- Any or all parquet files in the folder have special columns, such as TIME.

4. **Ingest a CSV/Parquet file or folder from a local file system by using command.**

   To ingest a single Parquet file from a local location, run the following command:

```
ibm-lh data-copy --source-data-files SOURCE_DATA_FILE \
--staging-location s3://lh-target/staging \
--target-table TARGET_TABLES \
--ingestion-engine-endpoint INGESTION_ENGINE_ENDPOINT \
--dbuser DBUSER \
--dbpassword DBPASSWORD \
--create-if-not-exist
```

   For examples:

```
ibm-lh data-copy --source-data-files /cust-bucket/warehouse/a_source_file1.parquet \
--staging-location s3://cust-bucket/warehouse/staging/ \
--target-table iceberg_target_catalog.ice_schema.cust_tab1 \
--ingestion-engine-endpoint "hostname=localhost,port=8080" \
--create-if-not-exist
```

```
ibm-lh data-copy --source-data-files /cust-bucket/warehouse/ \
--staging-location s3://cust-bucket/warehouse/staging/ \
--target-table iceberg_target_catalog.ice_schema.cust_tab1 \
--ingestion-engine-endpoint "hostname=localhost,port=8080" \
--create-if-not-exist
```

5. **Ingest any data file from local file system by using a command.**

   To ingest any data file from a local location, run the following command:

   Note: To ingest any type of data files from a local file system, data files are needed to be copied to `~ /ibm-lh-client/localstorage/volumes/ibm-lh` directory. Now, you can access data files from `/ibmlhdata/` directory by using the `ibm-lh data-copy` command.

```
ibm-lh data-copy --source-data-files SOURCE_DATA_FILE \" \
--staging-location s3://lh-target/staging \
--target-table TARGET_TABLES \
--staging-hive-catalog <catalog_name> \
--schema <SCHEMA> \
--ingestion-engine-endpoint INGESTION_ENGINE_ENDPOINT \
--trust-store-path <TRUST_STORE_PATH> \
--trust-store-password <TRUST_STORE_PASSWORD> \
--dbuser DBUSER \
--dbpassword DBPASSWORD \
--create-if-not-exist
```

   For examples:

```
ibm-lh data-copy --source-data-files /ibmlhdata/reptile.csv \
--staging-location  s3://watsonx.data/staging \
--target-table iceberg_data.ivt_sanity_test_1.reptile \
--staging-hive-catalog hive_test \
--schema /ibmlhdata/schema.cfg \
--ingestion-engine-endpoint "hostname=ibm-lh-lakehouse-presto-01-presto-svc-cpd-
instance.apps.ivt384.cp.fyre.ibm.com,port=443" \
--trust-store-path /mnt/infra/tls/aliases/ibm-lh-lakehouse-presto-01-presto-svc-cpd-
instance.apps.ivt384.cp.fyre.ibm.com:443.crt \
--trust-store-password changeit \
--dbuser xxxx\
--dbpassword xxxx \
--create-if-not-exist
```

6. **Ingest CSV or local Parquet or S3 Parquet files that use staging location.**

   To ingest CSV/local Parquet/S3 Parquet files that use staging location:

   ```
   ibm-lh data-copy --source-data-files SOURCE_DATA_FILE \
   --staging-location s3://lh-target/staging \
   --target-table TARGET_TABLES \
   --staging-hive-catalog <catalog_name> \
   --staging-hive-schema <schema_name> \
   --ingestion-engine-endpoint INGESTION_ENGINE_ENDPOINT \
   --trust-store-path <TRUST_STORE_PATH> \
   --trust-store-password <TRUST_STORE_PASSWORD> \
   --dbuser DBUSER \
   --dbpassword DBPASSWORD \
   --create-if-not-exist
   ```

   For examples:

   ```
   ibm-lh data-copy --source-data-files s3://watsonx-data-0823-2/test_icos/GVT-DATA-C.csv \
   --staging-location  s3://watsonx.data-staging \
   --target-table iceberg_data.test_iceberg.gvt_data_v \
   --staging-hive-catalog staging_catalog \
   --staging-hive-schema staging_schema \
   --ingestion-engine-endpoint "hostname=ibm-lh-lakehouse-presto-01-presto-svc-cpd-
   instance.apps.ivt384.cp.fyre.ibm.com,port=443" \
   --trust-store-path /mnt/infra/tls/aliases/ibm-lh-lakehouse-presto-01-presto-svc-cpd-
   instance.apps.ivt384.cp.fyre.ibm.com:443.crt \
   --trust-store-password changeit \
   --dbuser xxxx\
   --dbpassword xxxx \
   --create-if-not-exist
   ```

   Here, `--staging-location` is `s3://watsonx.data-staging`. The `--staging-hive-catalog` that is `staging_catalog` must be associated with the bucket `watsonx.data-staging`.

Creating an ingestion job by using the configuration file

You can run the **`ibm-lh`** tool to ingest data into IBM watsonx.data through the CLI. The config file to run the ingestion job and examples are listed in this topic.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Before you begin

Set the mandatory environment variable `ENABLED_INGEST_MODE` to `PRESTO` before starting an ingestion job by running the following command:

```
export ENABLED_INGEST_MODE=PRESTO
```

Set the environment variables for `SOURCE_S3_CREDS` and `STAGING_S3_CREDS` based on the requirements before starting an ingestion job by running the following commands:

```
export
SOURCE_S3_CREDS="AWS_ACCESS_KEY_ID=xxxxxxx,AWS_SECRET_ACCESS_KEY=yyyyyyyy,ENDPOINT_URL=https://
s3.jp-tok.cloud-object-storage.appdomain.cloud,AWS_REGION=us-east-1,BUCKET_NAME=cust-bucket"
```

```
export
STAGING_S3_CREDS="AWS_ACCESS_KEY_ID=xxxxxxx,AWS_SECRET_ACCESS_KEY=yyyyyyyy,ENDPOINT_URL=https://
s3.jp-tok.cloud-object-storage.appdomain.cloud,AWS_REGION=us-east-1,BUCKET_NAME=cust-bucket"
```

About this task

To run the ingestion jobs, you can use the configuration file option. The advantage of using a configuration file is that, multiple ingestion jobs can be completed in a single starting of the ingestion tool.

Run the following command to do multiple ingestion jobs after you update the configuration file:

```
ibm-lh data-copy --ingest-config /<your_ingest_configfilename>
```

Note: The commands must run within the **`ibm-lh`** container. For more details and instructions to install the **`ibm-lh-client`** package and use the **`ibm-lh`** tool for ingestion, see Installing ibm-lh-client and Setting up the ibm-lh command-line utility.

Note: To access IBM Cloud Object Storage (COS) and MinIO object storage, specify the ENDPOINT_URL to pass the corresponding url to the tool. For more information about IBM COS, see https://cloud.ibm.com/docs/cloud-object-storage?topic=cloud-object-storage-endpoints.

Note: Replace the absolute values inside angular brackets of command examples with values applicable to your environment. See "Options and parameters supported in ibm-lh tool" on page 359.

1. **Ingest a single CSV/Parquet file from S3 location by using a config file.**

   To ingest a single Parquet file from a S3 location, run the following command:

   ```
   [global-ingest-config]
   target-table:table_name
   ingestion-engine:hostname=<hostname>,port=<port>
   create-if-not-exist:<true>
   [ingest-config1]
   source-files:SOURCE_DATA_FILE
   staging-location:STAGING_LOCATION
   ```

   For example:

   ```
   [global-ingest-config]
   target-table:iceberg_cat.ice_schema.customer1_tab
   ingestion-engine:hostname=localhost,port=8080
   create-if-not-exist:true

   [ingest-config1]
   source-files:s3://cust-bucket/warehouse/a_source_file.parquet
   staging-location:s3://cust-bucket/warehouse/staging/
   ```

2. **Ingest multiple CSV/Parquet files and CSV folders from S3 location by using a config file.**

   To ingest multiple Parquet files from a S3 location, run the following command:

   ```
   [global-ingest-config]
   target-table:table_name
   ingestion-engine:hostname=<hostname>,port=<port>
   create-if-not-exist:<true>
   [ingest-config1]
   source-files:SOURCE_DATA_FILE
   staging-location:STAGING_LOCATION
   ```

   For examples:

```
[global-ingest-config]
target-table:iceberg_cat.ice_schema.customer1_tab
ingestion-engine:hostname=localhost,port=8080
create-if-not-exist:true

[ingest-config1]
source-files:s3://cust-bucket/warehouse/a_source_file1.csv,s3://cust-bucket/warehouse/
a_source_file2.csv
staging-location:s3://cust-bucket/warehouse/staging/
```

```
[global-ingest-config]
target-table:iceberg_cat.ice_schema.customer1_tab
ingestion-engine:hostname=localhost,port=8080
create-if-not-exist:true

[ingest-config1]
source-files:s3://cust-bucket/warehouse/
staging-location:s3://cust-bucket/warehouse/staging/
```

3. **Ingest all Parquet files in a folder from S3 location by using a config file.**

   To ingest all Parquet files in a folder from a S3 location, run the following command:

   ```
   [global-ingest-config]
   target-table:table_name
   ingestion-engine:hostname=<hostname>,port=<port>
   create-if-not-exist:<true>
   [ingest-config1]
   source-files:SOURCE_DATA_FILE
   ```

   For example:

   ```
   [global-ingest-config]
   target-table:iceberg_cat.ice_schema.customer1_tab
   ingestion-engine:hostname=localhost,port=8080
   create-if-not-exist:true

   [ingest-config1]
   source-files:s3://cust-bucket/warehouse/
   ```

   Note: In general, this option does not require a staging location. However, a few exceptional scenarios are there when a staging location must be specified. When the staging location is not used, make sure that the hive catalog configured with Presto can be used with source-files location. The following are the exceptional cases where a staging location is required:

   - Any or all parquet files in the folder are huge.
   - Any or all parquet files in the folder have special columns, such as TIME.

4. **Ingest a CSV/Parquet file or a folder of files from a local file system by using a config file.**

   To ingest a single CSV file from a local location, run the following command:

   ```
   [global-ingest-config]
   target-table:table_name
   ingestion-engine:hostname=<hostname>,port=<port>
   create-if-not-exist:<true>
   [ingest-config1]
   source-files:SOURCE_DATA_FILE
   staging-location:STAGING_LOCATION
   ```

   For examples:

   ```
   [global-ingest-config]
   target-table:iceberg_cat.ice_schema.customer1_tab
   ingestion-engine:hostname=localhost,port=8080
   create-if-not-exist:true

   [ingest-config1]
   source-files:/tmp/customer1.parquet
   staging-location:s3://cust-bucket/warehouse/staging/
   ```

```
[global-ingest-config]
target-table:iceberg_cat.ice_schema.customer1_tab
ingestion-engine:hostname=localhost,port=8080
create-if-not-exist:true

[ingest-config1]
source-files:/tmp/
staging-location:s3://cust-bucket/warehouse/staging/
```

5. **Ingest any data file from local file system by using a config file.**

   To ingest any data file from a local location, run the following command:

   Note: To ingest any type of data files from a local file system, data files are needed to be copied to `~ /ibm-lh-client/localstorage/volumes/ibm-lh` directory. Now, you can access data files from `/ibmlhdata/` directory by using the `ibm-lh data-copy` command.

   ```
   [global-ingest-config]
   target-table:table_name
   ingestion-engine:hostname=<hostname>,port=<port>
   create-if-not-exist:<true>

   [ingest-config1]
   source-files:SOURCE_DATA_FILE
   staging-location:STAGING_LOCATIONstaging-hive-catalog:<catalog_name>
   schema:<SCHEMA>
   dbuser:<DBUSER>
   dbpassword:<DBPASSWORD>
   trust-store-path:<TRUST_STORE_PATH>
   trust-store-password:<TRUST_STORE_PASSWORD>
   ```

   For examples:

   ```
   [global-ingest-config]
   target-table:iceberg_data.ivt_sanity_test_1.reptile
   ingestion-engine:hostname=ibm-lh-lakehouse-presto-01-presto-svc-cpd-
   instance.apps.ivt384.cp.fyre.ibm.com,port=443
   create-if-not-exist:true

   [ingest-config1]
   source-files:/ibmlhdata/reptile.csv
   staging-location:s3://watsonx.data/staging
   staging-hive-catalog:hive_test
   schema:schema.cfg
   dbuser:xxxx
   dbpassword:xxxx
   trust-store-path:/mnt/infra/tls/aliases/ibm-lh-lakehouse-presto-01-presto-svc-cpd-
   instance.apps.ivt384.cp.fyre.ibm.com:443.crt
   trust-store-password:changeit
   ```

6. **Ingest CSV/local Parquet/S3 Parquet files that use staging location.**

   To ingest CSV/local Parquet/S3 Parquet files that use staging location:

   ```
   [global-ingest-config]
   target-table:table_name
   ingestion-engine:hostname=<hostname>,port=<port>
   create-if-not-exist:<true>

   [ingest-config1]
   source-files:SOURCE_DATA_FILE
   staging-location:STAGING_LOCATION
   staging-hive-catalog:<catalog_name>
   schema:<SCHEMA>
   dbuser:<DBUSER>
   dbpassword:<DBPASSWORD>
   trust-store-path:<TRUST_STORE_PATH>
   trust-store-password:<TRUST_STORE_PASSWORD>
   ```

   For examples:

```
[global-ingest-config]
target-table:iceberg_data.test_iceberg.gvt_data_v
ingestion-engine:hostname=ibm-lh-lakehouse-presto-01-presto-svc-cpd-
instance.apps.ivt384.cp.fyre.ibm.com,port=443
create-if-not-exist:true

[ingest-config1]
source-files:s3://watsonx-data-0823-2/test_icos/GVT-DATA-C.csv
staging-location:s3://watsonx.data-staging
staging-hive-catalog:staging_catalog
staging-hive-schema:staging_schema
dbuser:xxxx
dbpassword:xxxx
trust-store-path:/mnt/infra/tls/aliases/ibm-lh-lakehouse-presto-01-presto-svc-cpd-
instance.apps.ivt384.cp.fyre.ibm.com:443.crt
trust-store-password:changeit
```

Here, `staging-location` is `s3://watsonx.data-staging`. The `staging-hive-catalog` that is `staging_catalog` must be associated with the bucket `watsonx.data-staging`.

Spark ingestion through `ibm-lh` tool command line

You can run the **`ibm-lh`** tool to ingest data into IBM watsonx.data through the command line interface (CLI) using the IBM Analytics Engine (Spark). The commands to run the ingestion job are listed in this topic.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Before you begin

- You must have the **Administrator** role and privileges in the catalog to do ingestion through the web console.
- Add and register IBM Analytics Engine (Spark). See "Registering an external engine" on page 300.
- Add bucket for the target catalog. See "Adding a storage-catalog pair" on page 306.
- Create schema and table in the catalog for the data to be ingested. See "Creating schemas" on page 349 and "Creating tables" on page 350.

Procedure

1. Set the mandatory environment variable `ENABLED_INGEST_MODE` to `SPARK_LEGACY` before starting an ingestion job by running the following command:

   ```
   export ENABLED_INGEST_MODE=SPARK_LEGACY
   ```

2. Set the following environment variables before starting an ingestion job by running the following commands:

   ```
   export IBM_LH_BEARER_TOKEN=<token>
   export IBM_LH_SPARK_JOB_ENDPOINT=https://<cpd_url>/v4/analytics_engines/<instance_id>/
   spark_applications
   export HMS_CLIENT_USER=lakehouse
   export HMS_CLIENT_PASSWORD=<instance secret>
   export
   SOURCE_S3_CREDS="AWS_ACCESS_KEY_ID=*******,AWS_SECRET_ACCESS_KEY=*******,ENDPOINT_URL=<endpoi
   nt_url>,AWS_REGION=<region>,BUCKET_NAME=<bucket_name>"
   export
   TARGET_S3_CREDS="AWS_ACCESS_KEY_ID=*******,AWS_SECRET_ACCESS_KEY=*******,ENDPOINT_URL=<endpoi
   nt_url>,AWS_REGION=<region>,BUCKET_NAME=<bucket_name>"
   export IBM_LH_SPARK_EXECUTOR_CORES=<value>
   export IBM_LH_SPARK_EXECUTOR_MEMORY=<value>
   export IBM_LH_SPARK_EXECUTOR_COUNT=<value>
   export IBM_LH_SPARK_DRIVER_CORES=<value>
   export IBM_LH_SPARK_DRIVER_MEMORY=<value>
   export INSTANCE_ID=<instance_id>
   export IBM_LH_URL=https://<lh_hostname>
   ```

```
export USE_NATIVE_SPARK=<true/false>
export USE_EXTERNAL_SPARK=<true/false>
```

Note: If IBM Analytics Engine serverless instance on IBM Cloud is registered as external Spark on watsonx.data, the Spark driver, executor vCPU and memory combinations must be in a 1:2, 1:4, or 1:8 ratio. See Default limits and quotas for Analytics Engine instances.

Environment variable name	Description
IBM_LH_BEARER_TOKEN	Authorization bearer token. For more information, see Get authorization token.
IBM_LH_SPARK_JOB_ENDPOINT	Spark applications v4 endpoint for CPD and v3 endpoint for SaaS. • Refer to Step 1 in document: Managing Analytics Engine Powered by Apache Spark instances to retrieve CPD Spark Endpoint
HMS_CLIENT_USER	User for Hive Metastore client. CPD Spark implementation uses `lakehouse`.
HMS_CLIENT_PASSWORD	Password for Hive Metastore client. For CPD, Run the following command: `oc exec -it <lhconsole-api-pod-name> -- cat /mnt/infra/ibm-lh-secrets/LH_INSTANCE_SECRET`
SOURCE_S3_CREDS	S3 credentials for the source file bucket in the format: `“AWS_ACCESS_KEY_ID=<access_key>,AWS_SECRET_ACCESS_KEY=<secret_key>,ENDPOINT_URL=<endpoint_url>,AWS_REGION=<region>,BUCKET_NAME=<bucket_name>”`
TARGET_S3_CREDS	S3 credentials for the target table bucket in the format: `“AWS_ACCESS_KEY_ID=<access_key>,AWS_SECRET_ACCESS_KEY=<secret_key>,ENDPOINT_URL=<endpoint_url>,AWS_REGION=<region>,BUCKET_NAME=<bucket_name>”`
IBM_LH_SPARK_EXECUTOR_CORES	Optional spark engine configuration setting for executor cores
IBM_LH_SPARK_EXECUTOR_MEMORY	Optional spark engine configuration setting for executor memory
IBM_LH_SPARK_EXECUTOR_COUNT	Optional spark engine configuration setting for executor count
IBM_LH_SPARK_DRIVER_CORES	Optional spark engine configuration setting for driver cores
IBM_LH_SPARK_DRIVER_MEMORY	Optional spark engine configuration setting for driver memory
INSTANCE_ID	Identify unique instances. The instance ID is available in the URL of the instance or from the instance details page (Click the information icon on the watsonx.data instance UI screen).
USE_NATIVE_SPARK	When native spark is used for ingestion, thigh parameter value must be `true`.
USE_EXTERNAL_SPARK	When external spark is used for ingestion, thigh parameter value must be `true`.

Environment variable name	Description
IBM_LH_URL	This parameter is used only when `USE_EXTERNAL_SPARK=true`. The value is `https://<lh_hostname>`. `<lh_hostname>` is the hostname of CPD instance.

3. You can run ingestion jobs to ingest data in 2 ways, using a simple command line or a config file.

   a. Run the following command to ingest data from single or multiple source data files:

```
ibm-lh data-copy --source-data-files s3://path/to/file/or/folder \
--target-table <catalog>.<schema>.<table> \
--ingestion-engine-endpoint "hostname=<hostname>,port=<port>,type=spark" \
--trust-store-password <truststore password> \
--trust-store-path <truststore path> \
--log-directory /tmp/mylogs \
--partition-by "<columnname1>, <columnname2>" \
--cert-file-path /root/ibm-lh-manual/ibm-lh/cert.pem \
--target-catalog-uri 'thrift://<hms_thrift_uri>'
```

   Where the parameters used are listed as follows:

Parameter	Description
`--source-data-files`	Path to S3 parquet or CSV file or folder. Folder paths must end with "/". File names are case sensitive.
`--target-table`	Target table in format `<catalogname>.<schemaname>.<tablename>`.
`--ingestion-engine-endpoint`	Ingestion engine endpoint is in the format `hostname='',port='',type=spark`". Type must be set to `spark`.
`--log-directory`	This option is used to specify the location of log files.
`--partition-by`	This parameter supports the functions for years, months, days, hours for timestamp in the `partition-by` list. If a target table already exist or the `create-if-not-exist` parameter is not mentioned the partition-by shall not make any effect on the data.
`--trust-store-password`	Password of the truststore certificate inside the spark job pod. Current password for Spark in CPD and SaaS is `changeit`
`--trust-store-path`	Path of the truststore cert inside the spark job pod. Current path of Spark in CPD and SaaS is `file:///opt/ibm/jdk/lib/security/cacerts`
`--target-catalog-uri`	HMS thrift endpoint. • CPD endpoint example: `thrift://<metastore_host_value>` • SaaS endpoint example: `thrift://<metastore_host_value>` <metastore_host_value> is taken from the details tab of the catalog in the **Infrastructure** page.

Parameter	Description
`--cert-file-path`	To verify CPD certificate.
`--create-if-not-exist`	Use this option if the target schema or table is not created. Do not use if the target schema or table is already created.
`--schema`	Use this option with value in the format `path/to/csvschema/config/file`. Use the path to a schema.cfg file which specifies header and delimiter values for CSV source file or folder.

b. Run the following command to ingest data from a config file:

```
ibm-lh data-copy --ingest-config /<your_ingest_configfilename>
```

Where the config file has the following information:

```
[global-ingest-config]
target-tables:<catalog>.<schema>.<table>
ingestion-engine:hostname='',port='',type=spark
create-if-not-exist:true/false

[ingest-config1]
source-files:s3://path/to/file/or/folder
target-catalog-uri:thrift://<hms_thrift_uri>
trust-store-path:<truststore path>
trust-store-password:<truststore password>
log-directory /tmp/mylogs
partition-by "<columnname1>, <columnname2>"
cert-file-path /root/ibm-lh-manual/ibm-lh/cert.pem
schema:/path/to/csvschema/config/file [Optional]
```

The parameters used in the config file ingestion job is listed as follows:

Parameter	Description
`source-files`	Path to s3 parquet or CSV file or folder. Folder paths must end with "/". File names are case sensitive.
`target-table`	Target table in format <catalogname>.<schemaname>.<tablename>.
`ingestion-engine`	Ingestion engine endpoint is in the format `hostname='',port='',type=spark`". Type must be set to `spark`.
`--partition-by`	This parameter supports the functions for years, months, days, hours for timestamp in the `partition-by` list. If a target table already exist or the `create-if-not-exist` parameter is not mentioned the partition-by shall not make any effect on the data.
`trust-store-password`	Password of the truststore certificate inside the spark job pod. Current password for Spark in CPD and SaaS is `changeit`
`trust-store-path`	Path of the truststore cert inside the spark job pod. Current path of Spark in CPD and SaaS is `file:///opt/ibm/jdk/lib/security/cacerts`

Parameter	Description
`target-catalog-uri`	HMS thrift endpoint. • CPD endpoint example: `thrift://<metastore_host_value>` • SaaS endpoint example: `thrift://<metastore_host_value>` <metastore_host_value> is taken from the details tab of the catalog in the **Infrastructure** page.
`--cert-file-path`	To verify CPD certificate.
`create-if-not-exist`	Use this option if the target schema or table is not created. Do not use if the target schema or table is already created.
`schema`	Use this option with value in the format `path/to/csvschema/config/file`. Use the path to a schema.cfg file which specifies header and delimiter values for CSV source file or folder.
`log-directory`	This option is used to specify the location of log files.

Note: The ability to handle special characters in table and schema names for ingestion is constrained by the underlying engines (Spark, Presto) and their respective special character support.

Regular syntax: `--target-tables <catalogname>.<schemaname>.<tablename>`.

Syntax with special character option 1: `--target-tables <catalogname>.<schemaname>."table\.name"`. Using this syntax, escape character `` `\` `` is used within double quotes to escape period(.). Escape character `` `\` `` is used only when special character period(.) is in the table name.

Syntax with special character option 2: `--target-tables <catalogname>.<schemaname>."'table.name'"`. Using this syntax, period(.) is not escaped nor need to use the escape character when using additional single quotes.

Limitations

Following are some of the limitations of Spark ingestion:

- Spark ingestion supports only source data files from object storage bucket. Local files are not supported.
- The default buckets in watsonx.data are not exposed to Spark engine. Hence, **`iceberg-bucket`** and **`hive-bucket`** are not supported for source or target table. Users can use their own MinIO or S3 compatible buckets that are exposed and accessible by Spark engine.

Spark REST API ingestion through `ibm-lh` tool command line

You can run the **`ibm-lh`** tool to ingest data into IBM watsonx.data through the command line interface (CLI) using the IBM Analytics Engine (Spark) REST API. This CLI based ingestion uses REST endpoint to do the ingestion. This is the default mode of ingestion. The commands to run the ingestion job are listed in this topic.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Before you begin

- You must have the **Administrator** role and privileges in the catalog to do ingestion through the web console.
- Add and register IBM Analytics Engine (Spark). See "Registering an external engine" on page 300.
- Add bucket for the target catalog. See "Adding a storage-catalog pair" on page 306.
- Create schema and table in the catalog for the data to be ingested. See "Creating schemas" on page 349 and "Creating tables" on page 350.

Procedure

1. Set the mandatory environment variable ENABLED_INGEST_MODE to SPARK before starting an ingestion job by running the following command:

```
export ENABLED_INGEST_MODE=SPARK
```

2. Set the following environment variables before starting an ingestion job by running the following commands:

```
export IBM_LH_SPARK_EXECUTOR_CORES=1
export IBM_LH_SPARK_EXECUTOR_MEMORY=2G
export IBM_LH_SPARK_EXECUTOR_COUNT=1
export IBM_LH_SPARK_DRIVER_CORES=1
export IBM_LH_SPARK_DRIVER_MEMORY=2G
export CPD_VERIFY_CERTIFICATE=<true/false>
```

Note: If IBM Analytics Engine serverless instance on IBM Cloud is registered as external Spark on watsonx.data, the Spark driver, executor vCPU and memory combinations must be in a 1:2, 1:4, or 1:8 ratio. See Default limits and quotas for Analytics Engine instances.

Environment variable name	Description
IBM_LH_SPARK_EXECUTOR_CORES	Optional spark engine configuration setting for executor cores
IBM_LH_SPARK_EXECUTOR_MEMORY	Optional spark engine configuration setting for executor memory
IBM_LH_SPARK_EXECUTOR_COUNT	Optional spark engine configuration setting for executor count
IBM_LH_SPARK_DRIVER_CORES	Optional spark engine configuration setting for driver cores
IBM_LH_SPARK_DRIVER_MEMORY	Optional spark engine configuration setting for driver memory
CPD_VERIFY_CERTIFICATE	To turn ON certificate verification, set the variable to `true`.

3. Run the following command to ingest data from a single or multiple source data files:

```
ibm-lh data-copy --target-table iceberg_data.ice_schema.ytab \
--source-data-files "s3://lh-ingest/hive/warehouse/folder_ingestion/" \
--user admin \
--password **** \
--url https://cpd-cpd-instance.apps.cpd-ocp-wxd-fips.cp.fyre.ibm.com \
--instance-id 1719823250083405 \
--schema /home/nz/config/schema.cfg \
--engine-id spark214 \
--log-directory /tmp/mylogs \
--partition-by "<columnname1>, <columnname2>" \
--cert-file-path /root/ibm-lh-manual/ibm-lh/cert.pem \
--create-if-not-exist
```

Where the parameters used are listed as follows:

Parameter	**Description**
`--cert-file-path`	To verify CPD certificate. This parameter is used when `CPD_VERIFY_CERTIFICATE=true`.
`--create-if-not-exist`	Use this option if the target schema or table is not created. Do not use if the target schema or table is already created.
`--engine-id`	Engine id of Spark engine when using REST API based SPARK ingestion.
`--instance-id`	Identify unique instances. In SaaS environment, CRN is the instance id.
`--log-directory`	This option is used to specify the location of log files.
`--password`	Password of the user connecting to the instance. In SaaS, API key to the instance is used.
`--partition-by`	This parameter supports the functions for years, months, days, hours for timestamp in the `partition-by` list. If a target table already exist or the `create-if-not-exist` parameter is not mentioned the partition-by shall not make any effect on the data.
`--schema`	Use this option with value in the format path/to/csvschema/config/file. Use the path to a schema.cfg file which specifies header and delimiter values for CSV source file or folder.
`--source-data-files`	Path to s3 parquet or CSV file or folder. Folder paths must end with “/”. File names are case sensitive.
`--target-table`	Target table in format `<catalogname>.<schemaname>.<tablename>`.
`--user`	User name of the user connecting to the instance.
`--url`	Base url of the location of IBM® watsonx.data cluster.

4. Run the following command to get the status of the ingest job:

```
ibm-lh get-status --job-id <Job-id> --instance-id --url --user --password
```

Where the parameter used is listed as follows:

Parameter	**Description**
`--job-id<Job id>`	Job id is generated when REST API or UI based ingestion is initiated. This job id is used in getting the status of ingestion job. This parameter is used only used with `ibm-lh get-status` command. The short command for this parameter is `-j`

5. Run the following command to get the history of all ingestion jobs:

```
ibm-lh get-status --all-jobs --instance-id --url --user --password
```

Where the parameter used is listed as follows:

Parameter	Description
`--all-jobs`	This `all-jobs` parameter gives the history of all ingestion jobs. This parameter is used only used with `ibm-lh get-status` command.

Note: `get-status` is supported with `ibm-lh` only in the interactive mode of ingestion.

Querying data

Running SQL queries

SQL is a standardized language for defining and manipulating data in a relational database. You can use the Query workspace interface in IBM watsonx.data to run SQL queries and scripts against your data.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

About this task

To run the SQL queries, do the following steps:

Procedure

1. Log in to the watsonx.data console.
2. From the navigation menu, select **SQL**. The **Query workspace** page opens.
3. Browse to explore the following options in the **Query workspace** page.
   - **Engine**: To select an engine and view the associated catalogs.
   - **Search for loaded tables**: To search the tables and columns.
   - **Saved queries**: To view the saved queries.
   - **Worksheet**: To write SQL queries.

   The **Query workspace** page provides basic options to **undo**, **redo**, **cut**, **copy**, **paste**, **save**, **clear**, and **delete**.

   Format selection option is enabled only when an SQL statement in a query worksheet is selected. The **Format worksheet** option formats all the content in the worksheet. **Comment selection** is used to explain sections of SQL statements.

   The **Delete** option is enabled only after an SQL query is saved.
4. Select an engine from the **Engine** menu.
5. Select the catalog, schema, table, or column in which you want to run the query.
6. Click the overflow menu and select the required query.
   - For a catalog and schema, you can run the **Generate Path** query.
   - For a table, you can run the **Generate path**, **Generate SELECT**, **Generate ALTER**, and **Generate DROP** query.
   - For a column, you can run the **Generate path**, **Generate SELECT**, and **Generate DROP** query.

7. Select the **Catalog** and corresponding **Schema** from the drop-down on top of the worksheet to run queries for all tables within the schema without having to specify the path (`<catalog>.<schema>`) for every queries.
8. Click the **Save** icon to save the query. A **Save worksheet** confirmation dialog appears.
9. Enter worksheet name, click **Save**.
10. Click the **Run** button to run the query. Using **Run to cursor** or **Run from cursor**, you can run queries from or until your cursor position.
11. Select **Result set** or **Details** tab to view the results. Click the **Download** icon to export the result set and details to a CSV file.
12. Click **Saved queries** to view the saved queries.
13. Click **Explain** to view the logical or distributed plan of execution for a specified SQL query. See "About Visual Explain" on page 385 for more details.

About Visual Explain

The visual explain feature in IBM watsonx.data shows the execution plan for a specified SQL query. This feature also validates the SQL query. The output results can be visualized in different formats, which can be rendered into a graph or a flow chart. Data exchange happens between single or multiple nodes within a fragment. Each fragment has a set of data that is distributed between the nodes.

watsonx.data on Red Hat OpenShift

IBM watsonx.data developer edition

With this visual explain feature, you can run the query and show the output in a distributed environment. You can output the results in different formats. When queries are run, they make a scan through the database. The queries retrieve table metadata to fetch the correct output.

With this visual explain feature, you can visualize the query in a graphical representation. When a query is run in the SQL editor and selects the **Explain** option, watsonx.data uses an EXPLAIN SQL statement on the query to create a corresponding graph. This graph can be used to analyze, fix, and improve the efficiency of your queries saving time and cost.

To view the execution plans for a query that is run in watsonx.data SQL editor, complete the following steps.

1. In the **Query workspace** page, enter the query and click **Run on** to get the results.
2. Run the following command to obtain the table statistics:

   ```
   analyze <table name>
   ```

3. Click **Explain** on the screen to visualize the graphical representation of the query.

On the **Explain** window, click a stage to view the details of that stage. The details of each stage that is displayed are the estimate values for rows, CPU, memory, and network.

Query history

A query history audits all the current and past queries across engines.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

The query history page in IBM watsonx.data provides the following details that are related to the queries that are run:

- Query ID
- Query
- State
- Engine

- User
- Created

In the **Query history** page, you can search, refresh, filter, and customize the queries. You can select a **Query** from the page, view or copy the details of query statement, logical execution plan, and distributed execution plan. You can open the queries directly in a workspace, and also get the explain details of a query from the overflow menu of each query listed.

For more information about exporting and importing query history, see "Exporting and importing the query history - Red Hat OpenShift" on page 386

Exporting and importing the query history - Red Hat OpenShift

The Presto coordinator stores the query history in `system.runtime.queries` table. But `system.runtime.queries` table truncates when you restart the Presto, resulting in loss of query history. To mitigate this issue, we can export query history as a csv file and also import the query history from `system.runtime.queries` table to a non-system table.

watsonx.data on Red Hat OpenShift

About this task

To import and export the query history, you must install the Presto CLI and generate the keystore. For more information, see con-presto-serv.dita.

Procedure

1. To export the query history, run the following commands.

```
export PRESTO_PASSWORD=<your-cpd-password>
```

```
./presto --server https://<route_url> --catalog system --schema \
runtime --execute "select * from queries" --user admin --truststore-\
path <path-to-trustore-file> --truststore-password <truststore-pwd> --output-format
CSV_HEADER > \
history.csv --password
```

   Example:

```
export PRESTO_PASSWORD=<your-cpd-password>
```

```
./presto --server https://$route_url --catalog system --schema \
runtime --execute "select * from queries" --user admin --truststore-\
path presto.jks --truststore-password changeit --output-format CSV_HEADER > \
history.csv --password
```

   This command generates a CSV file, which contains exported query history.

2. To import the query history, complete the following steps.

   a) Create a schema in a catalog in which you have the write access.

```
create schema iceberg_data.query_history with (location='s3a://hive-bucket/query_history')
```

   Example:

```
./presto --server https://${route_url} --execute "create schema
iceberg_data.query_history with (location='s3a://iceberg-bucket/query_history')" \
--user admin --truststore-path presto.jks \
--truststore-password changeit --password
```

   b) Create a table in same catalog. This table must have same metadata as that of `system.runtime.queries` table. Use `CREATE TABLE AS SELECT` statement to create this table.

```
create table <non-system-table-name> as select * from system.runtime.queries where 1=0;
```

Example:

```
./presto --server https://${route_url} --execute "create table
iceberg_data.query_history.queries as select * from system.runtime.queries" \
--user admin \
--truststore-path presto.jks \
--truststore-password changeit \
--password
```

c) To import the query history into the table that you have created, run the following query periodically.

```
INSERT INTO <non-system-table-name> SELECT * FROM system.runtime.queries
WHERE query_id NOT IN (SELECT query_id FROM <non-system-table-name>);
```

Example:

```
./presto --server https://${route_url} \
--execute "insert into iceberg_data.query_history.queries select * from
system.runtime.queries where query_id NOT IN (SELECT query_id FROM
iceberg_data.query_history.queries)" \
--user admin --truststore-path presto.jks \
--truststore-password changeit --password
```

3. To retrieve query from both the tables, use following statement.

```
select * from <non-system-table-name> union select * from system.runtime.queries order by
created;
```

Example:

```
./presto --server https://${route_url} \
--execute " select * from iceberg_data.query_history.queries union select * from
system.runtime.queries order by created " \
--user admin \
--truststore-path presto.jks \
--truststore-password changeit --password
```

Exporting and importing the query history- Developer

The Presto (Java) coordinator stores the query history in `system.runtime.queries` table. T `system.runtime.queries` table truncates the query history when you restart the Presto, resulting in loss of query history. To mitigate this issue, we can export query history as a csv file and also import the query history from `system.runtime.queries` table to a non-system table.

watsonx.data Developer edition

About this task

To import and export the query history, you must install the Presto (Java) CLI and generate the keystore. For more information, see con-presto-serv.dita.

Procedure

1. To export the query history, run the following command.

```
export PRESTO_PASSWORD=<your-password>
```

Then run this command,

```
./presto --server https://<host:port> --catalog system --schema \
runtime --execute "select * from queries" --user ibmlhadmin --truststore-\
path <path-to-trustore-file> --truststore-password <truststore-pwd> --output-format
CSV_HEADER > \
history.csv --password
```

Example:

```
export PRESTO_PASSWORD=<your-password>
```

```
./presto --server https://shiftier1.fyre.ibm.com:8443 --catalog system --schema \
runtime --execute "select * from queries" --user ibmlhadmin --truststore-\
path presto.jks --truststore-password changeit --output-format CSV_HEADER > \
history.csv --password
```

This command generates a CSV file, which contains the exported query history.

2. Create a table in a catalog, which has a write access. The new table must have the same metadata as of `system.runtime.queries`.

   a) To create a table, run.

   ```
   create table <non-system-table-name> as select * from
   system.runtime.queries where 1=
   ```

   Example:

   ```
   ./presto --server https://shiftier1.fyre.ibm.com:8443 \
   --keystore-path /Users/user1/Desktop/certs/truststore.jks \
   --keystore-password mypassword –execute ” create table\
   hive_data.query_history.queries as select * from system.runtime.queries where 1=0” \
   --user ibmlhadmin --password
   ```

3. To import query history into the non-system table, that you created, you can use this command periodically:

```
INSERT INTO <non-system-table-name>\
SELECT * FROM system.runtime.queries\
WHERE query_id NOT IN (SELECT query_id FROM <non-system-table-\
name>)
```

Example:

```
./presto --server https://shiftier1.fyre.ibm.com:8443 \
--keystore-path /Users/user1/Desktop/certs/truststore.jks \
--keystore-password\mypassword –execute ” INSERT INTO hive_data.query_history.queries\
SELECT * FROM system.runtime.queries \
WHERE query_id NOT IN (SELECTquery_id FROM <non-system-table-name>);” \
--user ibmlhadmin –password
```

4. To retrieve query from both the tables, use following statement.

```
select * from <non-system-table-name> union select * \
from system.runtime.queries order by created
```

Example:

```
./presto --server https://shiftier1.fyre.ibm.com:8443 \
--keystore-path /Users/user1/Desktop/certs/truststore.jks \
--keystore-password mypassword –execute ” \
select * from hive_data.query_history.queries union select * from
system.runtime.queries order by created” \
--user ibmlhadmin –password
```

QHMM

Query History Monitoring and Management (QHMM) is a service that stores and manages the diagnostic data such as queries history and query event-related information of the Presto engine in the Minio bucket, `wxd-system`.

watsonx.data on Red Hat OpenShift

For more information about QHMM, see the following:

QHMM overview

Query History Monitoring and Management (QHMM) is a service that stores and manages the diagnostic data such as, queries history and query event-related information of the Presto(Java) and Presto(C++) engine in a storage bucket. You can retrieve the stored history files for analysis, debugging and monitoring purpose.

QHMM primarily aims to address the issue of data persistence for serviceability data. When the engine restarts or go offline, there are chances of losing valuable diagnostic data they generate. QHMM resolves the issue by storing such data in object storage solutions like COS (Cloud Object Storage) or Minio buckets. The data is organized in a structured folder hierarchy, making it easily accessible for users to retrieve and analyze.

QHMM allows to retrieve the following diagnostic data:

- Logs generated for the following query events in Presto:
  - Query created event - event logged when a query is initiated.
  - Split completed event - split correspond to an individual task in a query execution. An event is logged when a split or a task is completed.
  - Query completed event - event logged when a query execution is completed.
  - Query Optimiser event - event logged when the query optimizer is enabled.
- Histories of the queries executed by Presto in the form of a `json` file.
- Query history table created where user can execute a query to view the histories of the queries executed by Presto.
- Heap dump by using API.

watsonx.data on Red Hat OpenShift

Data organization

QHMM maintains a structured folder hierarchy within the chosen object storage solution to store data efficiently. The structure is as follows:

`<base_path>/` `<wxdInstanceID>/` `<engine or service type>/` `<engine or service ID>/` `<recordtype>/` `dt=<dd-MM-yyyy>` `<records>`

- `<base_path>` : Default `base_path` is `qhmm` and you can change the path from the **QHMM configuration** page.
- `<wxdInstanceID>`: A unique identifier for the instance.
- `<engine or service type>`: The type of engine sending the data.
- `<engine or service ID>`: A unique identifier for the engine.
- `date<dd-MM-yyyy>`: The date on which the data is recorded.
- `<recordtype>`: The type of diagnostic data (e.g., query history, query events, dumps).
- `<records>`: The actual data records stored.

To store query history details, one more folder is added: `<base_path>/` `<wxdInstanceID>/` `<engine or service type>/` `<engine or service ID>/` `<recordtype>/` `dt=<dd-MM-yyyy>` `<user>` `<records>`

- `<user>`: The username of the person who executed the query.

Configuring Query monitoring

This topic guides you through the procedure to enable or disable Query History Monitoring and Management (QHMM) and specify the storage to be used for storing the query data.

You can enable or disable the QHMM service for your watsonx.data instance. If you enable the QHMM service, you must specify the storage to be used for storing the query data. watsonx.data allows using the default storage, `wxd-system` to store the QHMM data or register your own storage (BYOB). To use BYOB, register your own storage in watsonx.data and configure it to use as QHMM storage. For more information, see Performing Query monitoring.

Based on the storage selected, diagnostic tables are automatically created to store data.

You can choose the QHMM storage (default QHMM storage or your own storage) from watsonx.data console page (see Query monitoring)

You can retrieve the history files to analyze, debug, or monitor the queries. from the Query workspace. For more information, see Performing Query monitoring.

watsonx.data on Red Hat OpenShift

Procedure

1. Log in to the watsonx.data console.
2. From the navigation menu, select **Configurations**.
3. Click **Query monitoring**. The **Query monitoring** page opens.
4. You can view or edit the QHMM configuration details. The following details are available:
   - Status of QHMM - Enabled or disabled. Use the toggle switch to enable or disable QHMM. You can also disable QHMM by using commands. For more information, see Disabling_QHMM.
   - Storage that is configured to store QHMM data
   - The subpath in the storage where QHMM data is available
5. To edit the configuration details, click **Edit** and make the required changes. You can enable or disable, change the bucket and subpath.
6. Click **Save** after making the changes.

Disabling Query History Monitoring and Management (QHMM)

Query History Monitoring and Management (QHMM) is a service that stores and manages the diagnostic data such as query history and query event-related information of the Presto engine in the MinIO bucket `wxd-system`.
You can disable the QHMM service in watsonx.data from the watsonx.data console page or by using command line. This topic guides you through the steps to disable QHMM by using command line. For more information about disabling QHMM from watsonx.data console page, see Query monitoring.

watsonx.data on Red Hat OpenShift

Procedure

1. Log in to the Red Hat OpenShift cluster by using one of the following options:
   a) Run the following command to log in to the cluster by providing a username and password:

   ```
   ibm-lakehouse-manage login-to-ocp \
   --user=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

   b) Run the following command to log in to the cluster by providing a token:

   ```
   ibm-lakehouse-manage login-to-ocp \
   --server=${OCP_URL} \
   --token=${OCP_TOKEN}
   ```

2. Set the project by using the following command:

   ```
   oc project <PROJECT_CPD_INST_OPERANDS>
   ```

3. Run the following command to list the config maps related to Presto.

   ```
   oc get cm |grep presto-config-cm
   ```

4. Run the following command to disable QHMM. Replace the *<engine-id>* with the ID of the engine in which you want to disable the QHMM.

   ```
   oc edit ibm-lh-lakehouse-<engine-id>-presto-config-cm
   ```

5. Ensure that the value of `ENABLE_QHMM` flag is changed to `false` and save the configuration map.
6. Delete the Presto coordinator or worker pod to apply the updated environment configuration.

   ```
   oc delete pod <presto Coordinator/worker pod name>
   ```

Enabling file pruning

Enable the file pruning functionality in QHMM to manage the storage capacity. You can configure the maximum size and threshold percentage for the QHMM storage bucket. When the threshold is met during file upload or when a cleanup scheduler runs (default every 24 hours), older data is deleted.

watsonx.data on Red Hat OpenShift

Procedure

1. Log in to the Red Hat OpenShift cluster by using one of the following options:
   a) Run the following command to log in to the cluster by providing a username and password:

   ```
   ibm-lakehouse-manage login-to-ocp \
   --user=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

   b) Run the following command to log in to the cluster by providing a token:

   ```
   ibm-lakehouse-manage login-to-ocp \
   --server=${OCP_URL} \
   --token=${OCP_TOKEN}
   ```

2. Set the project by using the following command:

   ```
   oc project <PROJECT_CPD_INST_OPERANDS>
   ```

3. Run the following command to list the config maps related to Presto.

   ```
   oc get cm |grep presto-config-cm
   ```

4. Configure the following environment variables in the `presto-config-cm` config map file to enable file pruning for QHMM.
   - QHMM_BUCKET_MAX_USAGE_LIMIT: Maximum capacity of the storage in MB (default:10240 MB).
   - QHMM_RECORD_PRUNE_FREQUENCY_HRS: Frequency in hours at which the scheduler runs to prune data (default: 24 hours).
   - QHMM_RECORD_EXPIRY_DAYS: Record expiry time in days for deleting records from COS (default: 30 days).
   - QHMM_RECORD_PRUNE_THRESHOLD: Threshold at which QHMM triggers pruning or issues a warning to the user when the capacity is reached in percentage (default:80%).
   - ENABLE_QHMM_PRUNE: If QHMM pruning is enabled (default: false) or not.
5. Run the following command to enable file pruning for QHMM. Replace the *<engine-id>* with the ID of the engine in which you want to disable the QHMM.

   ```
   oc edit ibm-lh-lakehouse-<engine-id>-presto-config-cm
   ```

6. Ensure that the value of `ENABLE_QHMM` flag is changed to `false` and save the configuration map.
7. Delete the Presto coordinator or worker pod to apply the updated environment configuration.

   ```
   oc delete pod <presto Coordinator/worker pod name>
   ```

Analyzing diagnostic data

This topic guides you through the steps to retrieve and analyse the Query History Monitoring and Management (QHMM) data. You can analyse the QHMM data either from the watsonx.data user interface or manually by using commands.

watsonx.data on Red Hat OpenShift

For more information about QHMM, see the following:

Managing diagnostic data from user interface

The Query history monitoring and management page in IBM watsonx.data provides information for fetching the history data and analyzing the queries that are run. You can retrieve the history files to analyze, debug or monitor the queries. from the **Query workspace** page.

watsonx.data on Red Hat OpenShift

Procedure

1. Add storage to store the QHMM data.

   Add a storage to store the QHMM data. Supported storage options are, IBM Cloud Object Storage, Amazon S3, or MinIO. To create the storage, see Adding a storage-catalog pair..

   Important: QHMM supports only the Apache Hive catalog.
2. Associate the storage with a query engine (Presto (Java) or Presto (C++)). For details, see Associating a catalog with an engine.
3. Configure Query monitoring. For details, see "Configuring Query monitoring" on page 389.
   a. Enable QHMM. For details, see "Configuring Query monitoring" on page 389.
   b. Select the storage bucket to store QHMM data. To do that, see "Configuring Query monitoring" on page 389. From the **Infrastructure manager** , select the storage bucket (for QHMM data) . An information message is displayed on the top of the page "This bucket cannot be deleted or disabled because it is being used for storing query history data."
   c. Click **Save** to save the details. The queries history and query event-related information of the Presto engine is available in the storage bucket.
4. Retrieve query information.
   a. From the navigation menu, select **Query workspace**.
   b. Select the engine from **Engine** list and identify the catalog that you created to store the QHMM data.
   c. You can view the queries history and query event-related information of the Presto engine inside the tables within the catalog. Run queries to analyse the data.
   d. The following tables are available by default:
      - query_event_raw : Includes the raw data of query events in JSON format. To view the entire event data, use the following query:
        ```
        SELECT*FROM<catalog>.<schema>.query_event_raw;
        ```
      - query_history : Includes the query History data. To view the entire Query History data, use the following query:
        ```
        SELECT*FROM<catalog>.<schema>.query_history;
        ```
      - table_stats_information_memory : Includes the memory related information. To view the memory related information, use the following query:
        ```
        SELECT*from<catalog>.<schema>.table_stats_information_memory;
        ```

Managing diagnostic data by manual method

The Query history monitoring and management page in IBM watsonx.data provides information for fetching the history data and analyzing the queries that are run. You can retrieve the stored history files for analysis, debugging and monitoring purpose.

watsonx.data on Red Hat OpenShift

Before you begin

Ensure that you have the following information:

- `<instance-id>`: unique identifier of the watsonx.data instance. Save the instance ID for reference.
- `<engine>` : the engine used for data processing. Here, Presto.
- `<catalog>`: the catalog specific to QHMM. Save the catalog name for reference. Here we use, `wxd_system_data`.
- `<bucket>`: the MinIO bucket for storing the diagnostic data. Save the bucket name for reference. Here we use, `wxd-system.`
- Go to the **Query workspace** page and do the following:
  1. Run the following command to create a schema to organize the diagnostic data for QHMM.

     ```
     CREATE SCHEMA IF NOT EXISTS <catalog>.diag WITH (location = 's3a://wxd-system/diag/');
     ```

     Here, the `<catalog>` name is `wxd_system_data`.
  2. Run the following command to create a table to store the query event data.

     ```
     CREATE TABLE IF NOT EXISTS <catalog>.diag.query_event_raw (
       record VARCHAR,
       dt VARCHAR
     )
     WITH (
       external_location = 's3a://<bucket>/qhmm/<instance-id>/<engine>/<engine-id>/QueryEvent/',
       format = 'textfile',
       partitioned_by = ARRAY['dt']
     );
     ```

     Here, the `<catalog>` name is `wxd_system_data`.
  3. Run the following command to create a table to store the query history data in JSON format.

     ```
     CREATE TABLE IF NOT EXISTS <catalog>.diag.query_history(
       query_id VARCHAR,
       query VARCHAR,
       state VARCHAR,
       source VARCHAR,
       created VARCHAR,
       started VARCHAR,
       "end" VARCHAR,
       dt VARCHAR,
       user VARCHAR)
     WITH (
       external_location = 's3a://<bucket>/qhmm/<instance-id>/<engine>/<engine-id>/
     QueryHistory/',
       format = 'JSON',
       partitioned_by = ARRAY['dt','user']
     );
     ```

     Here, the `<catalog>` name is `wxd_system_data`.

About this task

The system stores the data in watsonx.data MinIO buckets. Here, **wxd-system**.

Note: The system removes the data after every 7 days, and the maximum storage capacity is 1 GB (exceeding limit trims the data to fit into 1 GB size).

Procedure

1. Log in to IBM watsonx.data console.
2. Go to the **Objects** tab in the minIO bucket page. The query table displays the **QueryEvent** and **QueryHistory** folders with data files in JSON format.
3. **Synchronizing with the QHMM data**

   a. From the navigation menu, go to the **Query workspace** page.

   b. Run the following command to fetch the current QHMM data into the watsonx.data table.

   ```
   USE <catalog>.diag;
   CALL system.sync_partition_metadata('diag', 'query_event_raw', 'FULL');
   CALL system.sync_partition_metadata('diag', 'query_history', 'FULL');
   ```

4. To view data that is fetched from QHMM, run the following command:

```
CREATE VIEW <catalog>.diag.query_event_view AS
SELECT
    json_extract_scalar (record, '$.clusterName') cluster_name,
    json_extract_scalar (record, '$.queryCompletedEvent.metadata.queryId') query_id,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.metadata.queryState'
    ) query_state,
    CAST(
        json_extract (record, '$.queryCompletedEvent.ioMetadata.inputs') AS JSON
    ) query_inputs,
    from_unixtime (
        CAST(
            json_extract_scalar (
                record,
                '$.queryCompletedEvent.createTime.epochSecond'
            ) AS bigint
        )
    ) create_time,
    from_unixtime (
        CAST(
            json_extract_scalar (
                record,
                '$.queryCompletedEvent.executionStartTime.epochSecond'
            ) AS bigint
        )
    ) execution_start_time,
    from_unixtime (
        CAST(
            json_extract_scalar (
                record,
                '$.queryCompletedEvent.endTime.epochSecond'
            ) AS bigint
        )
    ) end_time,
    json_extract_scalar (record, '$.cpuTimeMillis') cpuTimeMillis,
    json_extract_scalar (record, '$.wallTimeMillis') wallTimeMillis,
    json_extract_scalar (record, '$.queuedTimeMillis') queuedTimeMillis,
    json_extract_scalar (record, '$.analysisTimeMillis') analysisTimeMillis,
    (
        CAST(
            json_extract (
                record,
                '$.queryCompletedEvent.statistics.planningTime.seconds'
            ) AS BIGINT
        ) * 1000 + CAST(
            json_extract (
                record,
                '$.queryCompletedEvent.statistics.planningTime.nano'
            ) AS BIGINT
        ) / 1000000
    ) planningTimeMillis,
    json_extract_scalar (record, '$.queryCompletedEvent.context.user') user,
    CAST(
        "json_extract" (record, '$.queryCompletedEvent.stageStatistics') AS ARRAY (ROW
(gcStatistics MAP (VARCHAR, INTEGER)))
    ) gcStatistics,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.totalRows'
    ) total_rows,
```

```
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.outputRows'
    ) output_rows,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.writtenOutputRows'
    ) written_output_rows,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.totalBytes'
    ) total_bytes,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.outputBytes'
    ) output_bytes,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.cumulativeMemory'
    ) cumulative_memory,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.completedSplits'
    ) completed_splits,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.peakRunningTasks'
    ) peak_running_tasks,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.peakUserMemoryBytes'
    ) peak_user_memory_bytes,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.peakTotalNonRevocableMemoryBytes'
    ) peak_total_non_revocable_memory_bytes,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.peakTaskUserMemory'
    ) peak_task_user_memory,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.peakTaskTotalMemory'
    ) peak_task_total_memory,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.statistics.peakNodeTotalMemory'
    ) peak_node_total_memory,
    json_extract_scalar (record, '$.queryCompletedEvent.context.source') source,
    json_extract_scalar (record, '$.queryCompletedEvent.context.catalog') catalog,
    json_extract_scalar (record, '$.queryCompletedEvent.context.schema') schema,
    CAST(
        json_extract (
            record,
            '$.queryCompletedEvent.context.resourceGroupId'
        ) AS ARRAY (VARCHAR)
    ) resource_group_id,
    CAST(
        json_extract (
            record,
            '$.queryCompletedEvent.context.sessionProperties'
        ) AS MAP (VARCHAR, VARCHAR)
    ) session_properties,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.context.serverVersion'
    ) server_version,
    CAST(
        "json_extract" (
            record,
            '$.queryCompletedEvent.failureInfo.errorCode'
        ) AS MAP (VARCHAR, VARCHAR)
    ) error_code,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.failureInfo.failureType'
    ) failure_type,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.failureInfo.failureMessage'
    ) failure_message,
    json_extract_scalar (
        record,
        '$.queryCompletedEvent.failureInfo.failuresJson'
    ) failure_json,
    json_extract_scalar (record, '$.plan') plan,
    json_extract_scalar (record, '$.queryCompletedEvent.metadata.query') query,
    regexp_like("json_extract_scalar"(record, '$.plan'), 'InnerJoin|RightJoin|SemiJoin|
CrossJoin|FullJoin') isAJoinQuery
FROM
    <catalog>.diag.query_event_raw
```

5. To can analyze the statistics and memory usage information and view the garbage code details, run the following command:

```
CREATE VIEW <catalog>.diag.fullGC_TaskDetails AS
    SELECT
        query_id,
        create_time,
        execution_start_time,
        end_time,
        SUM(i."gcStatistics"['tasks']) as total_tasks,
        SUM(i."gcStatistics"['fullGcTasks']) total_full_gc_tasks,
        SUM(i."gcStatistics"['maxFullGcSec']) max_full_gc_sec,
        SUM(i."gcStatistics"['totalFullGcSec']) total_full_gc_sec,
        b.gcStatistics,
        query,
        isAJoinQuery
    FROM
      <catalog>.diag.query_event_view b
      CROSS JOIN UNNEST(b.gcStatistics) WITH ORDINALITY AS i ("gcStatistics", n)
      GROUP BY query_id, create_time, execution_start_time, end_time, b.gcStatistics, query,
isAJoinQuery;
```

6. To view query statistics and memory usage, run the following command:

```
CREATE VIEW <catalog>.diag.table_stats_information_memory AS (
SELECT catalogname, schema, "table", isAJoinQuery, row_count_stats, total_size_stats,
execution_start_time, peak_node_total_memory,
  CASE
    WHEN row_count_stats != 'NaN' OR total_size_stats != 'NaN' THEN 'YES'
    ELSE 'NO'
    END AS is_table_stats_available
    FROM (
      SELECT json_extract_scalar(i.statistics['totalSize'], '$.value') total_size_stats,
             json_extract_scalar(i.statistics['rowCount'], '$.value') row_count_stats,
             i.n, i.schema, i."table", i.catalogName, b.execution_start_time,
b.peak_node_total_memory, b.isAJoinQuery
      FROM <catalog>.diag.query_event_view b
    cross join
      UNNEST(CAST(b.query_inputs AS Array(ROW(catalogName VARCHAR, schema VARCHAR, "table"
VARCHAR, statistics MAP(VARCHAR, JSON)))))
      WITH ORDINALITY AS i (catalogName, schema, "table", statistics, n))
  );
```

7. To run the SQL commands sequentially in your SQL environment to analyze the data that is fetched, run the following command:

```
SELECT * FROM  <catalog>.diag.query_event_raw limit 10;
-- To check partitions
SELECT * FROM  <catalog>.diag."query_event_raw$partitions";

SELECT * from <catalog>.diag.query_event_view where query_state='FINISHED' limit 10;
SELECT * from <catalog>.diag.fullGC_TaskDetails limit 10;
SELECT * from <catalog>.diag.table_stats_information_memory limit 10;
```

Query Optimizer

Installing Query Optimizer in IBM watsonx.data version 2.0.0

You can install **Query Optimizer** in IBM watsonx.data version 2.0.0 to improve the performance of queries that are processed by Presto (C++) engine.

Note: For a new user to directly install **Query Optimizer** in IBM watsonx.data version 2.0.1 and later, see "Activating Query Optimizer in IBM watsonx.data version 2.0.1 and later" on page 398.

Before you begin

1. Install watsonx.data, see Installing watsonx.data.
2. If you are using a private container registry, mirror images for Query Optimizer dependencies by using `export components=wxd_query_optimizer`. For more information, see Mirroring IBM Cloud Pak for Data images to a private container registry.
3. Create buckets and associate catalogs, see Adding a bucket-catalog pair.
4. Create schemas and tables under the registered catalog, see Creating schemas and Creating tables.
5. If you are using the IBM file gold storage class for the **Query Optimizer** installation, run the steps in Setting up IBM Cloud File Storage (ibmc-file-gold-gid storage class).

About this task

This topic provides step by step instruction to install **Query Optimizer** in watsonx.data.

Procedure

1. Run the following command to install **Query Optimizer** in the instance namespace by using `cpd-cli` command:

   ```
   cpd-cli manage apply-olm --components=wxd_query_optimizer --release=${VERSION} --
   cpd_operator_ns=${PROJECT_CPD_INST_OPERATORS} --license_acceptance=true
   ```

2. Update the global settings on the cluster to configure node settings for the operator:

   Note: Updating the global settings is not required if it is already set for other services.

   a. Pause machine config pool (*mcp*) updates for changing `CRI-O pids_limit` with a **`KubeletConfig`** by running the following command:

   ```
   oc patch --type=merge --patch='{"spec":{"paused":true}}' machineconfigpool/master

   oc patch --type=merge --patch='{"spec":{"paused":true}}' machineconfigpool/worker
   ```

   b. Update the load balancer timeout. See Changing load balancer timeout settings.

   c. Update process IDs limits. See Changing the process IDs limit.

   d. Update kernel parameter settings. See Changing kernel parameter settings.

   e. Run the following command to resume *mcp* updates:

   ```
   oc patch --type=merge --patch='{"spec":{"paused":false}}' machineconfigpool/master

   oc patch --type=merge --patch='{"spec":{"paused":false}}' machineconfigpool/worker
   ```

   f. Run the following command to check the status of *mcp*:

   ```
   watch "oc get mcp"
   ```

   Note: You must wait until all the nodes show status as `UPDATED: True`

3. Run the following command to create a configmap to use elevated privileges in the instance namespace:

   ```
   oc apply -f - <<EOF
   apiVersion: v1
   data:
   ```

```
  DB2U_RUN_WITH_LIMITED_PRIVS: "false"
kind: ConfigMap
metadata:
  name: db2u-product-cm
  namespace: ${PROJECT_CPD_INST_OPERANDS}
EOF
```

4. Run the following command to list all engines:

   ```
   oc get wxdengine -n ${PROJECT_CPD_INST_OPERANDS} -o custom-
   columns='Name:metadata.name,Display Name:spec.engineDisplayName'
   ```

5. Run the following command to select and export the engine that you want to patch:

   ```
   export ENGINE= <engine name>
   ```

6. Patch the engine to enable the **Query Optimizer** feature by using `wxd_query_optimizer` variable.

   By default `wxd_query_optimizer` is set to `false`. To enable **Query Optimizer**, set `wxd_query_optimizer= true`. To optimize all the queries, set `enable_wxd_query_optimizer: true`.

   ```
   export BLOCK_STORAGE_CLASS=<BLOCK_STORAGE_CLASS>
   export FILE_STORAGE_CLASS=<FILE_STORAGE_CLASS>

     oc patch wxdengine $ENGINE \
     --type=merge \
     -n ${PROJECT_CPD_INST_OPERANDS} \
     -p '{ "spec": {
             "wxd_query_optimizer": true,
             "enable_wxd_query_optimizer": true,
             "wxdQueryOptimizerBlockStorageClass": "'$BLOCK_STORAGE_CLASS'",
             "wxdQueryOptimizerFileStorageClass": "'$FILE_STORAGE_CLASS'"
         } }'
   ```

   Where, `BLOCK_STORAGE_CLASS` should be a valid RWO (ReadWriteOnce) storage class, and `FILE_STORAGE_CLASS` should be a valid RWX (ReadWriteMany) storage class.

7. Run the following command to check the status of Query Optimizer setup:

   ```
   watch "oc get wxdengine -n $
   {PROJECT_CPD_INST_OPERANDS} -o custom-columns='Name:metadata.name,Display
   Name:spec.engineDisplayName,RECONCILE:status.engineStatus,STATUS:status.middleEndStatus'"
   ```

   Note: Wait for the engine to finish reloading the Query Optimizer settings and show that the reconcile is "Completed".

Activating Query Optimizer in IBM watsonx.data version 2.0.1 and later

You can install and activate **Query Optimizer** in IBM watsonx.data version 2.0.1 and later by following the instructions in this topic.

If you are upgrading from an existing IBM watsonx.data version 2.0.0 to 2.0.1 or later , then go to step 4 to activate **Query Optimizer**.

Before you begin

1. Install watsonx.data, see Installing watsonx.data.
2. If you are using a private container registry, mirror images for Query Optimizer dependencies by using `export components=wxd_query_optimizer`. For more information, see Mirroring IBM Cloud Pak for Data images to a private container registry.
3. Create buckets and associate catalogs, see Adding a bucket-catalog pair.
4. Create schemas and tables under the registered catalog, see Creating schemas and Creating tables.
5. Run the steps in Setting up IBM Cloud File Storage (ibmc-file-gold-gid storage class) by Db2 warehouse if you are using the IBM file gold storage class for the **Query Optimizer** install.

About this task

Follow the steps to activate **`Query Optimizer`** in watsonx.data.

Procedure

1. Run the following command to install **`Query Optimizer`** in the instance namespace by using `cpd-cli` command:

   ```
   cpd-cli manage apply-olm --components=wxd_query_optimizer --release=${VERSION} --
   cpd_operator_ns=${PROJECT_CPD_INST_OPERATORS} --license_acceptance=true
   ```

2. Update the global settings on the cluster to configure node settings for the operator:

   Note: Updating the global settings is not required if it is already set for other services.

   a. Pause machine config pool (*mcp*) updates for changing `CRI-O pids_limit` with a **`KubeletConfig`** by running the following command:

      ```
      oc patch --type=merge --patch='{"spec":{"paused":true}}' machineconfigpool/master

      oc patch --type=merge --patch='{"spec":{"paused":true}}' machineconfigpool/worker
      ```

   b. Update the load balancer timeout. See Changing load balancer timeout settings.

   c. Update process IDs limits. See Changing the process IDs limit.

   d. Update kernel parameter settings. See Changing kernel parameter settings.

   e. Run the following command to resume *mcp* updates:

      ```
      oc patch --type=merge --patch='{"spec":{"paused":false}}' machineconfigpool/master

      oc patch --type=merge --patch='{"spec":{"paused":false}}' machineconfigpool/worker
      ```

   f. Run the following command to check the status of *mcp*:

      ```
      watch "oc get mcp"
      ```

      Note: You must wait until all the nodes show status as `UPDATED: True`

3. Run the following command to create a configmap to use elevated privileges in the instance namespace:

   ```
   oc apply -f - <<EOF
   apiVersion: v1
   data:
     DB2U_RUN_WITH_LIMITED_PRIVS: "false"
   kind: ConfigMap
   metadata:
     name: db2u-product-cm
     namespace: ${PROJECT_CPD_INST_OPERANDS}
   EOF
   ```

4. Log in to watsonx.data console.
5. From the navigation menu, select **Configurations** and click the **Query Optimizer Manager** tile.
6. Click **Activate** and confirm the activation and restarting of the engines in the **Activate query optimizer** window.

   Note: **`Query Optimizer`** takes approximately 20 minutes to deploy and sync over metadata for all Hive catalogs and schemas. This time may vary based on the metadata size to be synced.

   Note: Verify that all expected tables have been synced. If tables are found missing during the automatic syncing process, you can manually sync the tables. See Manually syncing Query Optimizer.

   Note: You can click **Cancel activation** to cancel the deployment of **Query Optimizer Manager** during the deployment.

Manually syncing Query Optimizer with watsonx.data metastore

Before you begin

To sync tables from watsonx.data, the following items are required:

1. Identify the list of Hive tables in watsonx.data that you require for **`Query Optimizer`**.
2. Identify columns as primary and foreign keys in the Hive tables.
3. ANALYZE Hive tables in Presto (C++) to generate Hive statistics.

About this task

To provide optimized queries, **`Query Optimizer`** pulls data about table definitions and hive statistics to synchronize with Hive metastore in watsonx.data. You can select the specific Hive table that must be available for **`Query Optimizer`**. It is recommended to generate Hive statistics and label columns for primary and foreign keys to get the best results.

Activating **`Query Optimizer`** automatically synchronizes metadata for catalogs that are connected to Presto (C++) engines. However, you will need to run the following steps if:

- Metadata for inaccessible or corrupted catalogs or schemas during deployment are missing.
- Significant changes are made to a table.
- New tables are introduced after the initial sync operation.
- An intermittent issue is preventing tables from being synced during the automatic syncing process upon activation.

Procedure

1. Log into watsonx.data.
2. Go to **Query workspace**.
3. Run the ANALYZE command from the watsonx.data web console for the tables that you want to sync to generate the statistics (Statistics is the number of rows, column name, data_size, row count, and more).

   ```
   ANALYZE catalog_name.schema_name.table_name ;
   ```

4. Run the following command to register the properties of **`Query Optimizer`**:

   ```
   ExecuteWxdQueryOptimizer 'CALL SYSHADOOP.REGISTER_EXT_METASTORE('hive_data','type=watsonx-
   data,uri=thrift://optpassthru1.fyre.ibm.com:9083,use.SSL=true,ssl.cert=/tmp/
   hms.pem,auth.mode=PLAIN,auth.plain.credentials=lh-default:e085e323e65407efe5acbab7', ?, ?)';
   ```

   - watsonx.data catalog name - as shown on the **Infrastructure Manager** page (case sensitive).
   - HMS thrift uri - As obtained from the **Infrastructure Manager** page (click on the catalog).
   - HMS credentials - Must be created on the watsonx.data side. See Connecting to watsonx.data on OpenShift
   - HMS certificate - this must be provided as a file on the db2u container.
     - Identify the db2u **`Query Optimizer`** head pod.
     - Run the following command to generate certificate by substituting the values for <OPT_POD> and <HMS_ENDPOINT>.

       ```
       oc exec -it $OPT_POD -c db2u -- bash -c "echo QUIT | openssl s_client -showcerts
       -connect $HMS_ENDPOINT | awk '/-----BEGIN CERTIFICATE-----/ {p=1}; p; /-----END
       CERTIFICATE-----/ {p=0}' > /tmp/hms.pem"
       ```

5. Run the following command to sync for each schema in the catalog:

   ```
   ExecuteWxdQueryOptimizer 'CALL SYSHADOOP.EXT_METASTORE_SYNC('<CATALOG_NAME>',
   '<Schema_name>', '.*', 'SKIP', 'CONTINUE', 'OAAS')';
   ```

   `<Schema_name>` is the name of each schema to synchronize.

SKIP: Indicates that objects that are already defined should be skipped.

CONTINUE: The error is logged, but processing continues if multiple tables are to be imported.

6. Run the ALTER table command to define the constraints:

```
-- NOT NULL
ExecuteWxdQueryOptimizer 'ALTER TABLE "catalog_name".schema_name.Employees ALTER COLUMN
FirstName SET NOT NULL ALTER COLUMN LasttName SET NOT NULL ALTER COLUMN Salary SET NOT NULL
ALTER COLUMN EmployeeID SET NOT NULL';
ExecuteWxdQueryOptimizer 'ALTER TABLE "catalog_name".schema_name.Departments ALTER COLUMN
DepartmentName SET NOT NULL ALTER COLUMN DepartmentID SET NOT NULL ';
ExecuteWxdQueryOptimizer 'ALTER TABLE "catalog_name".schema_name.EmployeeDepartmentMapping
ALTER COLUMN MappingID SET NOT NULL ';

-- PRIMARY KEY
ExecuteWxdQueryOptimizer 'ALTER TABLE "catalog_name".schema_name.Employees ADD PRIMARY KEY
(EmployeeID) NOT ENFORCED';
ExecuteWxdQueryOptimizer 'ALTER TABLE "catalog_name".schema_name.Departments ADD PRIMARY KEY
(DepartmentID) NOT ENFORCED';
ExecuteWxdQueryOptimizer 'ALTER TABLE "catalog_name".schema_name.EmployeeDepartmentMapping
ADD PRIMARY KEY (MappingID) NOT ENFORCED';

-- FOREIGN KEY
ExecuteWxdQueryOptimizer 'ALTER TABLE "catalog_name".schema_name.EmployeeDepartmentMapping
ADD FOREIGN KEY (EmployeeID) REFERENCES "catalog_name".schema_name.Employees(EmployeeID) NOT
ENFORCED';
ExecuteWxdQueryOptimizer 'ALTER TABLE "catalog_name".schema_name.EmployeeDepartmentMapping
ADD FOREIGN KEY (DepartmentID) REFERENCES
"catalog_name".schema_name.Departments(DepartmentID) NOT ENFORCED';
```

Updating Query Optimizer configuration

In Query Optimizer, values for the following Db2 parameters are based on the configuration of the Presto (Java) and Presto (C++) engines.

- SHEAPTHRES in dbm cfg
- OPT_SORTHEAP in db cfg
- OPT_BUFFPAGE in db cfg

If the following engine properties are changed or if there is a switch to another engine, then the Db2 parameters must be updated.

Presto single node

```
presto_singlenode_query_max_memory from wxdengine
presto_singlenode_query_max_total_memory_per_node from wxdengine
presto_singlenode_jvm_Xmx from wxdengine
presto_singlenode_resources_requests_memory from wxdengine
```

Presto multinode

```
query.max-memory in presto coordinator from wxd api
query.max-memory in presto worker from wxd api
query.max-memory-per-node in presto worker from wxd api
jvm.xmx in presto worker from wxd api
query.max-memory-per-node in presto coordinator from wxd api
jvm.xmx in presto coordinator from wxd api
presto_coordinator_query_max_memory from wxdengine
presto_worker_query_max_memory from wxdengine
presto_worker_query_max_total_memory_per_node from wxdengine
presto_worker_jvm_Xmx from wxdengine
presto_worker_resources_requests_memory from wxdengine
```

Presto (C++)

```
presto_coordinator_query_max_memory from wxdengine
prestissimo_worker_query_max_memory_per_node from wxdengine
prestissimo_worker_system_memory_gb
prestissimo_worker_resources_limits_memory
presto_coordinator_query_max_memory_per_node from wxdengine
presto_coordinator_jvm_Xmx from wxdengine
presto_coordinator_resources_requests_memory from wxdengine
```

Before you begin

- Ensure Query Optimizer is installed. For details, see "Installing Query Optimizer in IBM watsonx.data version 2.0.0" on page 397.
- You have admin access to the cluster and have the privilege to run the update command.

Note: watsonx.data APIs cannot support updates in **`wxdengine`**.

About this task

In this task, we will update the Query Optimizer configuration.

Procedure

1. Set up the `PROJECT_CPD_INSTANCE` environment variable pointing to the namespace where watsonx.data is installed.

   ```
   export PROJECT_CPD_INSTANCE=<wxd_namespace>
   ```

2. List the engines from the namespace.

   ```
   oc get wxdengine -n ${PROJECT_CPD_INSTANCE}
   ```

3. Select one engine from the list and trigger the update.

   ```
   oc patch wxdengine <wxdengine name>   --type=merge   -n ${PROJECT_CPD_INSTANCE}   -p
   '{ "spec": {
             "update_optimizerplus": "true"
         } }'
   ```

Running queries from the Presto (C++) CLI and inspecting watsonx.data logs

IBM watsonx.data allows running SQL queries from Presto (C++) CLI with or without using **Query Optimizer**. You can also inspect the log files of the queries that are run.

Before you begin

1. Run the following command inside `ibm-lh-client/bin` to add an engine:

   ```
   ./manage-engines --op=<add|remove|list|details> --name=<enginename> --host=<hostname> --
   port=<portnumber> --username=<username|ibmlhapikey> --password=<password|apikey>
   ```

   - **`add`** is to add engine
   - **`remove`** is to remove engine
   - **`list`** is to list all engines
   - **`details`** is to get details of engine

2. Run the following command to register watsonx.data instance:

   ```
   /ibm-lh config <config> name <engname> --host <LH instance URL> --port <portnumber>
   ```

   `<config>` is the configuration operation that can be executed. Available config operations are as follows:

   - **`select`** to select a watsonx.data instance
   - **`ls`** to list all your watsonx.data instances
   - **`add_saas`** that adds new SaaS watsonx.data instance
   - **`add_dev`** that adds new developer edition watsonx.data instance
   - **`add_cpd`** that adds new CPD edition watsonx.data instance
   - **`rm`** that removes an existing watsonx.data instance

3. Add a Presto (C++) engine connection certificate to `ibm-lh-client`:

   ```
   ./cert-mgmt --op=<add|remove>  --host=<host> --port=<port>
   ```

   - **`--op=add`** is for importing a CA certificate
   - **`--op=remove`** is for removing a previously imported certificate
   - **`-host=`** and **`--port=`** identifies the location that you need to trust

Note: Optimizer is disabled if:

- global environment variable is **`false`** and session is disabled.
- global environment variable is **`true`** and session is disabled.
- global environment variable is **`false`** and no session parameter is passed.

Optimizer is enabled if:

- global environment variable is **`true`** and session is enabled.
- global environment variable is **`false`** and session is enabled.
- global environment variable is **`true`** and no session parameter is passed.

About this task

You can select the option of running Presto (C++) CLI queries with or without using **Query Optimizer** operator in watsonx.data by using the following steps. Also, watsonx.data allows inspecting the logs of the queries that are run.

Running Presto (C++) queries

1. **Option 1: Running Presto (C++) queries by using Query Optimizer**

   a. Run the following command to enter into the directory `ibm-lh-client/bin`:

   ```
   cd ibm-lh-client/bin
   ```

   b. Create an SQL file and export the file path to `LH_SANDBOX_DIR`. For example, with file name `sql-files`.

   ```
   export LH_SANDBOX_DIR=<path to sql-files>
   ```

   c. Get the list of engine names and choose the one to be used. For example, engine name `engine1`.

   ```
   ./manage-engines --op=list
   export engine_name=engine1
   ```

   d. Run the following command to run Presto (C++) queries using **Query Optimizer**.

   ```
   ./presto-run --engine=$engine_name --session enable_wxd_query_optimizer=true -f
   $LH_SANDBOX_DIR/sql-files.sql
   ```

   You must use either fully qualified name (3 part name such as **`<catalog.schema.table>`**) or 2 part name with the **USE** statement to qualify the catalog and schema.

   Examples: 3 part name: `select * from catalog.schema.table;`

   2 part name: `use "catalog".schema;` followed by `select * from schema.table;`

2. **Option 2: Running Presto (C++) CLI queries without using Query Optimizer**

   a. Run the following command to enter into the directory `ibm-lh-client/bin`:

   ```
   cd ibm-lh-client/bin
   ```

b. Create an SQL file and export the file path to `LH_SANDBOX_DIR`. For example, with file name `sql-files`.

```
export LH_SANDBOX_DIR=<path to sql-files>
```

c. Get the list of engine names and choose the one to be used. For example with engine name `engine1`.

```
./manage-engines --op=list
export engine_name=engine1
```

d. Run the following command to run Presto (Java) queries using **Query Optimizer**.

```
./presto-run --engine=$engine_name --session enable_wxd_query_optimizer=false -f
$LH_SANDBOX_DIR/sql-files.sql
```

Inspecting logs in watsonx.data

1. Run the following command to get **`Presto`** pod:

```
oc get pod -n <namespace> | grep "ibm-lh-lakehouse-presto"
```

2. Run the following command to get **`Prestissimo`** pod:

```
oc get pods -n cpd-instance | grep "ibm-lh-lakehouse-prestissimo"
```

3. Run the following command to enter into **`Presto`** pod:

```
oc exec -it <podname> -n cpd-instance bash
```

4. Run the following command to enter into **`Prestissimo`** pod:

```
oc exec -it <coordinator-podname> -n cpd-instance bash
```

5. Run the following commands to enable debug mode to view the Presto logs:

```
oc exec -it <presto/prestissimo pod> -n <namespace>
```

```
cd /opt/presto/etc
```

```
cat > log.properties

com.facebook.presto=INFO

com.facebook.presto.governance=DEBUG

com.facebook.presto.governance.util=DEBUG

com.facebook.presto.dispatcher=DEBUG
```

Note: Save the condition by using **`ctrl+d`**.

6. Run the following command to restart launcher:

```
cd /opt/presto/bin

./launcher_restart_handler.sh restart
```

7. Run one of the following commands to inspect the logs:

To inspect last 1000 logs:

```
tail -n 1000 /var/presto/data/var/log/server.log
```

To inspect live logs:

```
cd /var/presto/data/var/log
```

```
tail -f server.log
```

8. Search for the text '`Before optimization`'.
9. Map to the internal `dispatcher-query-xxx`.
10. Using the `dispatcher-query-xxx`, grep the line with text '`Presto successfully parsed optimized query`'.

    Sample response:

    ```
    2023-11-23T09:42:00.391Z INFO dispatcher-query-1764
    com.facebook.presto.governance.OptimizeQueryAuditAndGovernance    Before
    optimization - --q4
    2023-11-23T09:42:00.623Z    INFO    dispatcher-query-1764
    com.facebook.presto.dispatcher.DispatchManager    Presto (C++) successfully parsed
    optimized query
    ```

11. To check for query fall back, grep the line with text '`The optimisation failed(null or empty)... moving to original query`' using `dispatcher-query-xxx`.

    Sample response:

    ```

    2023-11-23T09:21:58.279Z    INFO    dispatcher-query-1741
    com.facebook.presto.dispatcher.DispatchManager    The optimisation failed(null
    or empty)....moving to original query - --q1
    ```

Deactivating Query Optimizer

Deactivate **Query Optimizer** in IBM watsonx.data by following the instructions.

About this task

Deactivate **`Query Optimizer`** operator in watsonx.data by using the following steps:

Procedure

1. Log in to watsonx.data console.
2. From the navigation menu, select **Configurations** and click **Query Optimizer Manager** tile.
3. Click **Deactivate**. A **Deactivate query optimizer** window displays. Click **Deactivate and restart engines**.

Uninstalling Query Optimizer

Uninstall **Query Optimizer** in IBM watsonx.data by following the instructions.

About this task

Uninstall **Query Optimizer** operator in watsonx.data by using the following steps:

Procedure

1. Run the following command to list all engines:

    ```
    -- List engines
    oc get wxdengine -n ${PROJECT_CPD_INST_OPERANDS} -o custom-
    columns='Name:metadata.name,Display Name:spec.engineDisplayName'
    ```

2. Run the following command to export the engine you want to uninstall **Query Optimizer** from:

    ```
    -- Select the engine to patch
    export ENGINE=<engine name>
    ```

3. Run the following command to uninstall **Query Optimizer** from the engine by patching it:

```
oc patch wxdengine $ENGINE -n ${PROJECT_CPD_INST_OPERANDS} --type json --patch '[
  { "op": "remove", "path": "/spec/wxd_query_optimizer" },
  { "op": "remove", "path": "/spec/enable_wxd_query_optimizer" },
  { "op": "remove", "path": "/spec/pull_prefix" },
  { "op": "remove", "path": "/spec/tag" },
  { "op": "remove", "path": "/spec/img_ref" }
]'
```

Note: This patching will restart the engine automatically.

4. Run the following command to check the status of uninstallation of **Query Optimizer** setup:

```
watch "oc get wxdengine -n $
{PROJECT_CPD_INST_OPERANDS} -o custom-columns='Name:metadata.name,Display
Name:spec.engineDisplayName,RECONCILE:status.engineStatus,STATUS:status.middleEndStatus'"
```

Note: Wait for the engine to finish reloading the **Query Optimizer** settings and show that the reconcile is "Completed".

5. Run the following command to uninstall all **Query Optimizer** resources that are no longer required from the operands namespace:

```
oc delete db2 lakehouse-oaas -n ${PROJECT_CPD_INST_OPERANDS}
oc delete secret ibm-lh-lakehouse-oaas-secret -n ${PROJECT_CPD_INST_OPERANDS}
```

6. Run the following command to uninstall `wxd_query_optimizer` operator:

```
cpd-cli manage delete-olm-artifacts \
--cpd_operator_ns=${PROJECT_CPD_INST_OPERATORS} \
--components=wxd_query_optimizer
```

Manage access

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Managing user access

Security in IBM watsonx.data is based on roles. A role is a group of permissions that control the actions you can perform in watsonx.data. To perform certain actions and manage specific sessions in watsonx.data, the user must also have the appropriate authorization.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

About this task

Authorization is granted by assigning a specific role to the user account. Use the Role Based Access Control feature in watsonx.data to grant users the access privileges they require for their role.

Procedure

1. Log in to the watsonx.data console.
2. From the navigation menu, select **Access control**. Under the **Infrastructure** tab, the different components (Engine, Catalog, Storage, and Database) are displayed in the table.
3. Click the overflow icon in the components row and then click **Manage access**. Alternatively, you can click the **Display name** of the component. The selected component page opens.
4. Under the **Access control** tab, click **Add access**.
5. In the **Grant access to users and user groups** window, provide the following details.

Field	Description
Name	You can select one or more users or user groups.
Role	Select the role from the drop-down list. You can assign roles based on the component type. For more information, see Infrastructure access.

6. Click **Add**. The user is added and assigned the role.
7. To change the role that is assigned to a user, complete the following steps:
    a) Under the **Infrastructure** tab, click the **Display name** of the component in the table.
    b) The **Access control** tab for selected component opens.
    c) Click the overflow menu for the selected user and then select **Change role**.
    d) In the **Change role** window, select the role from the drop-down list.
    e) Click **Save**.
8. To remove a user for a component, complete the following steps:
    a) Under the **Infrastructure** tab, click the **Display name** of the component in the table.
    b) The **Access control** tab for the selected component opens.
    c) Click the overflow menu for the selected user and then select **Remove**.
    d) In the **Confirm removal** window, click **Remove**.

    Note: When you add a user to watsonx.data instance and grant that user access control of a pre-defined catalog, that user cannot be removed.

Infrastructure access

Controlling access to the engines and other components is a critical requirement for many enterprises. To ensure that the resource usage is under control, IBM watsonx.data provides the ability to manage access controls on these resources. A user with admin privileges on the resources can grant access to other users.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

All users have access to the default iceberg-storage and hive-storage MinIO buckets, and default iceberg_data and hive_data Presto catalogs. An admin must grant explicit privileges to users who need access to the presto-01 engine.

The access control at the infrastructural level allows administrators to grant specific access to the wxd components—engines, catalogs, buckets, and databases.

- IBM watsonx.data users from CPD cluster must have `admin` role to activate their storage.
- IBM watsonx.data users with `manager` role on the engine can register their own storage but cannot activate the storage. They must contact the `admin` to activate their storage.
- Data access by using Presto engines is controlled by catalog roles (controlling the admin and user role privileges). The watsonx.data roles are mapped to their roles in the platform (IBM Cloud IAM roles or CPD instance roles). watsonx.data developer edition has its own user management to manage roles.
- Data ingestion by using Spark engines are controlled by storage roles.
- The catalog user role can access data only if the user is the data owner (that is, the creator of the schema or table) or has specific permissions such as select, insert through data policies.

Instance and install

Default admin access
    Instance admins (CPD) and Install admins (Dev) are the default administrators.

Default user access

Instance non-admins (CPD) and install non-admins (Dev) are the default users.

The following table explains the allowed actions for each role.

Table 31. Role-based access and privileges for instance and install

Action	Admin	User	Metastore Access
Create Presto (Java) or Presto (C++) engines	√		
Register Spark engines	√		
Create Milvus services	√		
Delete Milvus services	√		
View Milvus services	√		
Restart the internal HMS	√		
Scale Presto (Java) or Presto (C++) engines	√		
Unregister any storage	√		
Unregister any DB Connection	√		
Activate cataloged buckets (restart HMS)	√		
Register and unregister own storage	√	√	√
Register and unregister own DB connection	√	√	√
Access the metastore	√		√

Engines

An admin must grant explicit privileges to users who need access to an engine. The following tables explain the permitted actions for each role.

Presto (Java) or Presto (C++)

Default admin access

Instance admins (CPD) and Install admins (Dev) are the default administrators.

The following table explains the allowed actions for each role.

Table 32. Role-based access and privileges for Presto (Java) or Presto (C++) engine

Action	Admin	Manager	User	Users without an explicit role
Delete	√			
Grant and revoke access	√			
Pause and resume	√	√		
Restart	√	√		

Table 32. Role-based access and privileges for Presto (Java) or Presto (C++) engine (continued)

Action	Admin	Manager	User	Users without an explicit role
Associate and disassociate catalog	√	√		
Access the Presto (Java) or Presto (C++) query monitor UI	√	√		
View the engine	√	√	√	
Run workloads against the engine	√	√	√	

External Spark

Default admin access
Instance admins (CPD) and Install admins (Dev) are the default administrators.

The following table explains the allowed actions for each role.

Table 33. Role-based access and privileges for External Spark engine

Action	Admin	Manager	User	Users without an explicit role
Delete	√			
Grant and revoke access	√			
Update Spark engine metadata (like tags and description)	√	√		
View the engine	√	√	√	
Run workloads against the engine	√	√	√	

Co-located Spark (deprecated)

Role-based access control (RBAC) is based on zen service instance roles on Spark IAE instances.

Default admin access
Instance admins (CPD) and Install admins (Dev) are the default administrators.

Services

Milvus

Default admin access
Instance admins (CPD) and Install admins (Dev) are the default administrators.

The following table explains the allowed actions for each role.

Table 34. Role-based access and privileges for Milvus

Action	Admin	Editor	Viewer	User	Database creator (implicit role)	Collection creator (implicit role)	Partition creator (implicit role)
View assigned Milvus service	√	√	√	√			
Delete assigned Milvus service	√						
Grant access to assigned Milvus service	√						
Revoke access from assigned Milvus service	√						
Pause Milvus service	√						
Resume Milvus service	√						
Collection `CreateIndex`	√	√			√	√	
Collection `DropIndex`	√	√			√	√	
Global `CreateCollection`	√	√			√		
Global `DescribeCollection`	√	√	√		√	√	
Global `ShowCollections`	√	√	√		√		
Global `CreateAlias`	√	√			√		

Table 34. Role-based access and privileges for Milvus (continued)

Action	Admin	Editor	Viewer	User	Database creator (implicit role)	Collection creator (implicit role)	Partition creator (implicit role)
Global DropAlias	√	√			√		
Global DescribeAlias	√	√	√		√		
Global ListAliases	√	√	√		√	√	
Global FlushAll	√	√					
Global CreateResourceGroup	√						
Global DropResourceGroup	√						
Global DescribeResourceGroup	√						
Global ListResourceGroups	√						
Global TransferNode	√						
Global TransferReplica	√						
Global CreateDatabase	√	√					
Global DropDatabase	√	√			√		
Global ListDatabases	√	√	√				

Table 34. Role-based access and privileges for Milvus (continued)

Action	Admin	Editor	Viewer	User	Database creator (implicit role)	Collection creator (implicit role)	Partition creator (implicit role)
Collection `IndexDetail`	√	√	√		√	√	
Collection `Search`	√	√	√		√	√	√
Collection `Query`	√	√	√		√	√	√
Collection `Load`	√	√			√	√	√
Collection `GetLoadingProgress`	√	√			√	√	
Collection `GetLoadState`	√	√			√	√	
Collection `Release`	√	√			√	√	√
Collection `RenameCollection`	√	√			√	√	
Collection `DropCollection`	√	√			√	√	
Collection `Insert`	√	√			√	√	√
Collection `Delete`	√	√			√	√	√
Collection `Flush`	√	√			√	√	
Collection `GetFlushState`	√	√			√	√	
Collection `Upsert`	√	√			√	√	√
Collection `GetStatistics`	√	√			√	√	
Collection `Compaction`	√	√			√	√	

Table 34. Role-based access and privileges for Milvus (continued)

Action	Admin	Editor	Viewer	User	Database creator (implicit role)	Collection creator (implicit role)	Partition creator (implicit role)
Collection `Import`	√	√			√	√	
Collection `LoadBalance`	√	√			√	√	
Collection `CreatePartition`	√	√			√	√	
Collection `DropPartition`	√	√			√	√	√
Collection `ShowPartitions`	√	√	√		√	√	
Collection `HasPartition`	√	√	√		√	√	√

storage

Default admin access

Instance admins (CPD) and Install admins (Dev) are the default administrators.

All users can add their own storage and have admin access to it. Other users do not have access until they are granted explicit access. The following table explains the allowed actions for each role.

Table 35. Role-based access and privileges for storage

Action	Admin	Writer	Reader	Users without an explicit role
Unregister	√			
Update storage properties (credentials)	√			
Grant and revoke access	√			
Modify files	√	√		
Browse (storage browser in UI)	√	√	√	
View the storage	√	√	√	√

S3 REST API permissions (specific to IBM Spark and DAS)

Users can get a relative storage role for all subfolders and files in a storage or can be granted file action for particular folders or files. The following tables explain the storage-level and data-object-level S3 REST API permissions.

Note: The following tables are applicable only if you are using IBM Spark that by default uses a DAS signature or if you are using DAS proxy.

Storage-level access control
To assign storage-level access, go to **Access control** > **Infrastructure** or go to **Infrastructure manger** > select storage and assign roles.

Table 36. Storage-level access control

Storage role	**S3 REST API permission**
Reader	GET; HEAD
Writer	GET; HEAD; PUT; POST; PATCH; DELETE
Admin	GET; HEAD; PUT; POST; PATCH; DELETE

Data-object-level access control
To assign data-object-level access control, go to **Access control** > **Policies**.

Table 37. Data-object-level access control

Data object action	**S3 REST API permission**
Read	GET; HEAD
Write	GET; HEAD; PUT; PATCH; POST without `?delete` parameter
Delete	DELETE; POST with `?delete` parameter

Database

Default admin access (only if creator)
Instance admins (CPD) and Install admins (Dev) are the default administrators.

All users can add their own database and have admin access to it. Other users do not have access until they are granted explicit access. The following table explains the allowed actions for each role.

Table 38. Role-based access and privileges for database

Action	**Admin**	**Writer**	**Reader**	**Users without an explicit role**
Unregister	√			
Update db conn properties (credentials)	√			
Grant and revoke access	√			
Modify database objects	√	√		
View the database	√	√	√	√

Catalog

Every storage or a database must have a catalog that is associated with it. Admin of the storage or database is the admin of the associated catalog. Other users do not have access until they are granted explicit access.

Default admin access (based on access data control policies defined in watsonx.data by admin)
Instance admins (CPD) and Install admins (Dev) are the default administrators.

Default user access (based on access data control policies defined in watsonx.data by admin)
Instance non-admins (CPD) and install non-admins (Dev) are the default users.

The following table explains the allowed actions for each role.

Table 39. Role-based access and privileges for catalog

Action	Admin	User	Users without an explicit role
Delete	√		
Grant and revoke access	√		
Access to data	√	Based on data policy	
View the catalog	√	√	

Note: If you want to delete a catalog, you must first dissociate the catalog from the engine.

Schema

Default admin access (based on access data control policies defined in watsonx.data by admin)
Instance admins (CPD) and Install admins (Dev) are the default administrators.

Default user access (based on access data control policies defined in watsonx.data by admin)
Instance non-admins (CPD) and install non-admins (Dev) are the default users.

Action	Catalog Admin or schema creator	Others
Grant and revoke access	√	
Drop	√	
Access	√	based on access data control policies defined in watsonx.data by admin
Create table	√	based on access data control policies defined in watsonx.data by admin

Table

Default admin access (based on access data control policies defined in watsonx.data by admin)
Instance admins (CPD) and Install admins (Dev) are the default administrators.

Default user access (based on access data control policies defined in watsonx.data by admin)
Instance non-admins (CPD) and install non-admins (Dev) are the default users.

Action	Catalog Admin or schema admin or table creator	Others
Create, drop, and alter	√	based on access data control policies defined in watsonx.data by admin
Column access	√	based on access data control policies defined in watsonx.data by admin

Action	Catalog Admin or schema admin or table creator	Others
Select	√	based on access data control policies defined in watsonx.data by admin
Insert	√	based on access data control policies defined in watsonx.data by admin
Update	√	based on access data control policies defined in watsonx.data by admin
Delete	√	based on access data control policies defined in watsonx.data by admin

Data policy

Protecting access to data is a critical requirement for many enterprises. To ensure that your data is protected from unauthorized access, IBM watsonx.data allows admin to enforce access controls for data. A user with admin privileges on the data can create access policies to define, extend, limit, and deny access, by using the data security solution that is provided by watsonx.data.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

About this task

To maintain data security, you can create access policies for schemas, tables, and columns by permitting actions to individual users or group of users.

Ensure that you have Admin access to the catalog, storage, or service. For more information, see Infrastructure access.

Note: Ensure to add the users and user groups to the installation before you begin creating a policy. For more information, see Managing users.

Procedure

1. Access the instances page, locate the watsonx.data instance and click the overflow menu to open the watsonx.data console.
2. From the navigation menu, select **Access control**.
3. Go to the **Policies** tab and click **Add policy**. The **Create access control policy** page opens.
4. In the **Create access control policy** page, provide the following details to add a new policy.
5. In the **Details** page, enter the following details and click **Next**:

Field	Description
Policy name	Enter a name.
Policy description (Optional)	Give a brief description.
Policy status after creation	Set the status to activate the policy at the time of creation or later.

6. In the **Data objects** page, select a resource from the drop-down list.

   You can select one of the following categories:

- Eligible catalogs
- storages
- Eligible services

Eligible catalogs

a. Select a catalog.

b. Choose one, more than one, or all schemas.

Note:

- If you choose a single schema, you can select one, more than one, or all tables.
- If you choose more than one schema, you cannot select any tables. The policy applies to all tables within the schemas.

c. Choose one, more than one, or all tables.

Note:

- If you choose a single table, you can select one, more than one, or all columns.
- If you choose more than one table, you cannot select any columns. The policy applies to all columns of the tables.

Storages

a. Select a storage.

b. Choose an object. Choose **Regular Expression** to enter the object path manually or **Explore object path** to search and select the object.

Eligible services

a. Select a service.

Note: Currently, Milvus is the only service available. You can define policies to a Milvus service directly without selecting any databases. Select the service and proceed with step 7.

b. Choose one, more than one, or all databases.

c. Choose one, more than one, or all collections.

Note:

- If you choose a single database, you can select one, more than one, or all collections.
- If you choose more than one database, you cannot select any collections. The policy applies to all collections in the selected databases.

7. Click **Next** to add rules.
8. In the **Rules** page:
   a) Click **Add rule** to go to the **Add rule** page.
   b) Select the rule type **Allow** or **Deny**.
   c) Select the actions on the data objects. The list of actions depend on the data object chosen in the earlier page. You can select one or more actions.
   d) Choose if you want to grant access to individual users or a group of users from the **Users/groups** section.
   e) Select a user or user group from the drop-down list. Ensure to add the users and user groups to the installation. For more information, see Managing users.
   f) Click **Add** to add the rule.
   g) You can add more rules or click **Review**. the **Summary** page opens.
9. In the **Summary** page:
   a) You can review the policy.
   b) Click **Back** to go to the previous page.

c) Click **Cancel** to cancel the process.

d) Or click **Save** to save the policy.

Using DAS proxy to access S3 and S3 compatible buckets

External applications and query engines can access the S3 and S3 compatible buckets that are managed by watsonx.data through DAS proxy.

watsonx.data on Red Hat OpenShift

To access the S3 and S3 compatible buckets:

1. Get the DAS endpoint from the watsonx.data information window. Click the **i** icon on the home page to open the information window.
2. Replace the S3 endpoint with the DAS endpoint in your Java code.

   ```
   <cas endpoint>/cas/v1/proxy
   ```

3. Replace the access key with the encoded value in your Java code as follows:

   ```
   base64{<instanceid>|ZenAPIkey base64{username:<apikey>}}
   ```

   Note: To get the Base64 encoded string, use one of the following commands:

   -
     ```
     printf "username:<apikey>" | base64
     ```

   -
     ```
     echo -n "username:<apikey>" | base64
     ```

Java code example to use DAS:

```
        String bucketName = "bucket1";
        String keyName = "folder1/file1";
        # replace the target object store endpoint with the DAS proxy endpoint
        String endpoint = "<cas endpoint get from About page>/cas/v1/proxy";
        /** Replace the Access Key with watsonx.data user name and API key following the below
base64 encoded method.
        * CPD base64{<instanceid>|ZenAPIkey base64{username:<apikey>}}
        * SaaS base64{<crn>|Basic base64{ibmlhapikey_<user_id>:<IAM_APIKEY>}}
        */
        String accessKey = "encoded value";
        String secretKey = "any string";

        BasicAWSCredentials cos_cred = new BasicAWSCredentials(accessKey, secretKey);
        EndpointConfiguration cosEndPoint = new EndpointConfiguration(endpoint, "us-east");
        AmazonS3 s3Client = AmazonS3ClientBuilder.standard().withPathStyleAccessEnabled(true)
                .withCredentials(new AWSStaticCredentialsProvider(cos_cred))
                .withEndpointConfiguration(cosEndPoint).build();
        GetObject.GetObjectTest(s3Client, bucketName, keyName);
```

Note: For information about S3 REST API permissions, see S3 REST API permissions.

Note: To use DAS externally, you have to import the CA certificates and add them in your cluster. To get the certificates, run:

```
echo QUIT | openssl s_client -showcerts -connect <cas host>:443 | awk '/-----BEGIN
CERTIFICATE-----/ {p=1}; p; /-----END CERTIFICATE-----/ {p=0}' > cas.cert
```

To import the certificates to your local Java truststore, run:

```
sudo keytool -import -trustcacerts -cacerts -storepass changeit -noprompt -alias cas-cert
-file ./cas.cert
```

Using DAS proxy to access ADLS and ABS compatible buckets

External applications and query engines can access the Azure Data Lake Storage (ADLS) and Azure Blob Storage (ABS) compatible buckets that are managed by watsonx.data through DAS proxy.

watsonx.data on Red Hat OpenShift

Attention: DAS proxy support for ADLS and ABS works only with `AccountKey` to pass watsonx.data credential. Using SASToken to pass watsonx.data credential is not supported.

To access the ADLS and ABS compatible buckets:

1. Get the DAS endpoint from the watsonx.data information window. Click the **i** icon on the home page to open the information window.
2. Replace the ADLS or ABS endpoint with the DAS endpoint in your Java code. Replace the access name with the encoded value as follows:

   ```
   <ADLS or ABS account name>|base64{<instanceid>|ZenApikey base64{username:<apikey>}}
   ```

   Note: To get the Base64 encoded string, use one of the following commands:

   -
     ```
     printf "username:<apikey>" | base64
     ```

   -
     ```
     echo -n "username:<apikey>" | base64
     ```
3. Replace the container name in the Java code as follows:

   ```
   cas/v1/proxy/<bucketname in watsonx.data>
   ```

Java code example to use DAS:

```
      //For SaaS <ADLS or ABS account name>|base64{<crn>|Basic
base64{ibmlhapikey_<user_id>:<IAM_APIKEY>}}
      //For watsonx.data on AWS <ADLS or ABS account name>|base64{<crn>|Basic
base64{ibmlhapikey_ServiceId-<service_id>:<APIKEY>}}
      String accountName = "<ADLS or ABS account name>|base64{<instanceid>|ZenApikey
base64{username:<apikey>}}";
      String accountKey = "any string";
      String endPoint = "<DAS endpoint>";
      String containerName ="cas/v1/proxy/<bucketname in watsonx.data>";
      String endpoint = String.format(endPoint,accountName);

      BlobServiceClient blobServiceClient = new BlobServiceClientBuilder()
                .endpoint(endpoint)
                .credential(new StorageSharedKeyCredential(accountName, accountKey))
                .buildClient();

      BlobContainerClient containerClient =
blobServiceClient.getBlobContainerClient(containerName);
```

Working with data objects

You can do different operations in watsonx.data to create schema, table, enable listener, and sync data.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Creating schemas using CLI

You can create the schemas using the Presto CLI.

watsonx.data on Red Hat OpenShift

Before you begin

Important: You must specify the location when creating schema using CLI. For example,

```
location = s3a://<bucket-name>/
```

To create schemas in Presto, you must define the bucket to the target.

For example, to create a schema in the `ibmcos` catalog:

```
$ bin/presto-cli --catalog ibmcos
using /home/demouser/test/mar10/ibm-lh-dev/localstorage/volumes as data root directory for
user: demouser/1001
infra config location is /home/demouser/test/mar10/ibm-lh-dev/localstorage/volumes/infra
==== starting: presto-cli ====
presto> create schema myschema with (location='s3a://lh-demo-01/myschema_subpath');
CREATE SCHEMA
```

Note: The use of a `subpath` under the root of bucket is optional, but recommended for a better organization in your bucket.

About this task

Complete the following steps to create schema by using Presto CLI.

Note:

- The code in following steps is an example. Replace this code with your own catalog and schema name.
- Not all catalogs support creation of schemas.

Procedure

1. Inspect the contents of the object store by navigating to `http://localhost:9001`. Log in with object store credentials.
2. Log in to Presto CLI. For instructions to connect to Presto server from Presto CLI, see Chapter 7, "Connecting to a Presto server," on page 450.

   ```
   bin/presto-cli.sh --catalog <catalog_name>
   ```

3. Create a schema with the specified location.

   ```
   presto> CREATE SCHEMA IF NOT EXISTS <schema_name> with (location='s3a://ibm-lakehouse/');
       CREATE SCHEMA
   ```

4. Verify whether the schema is created.

   ```
   presto:default> show schemas;
          Schema
       --------------------
        default
        information_schema
        workshop
       (3 rows)

       Query 20230309_150053_00052_hy4cs, FINISHED, 1 node
       Splits: 19 total, 19 done (100.00%)
       251ms [3 rows, 48B] [11 rows/s, 191B/s]

       presto:default> use workshop;
       USE
   ```

Creating tables using CLI

You can create tables using the Presto CLI.

watsonx.data on Red Hat OpenShift

About this task

Complete the following steps to create table using the Presto CLI.

Procedure

1. Log in to Presto CLI. For instructions to connect to Presto server from Presto CLI, see Chapter 7, "Connecting to a Presto server," on page 450.

   ```
   bin/presto-cli.sh --catalog <catalog name>
   ```

2. Run the following command to create a table:

   ```
   use <catalog name>.<schema name>;
   create table <table name>(column1 datatype1, ...);
   ```

   Note: If you do not have the schema, see Creating schema using CLI to create schema, and then create a table.

Creating and inserting Parquet v1 tables in Presto (Java)

This topic describes how to control the Parquet version used by Presto (Java) when ingesting data. By default, Presto (Java) ingests data in Parquet v2 format for Iceberg tables and ORC format for Hive tables. However, this behavior of Presto (Java) poses a challenge if you have to read the tables from Presto (C++), as Presto (C++) can read tables in Parquet v1 format only.

watsonx.data on Red Hat OpenShift

About this task

Presto (C++) engines cannot read Parquet v2 tables created by Presto (Java). Presto (C+
+) only supports DWRF and Parquet v1 formats. Therefore, you must set session property `<catalog_name>.parquet_writer_version` to `PARQUET_1_0` before ingesting data with Presto (Java) engine.

Procedure

1. Log in to Presto (Java) CLI. For instructions to connect to Presto (Java) server from Presto (Java) CLI, see Connecting to a Presto (Java) server.

   ```
   bin/presto-cli.sh --catalog <catalog name>
   ```

2. Set the session property before ingesting data to a table:

   ```
   set session <catalog_name>.parquet_writer_version = 'PARQUET_1_0';
   ```

3. Run the following command to create a table using CTAS:

   ```
   create table <catalog name>.<schema name>.<table name> as (select * from tpch.tiny.customer
   limit 10);
   ```

   Note: Setting the `format_version` property during `CREATE TABLE` does not influence the Parquet version. You must set the session property before ingesting data into an existing Parquet v2 table.

Capturing DDL changes in watsonx.data through event listener

When tables or schemas are created, altered, or deleted in watsonx.data, an event can be captured and notified by enabling an event listener.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Before you begin

Deploy a Kafka server to subscribe and publish the captured event. The server can either be an on-prem server or a service in the public cloud. For example, Event Stream in IBM Cloud. Events are captured and stored in Kafka and can be subscribed by consumers.

Procedure

1. Enable the listener through watsonx.data REST API to capture the changes.

   POST `https://{hostname}/lakehouse/api/v2/{instance id}/configure_hms_listener`.

   Request Payload (JSON):

```
{
  "kafka_properties":[
    {
      "key":"hive.metastore.kafka.bootstrap.server",
      "value":"kafka-server-1:port,kafka-server-2:port",
      "encrypt":false
    },
    {
      "key":"hive.metastore.kafka.topic.name",
      "value":"TOPIC",
      "encrypt":false},
    {
      "key":"hive.metastore.kafka.topic.message.type",
      "value":"THRIFT_HIVE_JSON",
      "encrypt":false}
    ]
}
```

   The listener is enabled a few minutes after the API returns a success. The object change event is sent to the topic in Kafka if DDL is performed on table or schema. You can get the event from the topic.

2. If required, change the listener settings and call the API again with a different value.
3. Retrieve the current listener settings through this REST API:

   POST `https://{hostname}/lakehouse/api/v2/{instance id}/metastore_custom_properties`.

   Supported properties:

Properties	**Default value**	**Description**
`hive.metastore.kafka.bootstrap.server`		(Required) URL of Kafka brokerlist, separated by comma.
`hive.metastore.kafka.topic.name`	HMS_MSG	(Optional) The topic name used in Kafka. An event is sent to the specified topic. If a topic does not exist, watsonx.data creates it in Kafka.
`hive.metastore.kafka.topic.message.type`	HIVE_JSON	(Optional) Specifies the format of the event to be generated. - HIVE_JSON: JSON - THRIFT_HIVE_JSON: JSON string serialized by `org.apache.thrift.TSerializer`
`hive.metastore.kafka.username`		(Required if SASL is enabled) Username of Kafka server if authentication is enabled in Kafka server.
`hive.metastore.kafka.password`		(Required if SASL is enabled) Password of Kafka server if authentication is enabled in Kafka server.

Properties	Default value	Description
`hive.metastore.kafka.topic.replication.factor`	0	(Required if specified topic does not exist in Kafka.) The replication factors for creating topics. If the setting is different from the replication factors in Kafka, topic creation can fail.
`hive.metastore.kafka.topic.partition.number`	1	(Optional) The number of partitions per topic, which is used for creating topic.
`hive.metastore.kafka.security.protocol`		(Required if SASL is enabled and a non-default value is used in Kafka.) Protocol setting in Kafka to communicate with Kafka brokers.
`hive.metastore.kafka.sasl.mechanism`		(Required if SASL is enabled and a non-default value is used in Kafka.) SASL mechanism used for client connections.
`hive.metastore.kafka.truststore.certificates`		(Required if certificate is used in Kafka and `hive.metastore.kafka.disable.cert.verification` is false) Base64 encoded PEM cert chain string. The string can be generated by the command `echo "`cat <your_cert_file>`" | base64 -w 0`
`hive.metastore.kafka.disable.cert.verification`		(Optional) Disable cert verification when value is ‘true’ (case-insensitive). No effect for other values.
`hive.metastore.kafka.listener.partition.event.enabled`	`false`	(Optional) Capture partition event

Data is captured for the following events:

- Create schema
- Drop schema
- Create table
- Alter table
- Drop table
- Add partition
- Alter partition

4. When an event is triggered, details about the event are pushed to the Kafka server. You must subscribe to the Kafka topic through an application or Kafka client, and then the event is pushed by the Kafka server. The changed object (table/schema) details are included in the event. For `alter` event, the event includes the object details before and after the `alter`.

Example: Create table event

The DDL performed is:

```
CREATE TABLE "iceberg_data"."example"."table1"
(
    "ID" INT,
    "Name" VARCHAR
)
```

THRIFT_HIVE_JSON

Details about an event in `THRIFT_HIVE_JSON` format contain event type, schema name, table name, and table details. Table details are serialized as a string. An event size is smaller than `HIVE_JSON`.

```
{
  "eventType": "CREATE_TABLE",
  "dbName": "example",
  "tableName": "table1",
  "tableType": "EXTERNAL_TABLE",
  "table": "{\"1\":{\"str\":\"table1\"},\"2\":{\"str\":\"example\"},\"3\":
{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1699947381},\"5\":{\"i32\":0},\"6\":
{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"lst\":[\"rec\",2,{\"1\":{\"str\":\"id\"},\"2\":
{\"str\":\"int\"}},{\"1\":{\"str\":\"name\"},\"2\":{\"str\":\"string\"}}]},\"2\":
{\"str\":\"s3a://iceberg-bucket/example/table1\"},\"3\":
{\"str\":\"org.apache.hadoop.mapred.FileInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.mapred.FileOutputFormat\"},\"5\":{\"tf\":0},\"6\":
{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe\"},\"3\":{\"map\":
[\"str\",\"str\",0,{}]}}},\"10\":{\"map\":[\"str\",\"str\",0,{}]}}},\"8\":{\"lst\":
[\"rec\",0]},\"9\":{\"map\":[\"str\",\"str\",7,
{\"totalSize\":\"1226\",\"EXTERNAL\":\"TRUE\",\"numFiles\":\"1\",\"metadata_location\":\"s3a:
//iceberg-bucket/example/table1/metadata/
00000-12f298e9-747c-4730-9062-3784b7777280.metadata.json\",\"transient_lastDdlTime\":\"169994
7381\",\"numFilesErasureCoded\":\"0\",\"table_type\":\"iceberg\"}]},\"12\":
{\"str\":\"EXTERNAL_TABLE\"},\"13\":{\"rec\":{\"1\":{\"map\":[\"str\",\"lst\",1,
{\"ibmlhadmin\":[\"rec\",4,{\"1\":{\"str\":\"select\"},\"2\":{\"i32\":0},\"3\":
{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1},\"5\":{\"tf\":1}},{\"1\":
{\"str\":\"insert\"},\"2\":{\"i32\":0},\"3\":{\"str\":\"ibmlhadmin\"},\"4\":
{\"i32\":1},\"5\":{\"tf\":1}},{\"1\":{\"str\":\"update\"},\"2\":{\"i32\":0},\"3\":
{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1},\"5\":{\"tf\":1}},{\"1\":
{\"str\":\"delete\"},\"2\":{\"i32\":0},\"3\":{\"str\":\"ibmlhadmin\"},\"4\":
{\"i32\":1},\"5\":{\"tf\":1}}]}]},\"2\":{\"map\":[\"str\",\"lst\",0,{}]},\"3\":{\"map\":
[\"str\",\"lst\",0,{}]}}},\"17\":{\"str\":\"hive\"},\"18\":{\"i32\":1},\"25\":{\"i64\":1}}"
}
```

You can check the event types and the table information from JSON without de-serializing the table details. If you need the details, import the library `org.apache.thrift` and get `Table Object` from the JSON string in event. You can retrieve the details from the Java Object.

```
import org.apache.thrift.TDeserializer;
import org.apache.thrift.protocol.TJSONProtocol;
import org.apache.hadoop.hive.metastore.api.Table;
...
//Get string from "table" field first
TDeserializer deSerializer = new TDeserializer(new TJSONProtocol.Factory());
org.apache.hadoop.hive.metastore.api.Table table = new Table();
deSerializer.deserialize(table, tableStr, "UTF-8");
List<FieldSchema> columns = table.getSd().getCols()
```

HIVE_JSON

Details about an event in `HIVE_JSON` format contain event type and table details in `JSON` format. You can parse the `JSON` by any `JSON` library and get the details.

```
{
  "eventType": "CREATE_TABLE",
  "table": {
    "tableName": "table1",
    "dbName": "example",
    "owner": "ibmlhadmin",
    "createTime": 1699948569,
    "lastAccessTime": 0,
    "retention": 0,
    "sd": {
      "cols": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        }
      ],
      "location": "s3a://iceberg-bucket/example/table1",
      "inputFormat": "org.apache.hadoop.mapred.FileInputFormat",
      "outputFormat": "org.apache.hadoop.mapred.FileOutputFormat",
      "compressed": false,
      "numBuckets": 0,
      "serdeInfo": {
        "name": "table1",
        "serializationLib": "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
        "parameters": {},
        "description": null,
        "serializerClass": null,
        "deserializerClass": null,
        "serdeType": null,
        "setName": true,
        "setSerializationLib": true,
        "parametersSize": 0,
        "setParameters": true,
        "setDescription": false,
        "setSerializerClass": false,
        "setDeserializerClass": false,
        "setSerdeType": false
      },
      "bucketCols": null,
      "sortCols": null,
      "parameters": {},
      "skewedInfo": null,
      "storedAsSubDirectories": false,
      "colsSize": 2,
      "colsIterator": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        }
      ],
      "setCols": true,
      "setLocation": true,
      "setInputFormat": true,
      "setOutputFormat": true,
      "setCompressed": true,
      "setNumBuckets": true,
      "setSerdeInfo": true,
      "bucketColsSize": 0,
      "bucketColsIterator": null,
      "setBucketCols": false,
      "sortColsSize": 0,
      "sortColsIterator": null,
      "setSortCols": false,
      "parametersSize": 0,
      "setParameters": true,
      "setSkewedInfo": false,
      "setStoredAsSubDirectories": false
    },
    "partitionKeys": [],
    "parameters": {
      "totalSize": "4294",
      "EXTERNAL": "TRUE",
      "numFiles": "3",
      "metadata_location": "s3a://iceberg-bucket/example/table1/metadata/
00000-5814ea67-633c-4b4c-abe6-d85b40d68ea8.metadata.json",
      "transient_lastDdlTime": "1699948569",
      "numFilesErasureCoded": "0",
      "table_type": "iceberg"
    },
    "viewOriginalText": null,
    "viewExpandedText": null,
    "tableType": "EXTERNAL_TABLE",
    "privileges": {
      "userPrivileges": {
        "ibmlhadmin": [
          {
            "privilege": "select",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          },
          {
            "privilege": "insert",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          },
          {
            "privilege": "update",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          },
          {
            "privilege": "delete",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          }
        ]
      },
      "groupPrivileges": {},
      "rolePrivileges": {},
      "userPrivilegesSize": 1,
      "setUserPrivileges": true,
      "groupPrivilegesSize": 0,
      "setGroupPrivileges": true,
      "rolePrivilegesSize": 0,
      "setRolePrivileges": true
    },
    "temporary": false,
    "rewriteEnabled": false,
    "creationMetadata": null,
    "catName": "hive",
    "ownerType": "USER",
    "writeId": -1,
    "isStatsCompliant": false,
    "colStats": null,
    "accessType": 0,
    "requiredReadCapabilities": null,
    "requiredWriteCapabilities": null,
    "id": 11,
    "fileMetadata": null,
    "dictionary": null,
    "txnId": 0,
    "setTableName": true,
    "setDbName": true,
    "setOwner": true,
    "setCreateTime": true,
    "setLastAccessTime": true,
      "setRetention": true,
      "setSd": true,
      "partitionKeysSize": 0,
      "partitionKeysIterator": [],
      "setPartitionKeys": true,
      "parametersSize": 7,
      "setParameters": true,
      "setViewOriginalText": false,
      "setViewExpandedText": false,
      "setTableType": true,
      "setPrivileges": true,
      "setTemporary": false,
      "setRewriteEnabled": false,
      "setCreationMetadata": false,
      "setCatName": true,
      "setOwnerType": true,
      "setWriteId": false,
      "setIsStatsCompliant": false,
      "setColStats": false,
      "setAccessType": false,
      "requiredReadCapabilitiesSize": 0,
      "requiredReadCapabilitiesIterator": null,
      "setRequiredReadCapabilities": false,
      "requiredWriteCapabilitiesSize": 0,
      "requiredWriteCapabilitiesIterator": null,
      "setRequiredWriteCapabilities": false,
      "setId": true,
      "setFileMetadata": false,
      "setDictionary": false,
      "setTxnId": false
    }
  }
```

Example

Event content example:

THRIFT_HIVE_JSON

Create schema

```
{
  "eventType": "CREATE_DATABASE",
  "dbName": "example",
  "db": "{\"1\":{\"str\":\"example\"},\"3\":{\"str\":\"s3a://iceberg-
bucket/example\"},\"4\":{\"map\":[\"str\",\"str\",0,{}]},\"6\":{\"str\":\"ibmlhadmin\"},\"7\":
{\"i32\":1},\"8\":{\"str\":\"hive\"},\"9\":{\"i32\":1699947172}}"
}
```

Drop schema

```
{
  "eventType": "DROP_DATABASE",
  "dbName": "example",
  "db": "{\"1\":{\"str\":\"example\"},\"3\":{\"str\":\"s3a://iceberg-
bucket/example\"},\"4\":{\"map\":[\"str\",\"str\",0,{}]},\"6\":{\"str\":\"ibmlhadmin\"},\"7\":
{\"i32\":1},\"8\":{\"str\":\"hive\"},\"9\":{\"i32\":1699947172},\"11\":{\"i32\":1}}"
}
```

Create table

```
{
  "eventType": "CREATE_TABLE",
  "dbName": "example",
  "tableName": "table1",
  "tableType": "EXTERNAL_TABLE",
  "table": "{\"1\":{\"str\":\"table1\"},\"2\":{\"str\":\"example\"},\"3\":
{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1699947381},\"5\":{\"i32\":0},\"6\":{\"i32\":0},\"7\":
{\"rec\":{\"1\":{\"lst\":[\"rec\",2,{\"1\":{\"str\":\"id\"},\"2\":{\"str\":\"int\"}},{\"1\":
{\"str\":\"name\"},\"2\":{\"str\":\"string\"}}]},\"2\":{\"str\":\"s3a://iceberg-bucket/example/
table1\"},\"3\":{\"str\":\"org.apache.hadoop.mapred.FileInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.mapred.FileOutputFormat\"},\"5\":{\"tf\":0},\"6\":
{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe\"},\"3\":{\"map\":
[\"str\",\"str\",0,{}]}}},\"10\":{\"map\":[\"str\",\"str\",0,{}]}}},\"8\":{\"lst\":
[\"rec\",0]},\"9\":{\"map\":[\"str\",\"str\",7,
{\"totalSize\":\"1226\",\"EXTERNAL\":\"TRUE\",\"numFiles\":\"1\",\"metadata_location\":\"s3a://
iceberg-bucket/example/table1/metadata/
00000-12f298e9-747c-4730-9062-3784b7777280.metadata.json\",\"transient_lastDdlTime\":\"169994738
1\",\"numFilesErasureCoded\":\"0\",\"table_type\":\"iceberg\"}]},\"12\":
{\"str\":\"EXTERNAL_TABLE\"},\"13\":{\"rec\":{\"1\":{\"map\":[\"str\",\"lst\",1,{\"ibmlhadmin\":
[\"rec\",4,{\"1\":{\"str\":\"select\"},\"2\":{\"i32\":0},\"3\":{\"str\":\"ibmlhadmin\"},\"4\":
{\"i32\":1},\"5\":{\"tf\":1}},{\"1\":{\"str\":\"insert\"},\"2\":{\"i32\":0},\"3\":
{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1},\"5\":{\"tf\":1}},{\"1\":{\"str\":\"update\"},\"2\":
{\"i32\":0},\"3\":{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1},\"5\":{\"tf\":1}},{\"1\":
{\"str\":\"delete\"},\"2\":{\"i32\":0},\"3\":{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1},\"5\":
{\"tf\":1}}]}]},\"2\":{\"map\":[\"str\",\"lst\",0,{}]},\"3\":{\"map\":[\"str\",\"lst\",0,
{}]}}},\"17\":{\"str\":\"hive\"},\"18\":{\"i32\":1},\"25\":{\"i64\":1}}"
}
```

Alter table

```
{
  "eventType": "ALTER_TABLE",
  "dbName": "example",
  "tableName": "table1",
  "tableType": "EXTERNAL_TABLE",
  "oldTable": "{\"1\":{\"str\":\"table1\"},\"2\":{\"str\":\"example\"},\"3\":
{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1699947381},\"5\":{\"i32\":0},\"6\":{\"i32\":0},\"7\":
{\"rec\":{\"1\":{\"lst\":[\"rec\",2,{\"1\":{\"str\":\"id\"},\"2\":{\"str\":\"int\"}},{\"1\":
{\"str\":\"name\"},\"2\":{\"str\":\"string\"}}]},\"2\":{\"str\":\"s3a://iceberg-bucket/example/
table1\"},\"3\":{\"str\":\"org.apache.hadoop.mapred.FileInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.mapred.FileOutputFormat\"},\"5\":{\"tf\":0},\"6\":
{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe\"},\"3\":{\"map\":
[\"str\",\"str\",0,{}]}}},\"8\":{\"lst\":[\"str\",0]},\"9\":{\"lst\":[\"rec\",0]},\"10\":
{\"map\":[\"str\",\"str\",0,{}]},\"11\":{\"rec\":{\"1\":{\"lst\":[\"str\",0]},\"2\":{\"lst\":
[\"lst\",0]},\"3\":{\"map\":[\"lst\",\"str\",0,{}]}}},\"12\":{\"tf\":0}}},\"8\":{\"lst\":
[\"rec\",0]},\"9\":{\"map\":[\"str\",\"str\",7,
{\"totalSize\":\"1226\",\"EXTERNAL\":\"TRUE\",\"numFiles\":\"1\",\"transient_lastDdlTime\":\"169
9947381\",\"metadata_location\":\"s3a://iceberg-bucket/example/table1/metadata/
00000-12f298e9-747c-4730-9062-3784b7777280.metadata.json\",\"numFilesErasureCoded\":\"0\",\"tabl
e_type\":\"iceberg\"}]},\"12\":{\"str\":\"EXTERNAL_TABLE\"},\"15\":{\"tf\":0},\"17\":
{\"str\":\"hive\"},\"18\":{\"i32\":1},\"19\":{\"i64\":0},\"25\":{\"i64\":1}}",
  "newTable": "{\"1\":{\"str\":\"table1\"},\"2\":{\"str\":\"example\"},\"3\":
{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":0},\"5\":{\"i32\":0},\"6\":{\"i32\":0},\"7\":{\"rec\":
{\"1\":{\"lst\":[\"rec\",3,{\"1\":{\"str\":\"id\"},\"2\":{\"str\":\"int\"}},{\"1\":
{\"str\":\"name\"},\"2\":{\"str\":\"string\"}},{\"1\":{\"str\":\"age\"},\"2\":
{\"str\":\"int\"}}]},\"2\":{\"str\":\"s3a://iceberg-bucket/example/table1\"},\"3\":
{\"str\":\"org.apache.hadoop.mapred.FileInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.mapred.FileOutputFormat\"},\"5\":{\"tf\":0},\"6\":
{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe\"},\"3\":{\"map\":
[\"str\",\"str\",0,{}]}}},\"10\":{\"map\":[\"str\",\"str\",0,{}]}}},\"8\":{\"lst\":
[\"rec\",0]},\"9\":{\"map\":[\"str\",\"str\",8,{\"previous_metadata_location\":\"s3a://iceberg-
bucket/example/table1/metadata/
00000-12f298e9-747c-4730-9062-3784b7777280.metadata.json\",\"totalSize\":\"3068\",\"EXTERNAL\":\
"TRUE\",\"numFiles\":\"2\",\"transient_lastDdlTime\":\"1699947381\",\"metadata_location\":\"s3a:
//iceberg-bucket/example/table1/metadata/00001-e358db94-34f8-4a59-b5ca-
a17022edf957.metadata.json\",\"numFilesErasureCoded\":\"0\",\"table_type\":\"iceberg\"}]},\"12\"
:{\"str\":\"EXTERNAL_TABLE\"},\"13\":{\"rec\":{\"1\":{\"map\":[\"str\",\"lst\",1,
{\"ibmlhadmin\":[\"rec\",4,{\"1\":{\"str\":\"select\"},\"2\":{\"i32\":0},\"3\":
{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1},\"5\":{\"tf\":1}},{\"1\":{\"str\":\"insert\"},\"2\":
{\"i32\":0},\"3\":{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1},\"5\":{\"tf\":1}},{\"1\":
{\"str\":\"update\"},\"2\":{\"i32\":0},\"3\":{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1},\"5\":
{\"tf\":1}},{\"1\":{\"str\":\"delete\"},\"2\":{\"i32\":0},\"3\":{\"str\":\"ibmlhadmin\"},\"4\":
{\"i32\":1},\"5\":{\"tf\":1}}]}]},\"2\":{\"map\":[\"str\",\"lst\",0,{}]},\"3\":{\"map\":
[\"str\",\"lst\",0,{}]}}},\"17\":{\"str\":\"hive\"},\"18\":{\"i32\":1}}"
}
```

Drop table

```
{
  "eventType": "DROP_TABLE",
  "dbName": "example",
  "tableName": "table1",
  "tableType": "EXTERNAL_TABLE",
  "table": "{\"1\":{\"str\":\"table1\"},\"2\":{\"str\":\"example\"},\"3\":
{\"str\":\"ibmlhadmin\"},\"4\":{\"i32\":1699947381},\"5\":{\"i32\":0},\"6\":{\"i32\":0},\"7\":
{\"rec\":{\"1\":{\"lst\":[\"rec\",3,{\"1\":{\"str\":\"id\"},\"2\":{\"str\":\"int\"}},{\"1\":
{\"str\":\"name\"},\"2\":{\"str\":\"string\"}},{\"1\":{\"str\":\"age\"},\"2\":
{\"str\":\"int\"}}]},\"2\":{\"str\":\"s3a://iceberg-bucket/example/table1\"},\"3\":
{\"str\":\"org.apache.hadoop.mapred.FileInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.mapred.FileOutputFormat\"},\"5\":{\"tf\":0},\"6\":
{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe\"},\"3\":{\"map\":
```

```
[\"str\",\"str\",0,{}]}}},\"8\":{\"lst\":[\"str\",0]},\"9\":{\"lst\":[\"rec\",0]},\"10\":
{\"map\":[\"str\",\"str\",0,{}]},\"11\":{\"rec\":{\"1\":{\"lst\":[\"str\",0]},\"2\":{\"lst\":
[\"lst\",0]},\"3\":{\"map\":[\"lst\",\"str\",0,{}]}}},\"12\":{\"tf\":0}}},\"8\":{\"lst\":
[\"rec\",0]},\"9\":{\"map\":[\"str\",\"str\",8,{\"previous_metadata_location\":\"s3a://iceberg-
bucket/example/table1/metadata/
00000-12f298e9-747c-4730-9062-3784b7777280.metadata.json\",\"totalSize\":\"3068\",\"EXTERNAL\":\
"TRUE\",\"numFiles\":\"2\",\"transient_lastDdlTime\":\"1699947381\",\"metadata_location\":\"s3a:
//iceberg-bucket/example/table1/metadata/00001-e358db94-34f8-4a59-b5ca-
a17022edf957.metadata.json\",\"numFilesErasureCoded\":\"0\",\"table_type\":\"iceberg\"}]},\"12\"
:{\"str\":\"EXTERNAL_TABLE\"},\"15\":{\"tf\":0},\"17\":{\"str\":\"hive\"},\"18\":
{\"i32\":1},\"19\":{\"i64\":0},\"25\":{\"i64\":1}}"
}
```

Add partition

```
{
  "eventType": "ADD_PARTITION",
  "dbName": "partition_example",
  "tableName": "table1",
  "partitions": [
    {
      "country": "US"
    }
  ],
  "partitionListJson": [
    "{\"1\":{\"lst\":[\"str\",1,\"US\"]},\"2\":{\"str\":\"partition_example\"},\"3\":
{\"str\":\"table1\"},\"4\":{\"i32\":1700735087},\"5\":{\"i32\":0},\"6\":{\"rec\":{\"1\":
{\"lst\":[\"rec\",2,{\"1\":{\"str\":\"id\"},\"2\":{\"str\":\"int\"}},{\"1\":
{\"str\":\"name\"},\"2\":{\"str\":\"string\"}}]},\"2\":{\"str\":\"s3a://hive-bucket/
partition_example/table1/country=US\"},\"3\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat\"},\"5\":
{\"tf\":0},\"6\":{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe\"},\"3\":{\"map\":
[\"str\",\"str\",0,{}]}}},\"10\":{\"map\":[\"str\",\"str\",1,
{\"preferred_ordering_columns\":\"\"}]}}},\"7\":{\"map\":[\"str\",\"str\",8,
{\"presto_query_id\":\"20231123_102444_00028_9tgi3\",\"totalSize\":\"619\",\"numRows\":\"1\",\"r
awDataSize\":\"14\",\"numFiles\":\"1\",\"transient_lastDdlTime\":\"1700735087\",\"numFilesErasur
eCoded\":\"0\",\"presto_version\":\"0.282\"}]},\"9\":{\"str\":\"hive\"}}"
  ],
  "partitionList": [
    "{\"1\":{\"lst\":[\"str\",1,\"US\"]},\"2\":{\"str\":\"partition_example\"},\"3\":
{\"str\":\"table1\"},\"4\":{\"i32\":1700735087},\"5\":{\"i32\":0},\"6\":{\"rec\":{\"1\":
{\"lst\":[\"rec\",2,{\"1\":{\"str\":\"id\"},\"2\":{\"str\":\"int\"}},{\"1\":
{\"str\":\"name\"},\"2\":{\"str\":\"string\"}}]},\"2\":{\"str\":\"s3a://hive-bucket/
partition_example/table1/country=US\"},\"3\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat\"},\"5\":
{\"tf\":0},\"6\":{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe\"},\"3\":{\"map\":
[\"str\",\"str\",0,{}]}}},\"10\":{\"map\":[\"str\",\"str\",1,
{\"preferred_ordering_columns\":\"\"}]}}},\"7\":{\"map\":[\"str\",\"str\",8,
{\"presto_query_id\":\"20231123_102444_00028_9tgi3\",\"totalSize\":\"619\",\"numRows\":\"1\",\"r
awDataSize\":\"14\",\"numFiles\":\"1\",\"transient_lastDdlTime\":\"1700735087\",\"numFilesErasur
eCoded\":\"0\",\"presto_version\":\"0.282\"}]},\"9\":{\"str\":\"hive\"}}"
  ]
}
```

Alter partition

```
{
  "eventType": "ALTER_PARTITION",
  "dbName": "partition_example",
  "tableName": "table1",
  "oldPartition": "{\"1\":{\"lst\":[\"str\",1,\"US\"]},\"2\":
{\"str\":\"partition_example\"},\"3\":{\"str\":\"table1\"},\"4\":{\"i32\":1700735087},\"5\":
{\"i32\":0},\"6\":{\"rec\":{\"1\":{\"lst\":[\"rec\",2,{\"1\":{\"str\":\"id\"},\"2\":
{\"str\":\"int\"}},{\"1\":{\"str\":\"name\"},\"2\":{\"str\":\"string\"}}]},\"2\":
{\"str\":\"s3a://hive-bucket/partition_example/table1/country=US\"},\"3\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat\"},\"5\":
{\"tf\":0},\"6\":{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe\"},\"3\":{\"map\":
[\"str\",\"str\",0,{}]}}},\"8\":{\"lst\":[\"str\",0]},\"9\":{\"lst\":[\"rec\",0]},\"10\":
{\"map\":[\"str\",\"str\",1,{\"preferred_ordering_columns\":\"\"}]},\"11\":{\"rec\":{\"1\":
{\"lst\":[\"str\",0]},\"2\":{\"lst\":[\"lst\",0]},\"3\":{\"map\":[\"lst\",\"str\",0,
{}]}}},\"12\":{\"tf\":0}}},\"7\":{\"map\":[\"str\",\"str\",8,
{\"presto_query_id\":\"20231123_102444_00028_9tgi3\",\"totalSize\":\"619\",\"numRows\":\"1\",\"r
awDataSize\":\"14\",\"numFiles\":\"1\",\"transient_lastDdlTime\":\"1700735087\",\"numFilesErasur
eCoded\":\"0\",\"presto_version\":\"0.282\"}]},\"9\":{\"str\":\"hive\"},\"10\":{\"i64\":-1}}",
  "newPartition": "{\"1\":{\"lst\":[\"str\",1,\"US\"]},\"2\":
{\"str\":\"partition_example\"},\"3\":{\"str\":\"table1\"},\"4\":{\"i32\":1700735087},\"5\":
{\"i32\":0},\"6\":{\"rec\":{\"1\":{\"lst\":[\"rec\",2,{\"1\":{\"str\":\"id\"},\"2\":
{\"str\":\"int\"}},{\"1\":{\"str\":\"name\"},\"2\":{\"str\":\"string\"}}]},\"2\":
{\"str\":\"s3a://hive-bucket/partition_example/table1/country=US\"},\"3\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat\"},\"5\":
{\"tf\":0},\"6\":{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe\"},\"3\":{\"map\":
[\"str\",\"str\",0,{}]}}},\"8\":{\"lst\":[\"str\",0]},\"9\":{\"lst\":[\"rec\",0]},\"10\":
{\"map\":[\"str\",\"str\",1,{\"preferred_ordering_columns\":\"\"}]},\"11\":{\"rec\":{\"1\":
{\"lst\":[\"str\",0]},\"2\":{\"lst\":[\"lst\",0]},\"3\":{\"map\":[\"lst\",\"str\",0,
{}]}}},\"12\":{\"tf\":0}}},\"7\":{\"map\":[\"str\",\"str\",8,
{\"presto_query_id\":\"20231123_102444_00028_9tgi3\",\"totalSize\":\"1231\",\"numRows\":\"2\",\"
rawDataSize\":\"27\",\"numFiles\":\"2\",\"transient_lastDdlTime\":\"1700735168\",\"numFilesErasu
reCoded\":\"0\",\"presto_version\":\"0.282\"}]},\"9\":{\"str\":\"hive\"},\"10\":{\"i64\":-1}}",
  "keyValues": {
    "country": "US"
  }
}
```

Drop partition

```
{
  "eventType": "DROP_PARTITION",
  "dbName": "partition_example",
  "tableName": "table1",
  "partitions": [
    {
      "country": "US"
    }
  ],
  "partitionListJson": [
    "{\"1\":{\"lst\":[\"str\",1,\"US\"]},\"2\":{\"str\":\"partition_example\"},\"3\":
{\"str\":\"table1\"},\"4\":{\"i32\":1700734787},\"5\":{\"i32\":0},\"6\":{\"rec\":{\"1\":
{\"lst\":[\"rec\",2,{\"1\":{\"str\":\"id\"},\"2\":{\"str\":\"int\"}},{\"1\":
{\"str\":\"name\"},\"2\":{\"str\":\"string\"}}]},\"2\":{\"str\":\"s3a://hive-bucket/
partition_example/table1/country=US\"},\"3\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat\"},\"5\":
{\"tf\":0},\"6\":{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe\"},\"3\":{\"map\":
[\"str\",\"str\",0,{}]}}},\"8\":{\"lst\":[\"str\",0]},\"9\":{\"lst\":[\"rec\",0]},\"10\":
{\"map\":[\"str\",\"str\",1,{\"preferred_ordering_columns\":\"\"}]},\"11\":{\"rec\":{\"1\":
{\"lst\":[\"str\",0]},\"2\":{\"lst\":[\"lst\",0]},\"3\":{\"map\":[\"lst\",\"str\",0,
{}]}}},\"12\":{\"tf\":0}}},\"7\":{\"map\":[\"str\",\"str\",8,
{\"presto_query_id\":\"20231123_101944_00025_9tgi3\",\"totalSize\":\"619\",\"numRows\":\"1\",\"r
awDataSize\":\"14\",\"numFiles\":\"1\",\"transient_lastDdlTime\":\"1700734787\",\"numFilesErasur
eCoded\":\"0\",\"presto_version\":\"0.282\"}]},\"9\":{\"str\":\"hive\"},\"10\":{\"i64\":-1}}"
  ],
  "deleteData": true,
  "partitionList": [
    "{\"1\":{\"lst\":[\"str\",1,\"US\"]},\"2\":{\"str\":\"partition_example\"},\"3\":
{\"str\":\"table1\"},\"4\":{\"i32\":1700734787},\"5\":{\"i32\":0},\"6\":{\"rec\":{\"1\":
{\"lst\":[\"rec\",2,{\"1\":{\"str\":\"id\"},\"2\":{\"str\":\"int\"}},{\"1\":
{\"str\":\"name\"},\"2\":{\"str\":\"string\"}}]},\"2\":{\"str\":\"s3a://hive-bucket/
partition_example/table1/country=US\"},\"3\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat\"},\"4\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat\"},\"5\":
{\"tf\":0},\"6\":{\"i32\":0},\"7\":{\"rec\":{\"1\":{\"str\":\"table1\"},\"2\":
{\"str\":\"org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe\"},\"3\":{\"map\":
[\"str\",\"str\",0,{}]}}},\"8\":{\"lst\":[\"str\",0]},\"9\":{\"lst\":[\"rec\",0]},\"10\":
{\"map\":[\"str\",\"str\",1,{\"preferred_ordering_columns\":\"\"}]},\"11\":{\"rec\":{\"1\":
{\"lst\":[\"str\",0]},\"2\":{\"lst\":[\"lst\",0]},\"3\":{\"map\":[\"lst\",\"str\",0,
{}]}}},\"12\":{\"tf\":0}}},\"7\":{\"map\":[\"str\",\"str\",8,
{\"presto_query_id\":\"20231123_101944_00025_9tgi3\",\"totalSize\":\"619\",\"numRows\":\"1\",\"r
awDataSize\":\"14\",\"numFiles\":\"1\",\"transient_lastDdlTime\":\"1700734787\",\"numFilesErasur
eCoded\":\"0\",\"presto_version\":\"0.282\"}]},\"9\":{\"str\":\"hive\"},\"10\":{\"i64\":-1}}"
  ]
}
```

HIVE_JSON

Create schema

```
{
  "eventType": "CREATE_DATABASE",
  "db": {
    "name": "example",
    "description": null,
    "locationUri": "s3a://iceberg-bucket/example",
    "parameters": {},
    "privileges": null,
    "ownerName": "ibmlhadmin",
    "ownerType": "USER",
    "catalogName": "hive",
    "createTime": 1699948492,
    "managedLocationUri": null,
    "type": null,
    "connector_name": null,
    "remote_dbname": null,
    "setName": true,
    "setDescription": false,
    "setLocationUri": true,
    "parametersSize": 0,
    "setParameters": true,
    "setPrivileges": false,
    "setOwnerName": true,
    "setOwnerType": true,
    "setCatalogName": true,
    "setCreateTime": true,
    "setManagedLocationUri": false,
    "setType": false,
    "setConnector_name": false,
    "setRemote_dbname": false
  }
}
```

Drop schema

```
{
  "eventType": "DROP_DATABASE",
  "db": {
    "name": "example",
    "description": null,
    "locationUri": "s3a://iceberg-bucket/example",
    "parameters": {},
    "privileges": null,
    "ownerName": "ibmlhadmin",
    "ownerType": "USER",
    "catalogName": "hive",
    "createTime": 1699948492,
    "managedLocationUri": null,
    "type": "NATIVE",
    "connector_name": null,
    "remote_dbname": null,
    "setName": true,
    "setDescription": false,
    "setLocationUri": true,
    "parametersSize": 0,
    "setParameters": true,
    "setPrivileges": false,
    "setOwnerName": true,
    "setOwnerType": true,
    "setCatalogName": true,
    "setCreateTime": true,
    "setManagedLocationUri": false,
    "setType": true,
    "setConnector_name": false,
    "setRemote_dbname": false
  }
}
```

Create table

```
{
  "eventType": "CREATE_TABLE",
  "table": {
    "tableName": "table1",
    "dbName": "example",
    "owner": "ibmlhadmin",
    "createTime": 1699948569,
    "lastAccessTime": 0,
    "retention": 0,
    "sd": {
      "cols": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        }
      ],
      "location": "s3a://iceberg-bucket/example/table1",
      "inputFormat": "org.apache.hadoop.mapred.FileInputFormat",
      "outputFormat": "org.apache.hadoop.mapred.FileOutputFormat",
      "compressed": false,
      "numBuckets": 0,
      "serdeInfo": {
        "name": "table1",
        "serializationLib": "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
        "parameters": {},
        "description": null,
        "serializerClass": null,
        "deserializerClass": null,
        "serdeType": null,
        "setName": true,
        "setSerializationLib": true,
        "parametersSize": 0,
        "setParameters": true,
        "setDescription": false,
        "setSerializerClass": false,
        "setDeserializerClass": false,
        "setSerdeType": false
      },
      "bucketCols": null,
      "sortCols": null,
      "parameters": {},
      "skewedInfo": null,
      "storedAsSubDirectories": false,
      "colsSize": 2,
      "colsIterator": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        }
      ],
      "setCols": true,
      "setLocation": true,
      "setInputFormat": true,
      "setOutputFormat": true,
      "setCompressed": true,
      "setNumBuckets": true,
      "setSerdeInfo": true,
      "bucketColsSize": 0,
      "bucketColsIterator": null,
      "setBucketCols": false,
      "sortColsSize": 0,
      "sortColsIterator": null,
      "setSortCols": false,
      "parametersSize": 0,
      "setParameters": true,
      "setSkewedInfo": false,
      "setStoredAsSubDirectories": false
    },
    "partitionKeys": [],
    "parameters": {
      "totalSize": "4294",
      "EXTERNAL": "TRUE",
      "numFiles": "3",
      "metadata_location": "s3a://iceberg-bucket/example/table1/metadata/
00000-5814ea67-633c-4b4c-abe6-d85b40d68ea8.metadata.json",
      "transient_lastDdlTime": "1699948569",
      "numFilesErasureCoded": "0",
      "table_type": "iceberg"
    },
    "viewOriginalText": null,
    "viewExpandedText": null,
    "tableType": "EXTERNAL_TABLE",
    "privileges": {
      "userPrivileges": {
        "ibmlhadmin": [
          {
            "privilege": "select",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          },
          {
            "privilege": "insert",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          },
          {
            "privilege": "update",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          },
          {
            "privilege": "delete",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          }
        ]
      },
      "groupPrivileges": {},
      "rolePrivileges": {},
      "userPrivilegesSize": 1,
      "setUserPrivileges": true,
      "groupPrivilegesSize": 0,
      "setGroupPrivileges": true,
      "rolePrivilegesSize": 0,
      "setRolePrivileges": true
    },
    "temporary": false,
    "rewriteEnabled": false,
    "creationMetadata": null,
    "catName": "hive",
    "ownerType": "USER",
    "writeId": -1,
    "isStatsCompliant": false,
    "colStats": null,
      "accessType": 0,
      "requiredReadCapabilities": null,
      "requiredWriteCapabilities": null,
      "id": 11,
      "fileMetadata": null,
      "dictionary": null,
      "txnId": 0,
      "setTableName": true,
      "setDbName": true,
      "setOwner": true,
      "setCreateTime": true,
      "setLastAccessTime": true,
      "setRetention": true,
      "setSd": true,
      "partitionKeysSize": 0,
      "partitionKeysIterator": [],
      "setPartitionKeys": true,
      "parametersSize": 7,
      "setParameters": true,
      "setViewOriginalText": false,
      "setViewExpandedText": false,
      "setTableType": true,
      "setPrivileges": true,
      "setTemporary": false,
      "setRewriteEnabled": false,
      "setCreationMetadata": false,
      "setCatName": true,
      "setOwnerType": true,
      "setWriteId": false,
      "setIsStatsCompliant": false,
      "setColStats": false,
      "setAccessType": false,
      "requiredReadCapabilitiesSize": 0,
      "requiredReadCapabilitiesIterator": null,
      "setRequiredReadCapabilities": false,
      "requiredWriteCapabilitiesSize": 0,
      "requiredWriteCapabilitiesIterator": null,
      "setRequiredWriteCapabilities": false,
      "setId": true,
      "setFileMetadata": false,
      "setDictionary": false,
      "setTxnId": false
    }
  }
```

Alter table

```
  {
    "eventType": "ALTER_TABLE",
    "oldTable": {
      "tableName": "table1",
      "dbName": "example",
      "owner": "ibmlhadmin",
      "createTime": 1699948569,
      "lastAccessTime": 0,
      "retention": 0,
      "sd": {
        "cols": [
          {
            "name": "id",
            "type": "int",
            "comment": null,
            "setName": true,
            "setType": true,
            "setComment": false
          },
          {
            "name": "name",
            "type": "string",
            "comment": null,
            "setName": true,
            "setType": true,
            "setComment": false
          }
        ],
        "location": "s3a://iceberg-bucket/example/table1",
        "inputFormat": "org.apache.hadoop.mapred.FileInputFormat",
        "outputFormat": "org.apache.hadoop.mapred.FileOutputFormat",
        "compressed": false,
        "numBuckets": 0,
      "serdeInfo": {
        "name": "table1",
        "serializationLib": "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
        "parameters": {},
        "description": null,
        "serializerClass": null,
        "deserializerClass": null,
        "serdeType": null,
        "setName": true,
        "setSerializationLib": true,
        "parametersSize": 0,
        "setParameters": true,
        "setDescription": false,
        "setSerializerClass": false,
        "setDeserializerClass": false,
        "setSerdeType": false
      },
      "bucketCols": [],
      "sortCols": [],
      "parameters": {},
      "skewedInfo": {
        "skewedColNames": [],
        "skewedColValues": [],
        "skewedColValueLocationMaps": {},
        "skewedColNamesSize": 0,
        "skewedColNamesIterator": [],
        "setSkewedColNames": true,
        "skewedColValuesSize": 0,
        "skewedColValuesIterator": [],
        "setSkewedColValues": true,
        "skewedColValueLocationMapsSize": 0,
        "setSkewedColValueLocationMaps": true
      },
      "storedAsSubDirectories": false,
      "colsSize": 2,
      "colsIterator": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        }
      ],
      "setCols": true,
      "setLocation": true,
      "setInputFormat": true,
      "setOutputFormat": true,
      "setCompressed": true,
      "setNumBuckets": true,
      "setSerdeInfo": true,
      "bucketColsSize": 0,
      "bucketColsIterator": [],
      "setBucketCols": true,
      "sortColsSize": 0,
      "sortColsIterator": [],
      "setSortCols": true,
      "parametersSize": 0,
      "setParameters": true,
      "setSkewedInfo": true,
      "setStoredAsSubDirectories": true
    },
    "partitionKeys": [],
    "parameters": {
      "totalSize": "4294",
      "EXTERNAL": "TRUE",
      "numFiles": "3",
      "metadata_location": "s3a://iceberg-bucket/example/table1/metadata/
00000-5814ea67-633c-4b4c-abe6-d85b40d68ea8.metadata.json",
      "transient_lastDdlTime": "1699948569",
      "numFilesErasureCoded": "0",
      "table_type": "iceberg"
    },
    "viewOriginalText": null,
    "viewExpandedText": null,
    "tableType": "EXTERNAL_TABLE",
    "privileges": null,
    "temporary": false,
    "rewriteEnabled": false,
    "creationMetadata": null,
    "catName": "hive",
    "ownerType": "USER",
    "writeId": 0,
    "isStatsCompliant": false,
    "colStats": null,
    "accessType": 0,
    "requiredReadCapabilities": null,
    "requiredWriteCapabilities": null,
    "id": 11,
    "fileMetadata": null,
    "dictionary": null,
    "txnId": 0,
    "setTableName": true,
    "setDbName": true,
    "setOwner": true,
    "setCreateTime": true,
    "setLastAccessTime": true,
    "setRetention": true,
    "setSd": true,
    "partitionKeysSize": 0,
    "partitionKeysIterator": [],
    "setPartitionKeys": true,
    "parametersSize": 7,
    "setParameters": true,
    "setViewOriginalText": false,
    "setViewExpandedText": false,
    "setTableType": true,
    "setPrivileges": false,
    "setTemporary": false,
    "setRewriteEnabled": true,
    "setCreationMetadata": false,
    "setCatName": true,
    "setOwnerType": true,
    "setWriteId": true,
    "setIsStatsCompliant": false,
    "setColStats": false,
    "setAccessType": false,
    "requiredReadCapabilitiesSize": 0,
    "requiredReadCapabilitiesIterator": null,
    "setRequiredReadCapabilities": false,
    "requiredWriteCapabilitiesSize": 0,
    "requiredWriteCapabilitiesIterator": null,
    "setRequiredWriteCapabilities": false,
    "setId": true,
    "setFileMetadata": false,
    "setDictionary": false,
    "setTxnId": false
  },
  "newTable": {
    "tableName": "table1",
    "dbName": "example",
    "owner": "ibmlhadmin",
    "createTime": 0,
    "lastAccessTime": 0,
    "retention": 0,
    "sd": {
      "cols": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "age",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        }
      ],
      "location": "s3a://iceberg-bucket/example/table1",
      "inputFormat": "org.apache.hadoop.mapred.FileInputFormat",
      "outputFormat": "org.apache.hadoop.mapred.FileOutputFormat",
      "compressed": false,
      "numBuckets": 0,
      "serdeInfo": {
        "name": "table1",
        "serializationLib": "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
        "parameters": {},
        "description": null,
        "serializerClass": null,
        "deserializerClass": null,
        "serdeType": null,
        "setName": true,
        "setSerializationLib": true,
        "parametersSize": 0,
        "setParameters": true,
        "setDescription": false,
        "setSerializerClass": false,
        "setDeserializerClass": false,
        "setSerdeType": false
      },
      "bucketCols": null,
      "sortCols": null,
      "parameters": {},
      "skewedInfo": null,
      "storedAsSubDirectories": false,
      "colsSize": 3,
      "colsIterator": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "age",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        }
      ],
      "setCols": true,
      "setLocation": true,
      "setInputFormat": true,
      "setOutputFormat": true,
      "setCompressed": true,
      "setNumBuckets": true,
      "setSerdeInfo": true,
      "bucketColsSize": 0,
      "bucketColsIterator": null,
      "setBucketCols": false,
      "sortColsSize": 0,
      "sortColsIterator": null,
      "setSortCols": false,
      "parametersSize": 0,
      "setParameters": true,
      "setSkewedInfo": false,
      "setStoredAsSubDirectories": false
    },
    "partitionKeys": [],
    "parameters": {
      "previous_metadata_location": "s3a://iceberg-bucket/example/table1/metadata/
```

```
00000-5814ea67-633c-4b4c-abe6-d85b40d68ea8.metadata.json",
      "totalSize": "6136",
      "EXTERNAL": "TRUE",
      "numFiles": "4",
      "metadata_location": "s3a://iceberg-bucket/example/table1/metadata/
00001-9d5197c2-32ea-426b-ab71-d0051720c1a2.metadata.json",
      "transient_lastDdlTime": "1699948569",
      "numFilesErasureCoded": "0",
      "table_type": "iceberg"
    },
    "viewOriginalText": null,
    "viewExpandedText": null,
    "tableType": "EXTERNAL_TABLE",
    "privileges": {
      "userPrivileges": {
        "ibmlhadmin": [
          {
            "privilege": "select",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          },
          {
            "privilege": "insert",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          },
          {
            "privilege": "update",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          },
          {
            "privilege": "delete",
            "createTime": 0,
            "grantor": "ibmlhadmin",
            "grantorType": "USER",
            "grantOption": true,
            "setPrivilege": true,
            "setCreateTime": true,
            "setGrantor": true,
            "setGrantorType": true,
            "setGrantOption": true
          }
        ]
      },
      "groupPrivileges": {},
      "rolePrivileges": {},
      "userPrivilegesSize": 1,
      "setUserPrivileges": true,
      "groupPrivilegesSize": 0,
      "setGroupPrivileges": true,
      "rolePrivilegesSize": 0,
      "setRolePrivileges": true
    },
    "temporary": false,
    "rewriteEnabled": false,
    "creationMetadata": null,
    "catName": "hive",
    "ownerType": "USER",
    "writeId": -1,
    "isStatsCompliant": false,
    "colStats": null,
    "accessType": 0,
    "requiredReadCapabilities": null,
    "requiredWriteCapabilities": null,
    "id": 0,
    "fileMetadata": null,
    "dictionary": null,
    "txnId": 0,
    "setTableName": true,
    "setDbName": true,
    "setOwner": true,
    "setCreateTime": true,
    "setLastAccessTime": true,
    "setRetention": true,
    "setSd": true,
    "partitionKeysSize": 0,
    "partitionKeysIterator": [],
    "setPartitionKeys": true,
    "parametersSize": 8,
    "setParameters": true,
    "setViewOriginalText": false,
    "setViewExpandedText": false,
    "setTableType": true,
    "setPrivileges": true,
    "setTemporary": false,
    "setRewriteEnabled": false,
    "setCreationMetadata": false,
    "setCatName": true,
    "setOwnerType": true,
    "setWriteId": false,
    "setIsStatsCompliant": false,
    "setColStats": false,
    "setAccessType": false,
    "requiredReadCapabilitiesSize": 0,
    "requiredReadCapabilitiesIterator": null,
    "setRequiredReadCapabilities": false,
    "requiredWriteCapabilitiesSize": 0,
    "requiredWriteCapabilitiesIterator": null,
    "setRequiredWriteCapabilities": false,
    "setId": false,
    "setFileMetadata": false,
    "setDictionary": false,
    "setTxnId": false
  }
}
```

Drop table

```
{
  "eventType": "DROP_TABLE",
  "table": {
    "tableName": "table1",
    "dbName": "example",
    "owner": "ibmlhadmin",
    "createTime": 1699948569,
    "lastAccessTime": 0,
    "retention": 0,
    "sd": {
      "cols": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "age",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        }
      ],
      "location": "s3a://iceberg-bucket/example/table1",
      "inputFormat": "org.apache.hadoop.mapred.FileInputFormat",
      "outputFormat": "org.apache.hadoop.mapred.FileOutputFormat",
      "compressed": false,
      "numBuckets": 0,
      "serdeInfo": {
        "name": "table1",
        "serializationLib": "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
        "parameters": {},
        "description": null,
        "serializerClass": null,
        "deserializerClass": null,
        "serdeType": null,
        "setName": true,
        "setSerializationLib": true,
        "parametersSize": 0,
        "setParameters": true,
        "setDescription": false,
        "setSerializerClass": false,
        "setDeserializerClass": false,
        "setSerdeType": false
      },
      "bucketCols": [],
      "sortCols": [],
      "parameters": {},
      "skewedInfo": {
        "skewedColNames": [],
        "skewedColValues": [],
        "skewedColValueLocationMaps": {},
        "skewedColNamesSize": 0,
        "skewedColNamesIterator": [],
        "setSkewedColNames": true,
        "skewedColValuesSize": 0,
        "skewedColValuesIterator": [],
        "setSkewedColValues": true,
        "skewedColValueLocationMapsSize": 0,
        "setSkewedColValueLocationMaps": true
      },
      "storedAsSubDirectories": false,
      "colsSize": 3,
      "colsIterator": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "age",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        }
      ],
      "setCols": true,
      "setLocation": true,
      "setInputFormat": true,
      "setOutputFormat": true,
      "setCompressed": true,
      "setNumBuckets": true,
      "setSerdeInfo": true,
      "bucketColsSize": 0,
      "bucketColsIterator": [],
      "setBucketCols": true,
      "sortColsSize": 0,
      "sortColsIterator": [],
    "setSortCols": true,
    "parametersSize": 0,
    "setParameters": true,
    "setSkewedInfo": true,
    "setStoredAsSubDirectories": true
  },
  "partitionKeys": [],
  "parameters": {
    "previous_metadata_location": "s3a://iceberg-bucket/example/table1/metadata/
00000-5814ea67-633c-4b4c-abe6-d85b40d68ea8.metadata.json",
    "totalSize": "6136",
    "EXTERNAL": "TRUE",
    "numFiles": "4",
    "metadata_location": "s3a://iceberg-bucket/example/table1/metadata/
00001-9d5197c2-32ea-426b-ab71-d0051720c1a2.metadata.json",
    "transient_lastDdlTime": "1699948569",
    "numFilesErasureCoded": "0",
    "table_type": "iceberg"
  },
  "viewOriginalText": null,
  "viewExpandedText": null,
  "tableType": "EXTERNAL_TABLE",
  "privileges": null,
  "temporary": false,
  "rewriteEnabled": false,
  "creationMetadata": null,
  "catName": "hive",
  "ownerType": "USER",
  "writeId": 0,
  "isStatsCompliant": false,
  "colStats": null,
  "accessType": 0,
  "requiredReadCapabilities": null,
  "requiredWriteCapabilities": null,
  "id": 11,
  "fileMetadata": null,
  "dictionary": null,
  "txnId": 0,
  "setTableName": true,
  "setDbName": true,
  "setOwner": true,
  "setCreateTime": true,
  "setLastAccessTime": true,
  "setRetention": true,
  "setSd": true,
  "partitionKeysSize": 0,
  "partitionKeysIterator": [],
  "setPartitionKeys": true,
  "parametersSize": 8,
  "setParameters": true,
  "setViewOriginalText": false,
  "setViewExpandedText": false,
  "setTableType": true,
  "setPrivileges": false,
  "setTemporary": false,
  "setRewriteEnabled": true,
  "setCreationMetadata": false,
  "setCatName": true,
  "setOwnerType": true,
  "setWriteId": true,
  "setIsStatsCompliant": false,
  "setColStats": false,
  "setAccessType": false,
  "requiredReadCapabilitiesSize": 0,
  "requiredReadCapabilitiesIterator": null,
  "setRequiredReadCapabilities": false,
  "requiredWriteCapabilitiesSize": 0,
  "requiredWriteCapabilitiesIterator": null,
  "setRequiredWriteCapabilities": false,
  "setId": true,
  "setFileMetadata": false,
  "setDictionary": false,
  "setTxnId": false
}
}
```

Add partition

```
{
  "eventType": "ADD_PARTITION",
    "partitions": [
      {
        "country": "US"
      }
    ],
    "partitionList": [
      {
        "values": [
          "US"
        ],
        "dbName": "partition_example",
        "tableName": "table1",
        "createTime": 1700735507,
        "lastAccessTime": 0,
        "sd": {
          "cols": [
            {
              "name": "id",
              "type": "int",
              "comment": null,
              "setName": true,
              "setType": true,
              "setComment": false
            },
            {
              "name": "name",
              "type": "string",
              "comment": null,
              "setName": true,
              "setType": true,
              "setComment": false
            }
          ],
          "location": "s3a://hive-bucket/partition_example/table1/country=US",
          "inputFormat": "org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat",
          "outputFormat": "org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat",
          "compressed": false,
          "numBuckets": 0,
          "serdeInfo": {
            "name": "table1",
            "serializationLib": "org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe",
            "parameters": {},
            "description": null,
            "serializerClass": null,
            "deserializerClass": null,
            "serdeType": null,
            "setName": true,
            "setSerializationLib": true,
            "parametersSize": 0,
            "setParameters": true,
            "setDescription": false,
            "setSerializerClass": false,
            "setDeserializerClass": false,
            "setSerdeType": false
          },
          "bucketCols": null,
          "sortCols": null,
          "parameters": {
            "preferred_ordering_columns": ""
          },
          "skewedInfo": null,
          "storedAsSubDirectories": false,
          "colsSize": 2,
          "colsIterator": [
            {
              "name": "id",
              "type": "int",
              "comment": null,
              "setName": true,
              "setType": true,
              "setComment": false
            },
            {
              "name": "name",
              "type": "string",
              "comment": null,
              "setName": true,
              "setType": true,
              "setComment": false
            }
          ],
          "setCols": true,
          "setLocation": true,
          "setInputFormat": true,
          "setOutputFormat": true,
          "setCompressed": true,
          "setNumBuckets": true,
          "setSerdeInfo": true,
          "bucketColsSize": 0,
          "bucketColsIterator": null,
          "setBucketCols": false,
          "sortColsSize": 0,
          "sortColsIterator": null,
          "setSortCols": false,
          "parametersSize": 1,
          "setParameters": true,
          "setSkewedInfo": false,
          "setStoredAsSubDirectories": false
        },
        "parameters": {
          "presto_query_id": "20231123_103145_00032_9tgi3",
          "totalSize": "619",
          "numRows": "1",
          "rawDataSize": "14",
          "numFiles": "1",
          "transient_lastDdlTime": "1700735507",
          "numFilesErasureCoded": "0",
          "presto_version": "0.282"
        },
        "privileges": null,
        "catName": "hive",
        "writeId": -1,
        "isStatsCompliant": false,
        "colStats": null,
        "fileMetadata": null,
        "valuesSize": 1,
        "valuesIterator": [
          "US"
        ],
        "setValues": true,
        "setDbName": true,
        "setTableName": true,
        "setCreateTime": true,
        "setLastAccessTime": true,
        "setSd": true,
        "parametersSize": 8,
        "setParameters": true,
        "setPrivileges": false,
        "setCatName": true,
        "setWriteId": false,
        "setIsStatsCompliant": false,
        "setColStats": false,
        "setFileMetadata": false
      }
    ]
  }
```

Alter partition

```
{
  "eventType": "ALTER_PARTITION",
  "oldPartition": {
    "values": [
      "US"
    ],
    "dbName": "partition_example",
    "tableName": "table1",
    "createTime": 1700735507,
    "lastAccessTime": 0,
    "sd": {
      "cols": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
            "comment": null,
            "setName": true,
            "setType": true,
            "setComment": false
          }
        ],
        "location": "s3a://hive-bucket/partition_example/table1/country=US",
        "inputFormat": "org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat",
        "outputFormat": "org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat",
        "compressed": false,
        "numBuckets": 0,
        "serdeInfo": {
          "name": "table1",
          "serializationLib": "org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe",
          "parameters": {},
          "description": null,
          "serializerClass": null,
          "deserializerClass": null,
          "serdeType": null,
          "setName": true,
          "setSerializationLib": true,
          "parametersSize": 0,
          "setParameters": true,
          "setDescription": false,
          "setSerializerClass": false,
          "setDeserializerClass": false,
          "setSerdeType": false
        },
        "bucketCols": [],
        "sortCols": [],
        "parameters": {
          "preferred_ordering_columns": ""
        },
        "skewedInfo": {
          "skewedColNames": [],
          "skewedColValues": [],
          "skewedColValueLocationMaps": {},
          "skewedColNamesSize": 0,
          "skewedColNamesIterator": [],
          "setSkewedColNames": true,
          "skewedColValuesSize": 0,
          "skewedColValuesIterator": [],
          "setSkewedColValues": true,
          "skewedColValueLocationMapsSize": 0,
          "setSkewedColValueLocationMaps": true
        },
        "storedAsSubDirectories": false,
        "colsSize": 2,
        "colsIterator": [
          {
            "name": "id",
            "type": "int",
            "comment": null,
            "setName": true,
            "setType": true,
            "setComment": false
          },
          {
            "name": "name",
            "type": "string",
            "comment": null,
            "setName": true,
            "setType": true,
            "setComment": false
          }
        ],
        "setCols": true,
        "setLocation": true,
        "setInputFormat": true,
        "setOutputFormat": true,
        "setCompressed": true,
        "setNumBuckets": true,
        "setSerdeInfo": true,
        "bucketColsSize": 0,
        "bucketColsIterator": [],
        "setBucketCols": true,
        "sortColsSize": 0,
        "sortColsIterator": [],
        "setSortCols": true,
        "parametersSize": 1,
        "setParameters": true,
        "setSkewedInfo": true,
        "setStoredAsSubDirectories": true
      },
      "parameters": {
        "presto_query_id": "20231123_103145_00032_9tgi3",
        "totalSize": "619",
        "numRows": "1",
        "rawDataSize": "14",
        "numFiles": "1",
        "transient_lastDdlTime": "1700735507",
        "numFilesErasureCoded": "0",
        "presto_version": "0.282"
      },
      "privileges": null,
      "catName": "hive",
      "writeId": -1,
      "isStatsCompliant": false,
      "colStats": null,
      "fileMetadata": null,
      "valuesSize": 1,
      "valuesIterator": [
        "US"
      ],
      "setValues": true,
      "setDbName": true,
      "setTableName": true,
      "setCreateTime": true,
      "setLastAccessTime": true,
      "setSd": true,
      "parametersSize": 8,
      "setParameters": true,
      "setPrivileges": false,
      "setCatName": true,
      "setWriteId": true,
      "setIsStatsCompliant": false,
      "setColStats": false,
      "setFileMetadata": false
    },
    "newPartition": {
      "values": [
        "US"
      ],
      "dbName": "partition_example",
      "tableName": "table1",
      "createTime": 1700735507,
      "lastAccessTime": 0,
      "sd": {
        "cols": [
          {
            "name": "id",
            "type": "int",
            "comment": null,
            "setName": true,
            "setType": true,
            "setComment": false
          },
          {
            "name": "name",
            "type": "string",
            "comment": null,
            "setName": true,
            "setType": true,
            "setComment": false
          }
        ],
        "location": "s3a://hive-bucket/partition_example/table1/country=US",
        "inputFormat": "org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat",
        "outputFormat": "org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat",
        "compressed": false,
        "numBuckets": 0,
        "serdeInfo": {
          "name": "table1",
          "serializationLib": "org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe",
          "parameters": {},
          "description": null,
          "serializerClass": null,
          "deserializerClass": null,
          "serdeType": null,
          "setName": true,
          "setSerializationLib": true,
          "parametersSize": 0,
          "setParameters": true,
          "setDescription": false,
        "setSerializerClass": false,
        "setDeserializerClass": false,
        "setSerdeType": false
      },
      "bucketCols": [],
      "sortCols": [],
      "parameters": {
        "preferred_ordering_columns": ""
      },
      "skewedInfo": {
        "skewedColNames": [],
        "skewedColValues": [],
        "skewedColValueLocationMaps": {},
        "skewedColNamesSize": 0,
        "skewedColNamesIterator": [],
        "setSkewedColNames": true,
        "skewedColValuesSize": 0,
        "skewedColValuesIterator": [],
        "setSkewedColValues": true,
        "skewedColValueLocationMapsSize": 0,
        "setSkewedColValueLocationMaps": true
      },
      "storedAsSubDirectories": false,
      "colsSize": 2,
      "colsIterator": [
        {
          "name": "id",
          "type": "int",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        },
        {
          "name": "name",
          "type": "string",
          "comment": null,
          "setName": true,
          "setType": true,
          "setComment": false
        }
      ],
      "setCols": true,
      "setLocation": true,
      "setInputFormat": true,
      "setOutputFormat": true,
      "setCompressed": true,
      "setNumBuckets": true,
      "setSerdeInfo": true,
      "bucketColsSize": 0,
      "bucketColsIterator": [],
      "setBucketCols": true,
      "sortColsSize": 0,
      "sortColsIterator": [],
      "setSortCols": true,
      "parametersSize": 1,
      "setParameters": true,
      "setSkewedInfo": true,
      "setStoredAsSubDirectories": true
    },
    "parameters": {
      "presto_query_id": "20231123_103145_00032_9tgi3",
      "totalSize": "1231",
      "numRows": "2",
      "rawDataSize": "27",
      "numFiles": "2",
      "transient_lastDdlTime": "1700735602",
      "numFilesErasureCoded": "0",
      "presto_version": "0.282"
    },
    "privileges": null,
    "catName": "hive",
    "writeId": -1,
    "isStatsCompliant": false,
    "colStats": null,
    "fileMetadata": null,
    "valuesSize": 1,
    "valuesIterator": [
      "US"
    ],
    "setValues": true,
    "setDbName": true,
     "setTableName": true,
     "setCreateTime": true,
     "setLastAccessTime": true,
     "setSd": true,
     "parametersSize": 8,
     "setParameters": true,
     "setPrivileges": false,
     "setCatName": true,
     "setWriteId": true,
     "setIsStatsCompliant": false,
     "setColStats": false,
     "setFileMetadata": false
   }
 }
```

Drop partition

```
 {
   "eventType": "DROP_PARTITION",
   "partitions": [
     {
       "country": "US"
     }
   ],
   "partitionList": [
     {
       "values": [
         "US"
       ],
       "dbName": "partition_example",
       "tableName": "table1",
       "createTime": 1700735507,
       "lastAccessTime": 0,
       "sd": {
         "cols": [
           {
             "name": "id",
             "type": "int",
             "comment": null,
             "setName": true,
             "setType": true,
             "setComment": false
           },
           {
             "name": "name",
             "type": "string",
             "comment": null,
             "setName": true,
             "setType": true,
             "setComment": false
           }
         ],
         "location": "s3a://hive-bucket/partition_example/table1/country=US",
         "inputFormat": "org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat",
         "outputFormat": "org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat",
         "compressed": false,
         "numBuckets": 0,
         "serdeInfo": {
           "name": "table1",
           "serializationLib": "org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe",
           "parameters": {},
           "description": null,
           "serializerClass": null,
           "deserializerClass": null,
           "serdeType": null,
           "setName": true,
           "setSerializationLib": true,
           "parametersSize": 0,
           "setParameters": true,
           "setDescription": false,
           "setSerializerClass": false,
           "setDeserializerClass": false,
           "setSerdeType": false
         },
         "bucketCols": [],
         "sortCols": [],
         "parameters": {
           "preferred_ordering_columns": ""
         },
         "skewedInfo": {
          "skewedColNames": [],
          "skewedColValues": [],
          "skewedColValueLocationMaps": {},
          "skewedColNamesSize": 0,
          "skewedColNamesIterator": [],
          "setSkewedColNames": true,
          "skewedColValuesSize": 0,
          "skewedColValuesIterator": [],
          "setSkewedColValues": true,
          "skewedColValueLocationMapsSize": 0,
          "setSkewedColValueLocationMaps": true
        },
        "storedAsSubDirectories": false,
        "colsSize": 2,
        "colsIterator": [
          {
            "name": "id",
            "type": "int",
            "comment": null,
            "setName": true,
            "setType": true,
            "setComment": false
          },
          {
            "name": "name",
            "type": "string",
            "comment": null,
            "setName": true,
            "setType": true,
            "setComment": false
          }
        ],
        "setCols": true,
        "setLocation": true,
        "setInputFormat": true,
        "setOutputFormat": true,
        "setCompressed": true,
        "setNumBuckets": true,
        "setSerdeInfo": true,
        "bucketColsSize": 0,
        "bucketColsIterator": [],
        "setBucketCols": true,
        "sortColsSize": 0,
        "sortColsIterator": [],
        "setSortCols": true,
        "parametersSize": 1,
        "setParameters": true,
        "setSkewedInfo": true,
        "setStoredAsSubDirectories": true
      },
      "parameters": {
        "presto_query_id": "20231123_103145_00032_9tgi3",
        "totalSize": "1231",
        "numRows": "2",
        "rawDataSize": "27",
        "numFiles": "2",
        "transient_lastDdlTime": "1700735602",
        "numFilesErasureCoded": "0",
        "presto_version": "0.282"
      },
      "privileges": null,
      "catName": "hive",
      "writeId": -1,
      "isStatsCompliant": false,
      "colStats": null,
      "fileMetadata": null,
      "valuesSize": 1,
      "valuesIterator": [
        "US"
      ],
      "setValues": true,
      "setDbName": true,
      "setTableName": true,
      "setCreateTime": true,
      "setLastAccessTime": true,
      "setSd": true,
      "parametersSize": 8,
      "setParameters": true,
      "setPrivileges": false,
      "setCatName": true,
      "setWriteId": true,
      "setIsStatsCompliant": false,
        "setColStats": false,
        "setFileMetadata": false
      }
    ],
    "deleteData": true
}
```

Syncing external Iceberg data into watsonx.data

You can write data to object store by using Spark and other external tools like Snowflake in Iceberg table format that is outside watsonx.data. You can sync the object store (bucket) metadata with watsonx.data without moving the data physically.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. In **Infrastructure Manager**, click **Add component**.
2. Click **Add bucket**.
3. Enter the bucket details.
4. Select **Activate now**.
5. Select **Catalog type** as **Apache Iceberg**.
6. Enter the catalog name.
7. Select all the data present on the bucket to sync or create a catalog without any preexisting data.
8. After registration, click the catalog and go to **Sync logs**. You can see the status of synchronization (success, failure, partial success) and the last sync time.
9. After completing the synchronization, go to the **Data manager**. You can see the catalog that you created and the tables that are pulled from the bucket created by Snowflake.

   Note: You can use SQL editor (**Query workspace**) and use these tables to select query and insert data into existing table.
10. If Snowflake user makes data changes in the bucket and watsonx.data wants to pull those changes in the bucket, then click **Sync data** from UI for three strategy options:
    a. **Sync all data:** Synchronize all the data or update the existing table that was promoted earlier.
    b. **Sync new data only:** Pull the newly created table only.
    c. **Sync existing data only:** Update the table registered earlier to the latest changes only.

Chapter 7. Connecting to a Presto server

Presto CLI provides a terminal-based interactive shell to run queries. You can connect to the Presto server either through Presto CLI installed as part of the ibm-lh-client package or through Presto CLI installed separately.

watsonx.data on Red Hat OpenShift

To connect to a Presto Server from a client program or CLI, the following items are required:

- Hostname and port for the Presto server or workstation where the IBM watsonx.data developer or software is installed.
- Certificates served by the Presto server to establish trust.
- Authorized user credentials to access the Presto server.

Note: It is important to connect to the Presto server that uses its hostname and not its IP address. Because the TLS certificate that is served by the Presto Server is associated with a fully qualified host and domain-name (FQDN). Client programs, typically, cannot establish trust by using the IP address. With OpenShift Ingress in particular, DNS entries, based on hostnames, play an important role in routing to the intended Kubernetes Service.

Tip: To confirm there is network access from your client workstation that needs to connect a Presto server you can test the access by using one of the following commands:

```
 curl -ki https://<presto-hostname>:<presto-portnumber>
```

```
nc -v <presto-hostname> <presto-portnumber>
```

- The Presto server exposes a HTTPs (TCP) port. Therefore, you can use any convenient HTTP or TCP-based utility to ensure that the Presto server can be accessed from your network.
- When you run the `curl` command, the server may return an **HTTP/1.1 401 Unauthorized response**. This is expected as the server is secured by authentication.
- When you run the nc command, the server returns a success message.

In watsonx.data, you can connect to the Presto server in multiple ways based on the platform and utilities you are using. See the following sections for more details:

- Using built-in presto-cli in the developer edition
- Using presto-cli and presto-run in the ibm-lh-client package
- Using presto-cli executable (remote) – Developer
- Using presto-cli executable (remote) - software
- Using Java/JDBC – Developer
- Using Java/JDBC - software
- Using Python scripts – Developer
- Using Python scripts – software

Important: You must specify the location when creating schema by using CLI. For example,

```
location = s3a://<storage-name>/
```

Using built-in presto-cli in the developer edition

To connect to the built-in presto-cli in the developer edition:

1. Open a command prompt in the workstation where IBM watsonx.data developer is installed.
2. Go to the `/bin` folder of the installation directory and run the presto-cli.sh command.

3. Inspect the available catalogs in the watsonx.data.

```
presto> show catalogs;
    Catalog
---------------
 iceberg_minio
 jmx
 system
 tpcds
 tpch
(5 rows)

Query 20230308_195425_00000_hy4cs, FINISHED, 1 node
Splits: 19 total, 19 done (100.00%)
0:02 [0 rows, 0B] [0 rows/s, 0B/s]
```

Using presto-cli and presto-run in the ibm-lh-client package

For details and instructions that are related to using presto-cli and presto-run in the ibm-lh-client package, see Installing `ibm-lh-client` .

Using presto-cli executable (remote) – Developer edition

This section provides instructions to connect the Presto server in watsonx.data developer edition from a remote `presto-cli`.

Note:

You can download the presto-cli executable from Install the Presto CLI.

1. Get the hostname and port of the server or workstation where the watsonx.data developer package is installed.
2. Import the certificate of the Presto server to establish trust. You can import the certificate either by using the `cert-mgmt` utility available in a watsonx.data client installation or manually. Based on your setup, complete one of the following steps:
   a. From your client machine, point or copy the truststore or certificate file present in the `ibm-lh-client` installation directory. The certificates are at `ibm-lh-client/localstorage/volumes/infra/tls/`. In this location, you can find:
      - `truststore.jks` - can be used for Java/JDBC based applications.
      - `cabundle.crt` - can be used for non-Java applications.
      - aliases/ - is a subdirectory with individual certificates. For example, aliases/<presto server hostname>: 8443.crt refers to the certificate for a specific Presto server only.

      For example, `bin/cert-mgmt --op=add --host=<hostname of the Presto server> --port=8443`
   b. Manually import certificates into the truststore. For instructions, see Importing self-signed certificates from a Presto server to a Java truststore.
3. Authenticate with the Presto server.

```
./presto --server <https://hostname:port> --keystore-path /<directory>/<certificate
file>.jks --keystore-password <keystore password> --truststore-path /<directory>/lh-ssl-
ks.jks --truststore-password <truststore password> --catalog iceberg_data --user <username>
--password <password>
```

   Note: The default credential for watsonx.data developer edition username - ibmlhadmin with password - password. However, if a different password was used while installing the developer edition or later changed by using the `ibm-lh-dev/bin/user-mgmt change-password` command, use the appropriate username and password.

   The `ibm-lh-dev/bin/user-mgmt` utility can be used to add additional users and `ibm-lh-dev/bin/role-mgmt` can be used to grant such users either an Admin or User Role.

Provide the credentials values when connecting to the Presto server in the developer installation from a client. No API Keys or tokens are required or available.

Using presto-cli executable (remote) - software

This section provides instructions to connect the Presto server in watsonx.data software edition from a remote `presto-cli`.

Before you get started, ensure the following:

- The Red Hat OpenShift Project namespace administrator or cluster administrator has exposed a route to the Presto (Java) server. The administrators should refer to Exposing secure route to Presto server for instructions.
- You have received the hostname and port (by default 443) associated with the specific Red Hat OpenShift route from the administrator.

1. Import the certificate of the Presto server to establish trust. You can import the certificate either by using the cert-mgmt utility or manually. Based on your setup, complete one of the following steps:

   Note: Presto servers in watsonx.data use TLS for securing access from clients. Sometimes, the Presto server may be associated with a Self-signed certificate instead of a certificate authority (CA) signed certificate. In such cases, the client programs that connect via the HTTPs (TLS) protocol must be configured to trust the self-signed certificate configured in the Presto Server.

   a. If you are connecting from the watsonx.data utilities, go to `bin/cert-mgmt` and run the following command:

   ```
   bin/cert-mgmt --op=add --host=<hostname of the Presto server> --port=443
   ```

   You see the following output:

   ```
   /mnt/infra/tls/truststore.jks updated
       /mnt/infra/tls/cabundle.crt updated
   ```

   Note: The ibm-lh-client installation includes utilities and packages to connect to Presto engines in watsonx.data software deployments. The client installation includes the `bin/cert-mgmt` executable that simplifies fetching the certificate that is used with a Presto server. It stores such certificates in a Java (JKS) format truststore for Java based clients and also in PEM format for use with nonjava clients.

   The presto-cli and presto-run watsonx.data (ibm-lh-client) utilities are Java-based and automatically use this JKS truststore. JDBC programs (such as those run via the `bin/dev-sandbox` utility) must be configured to explicitly use this JKS truststore. The `cabundle.crt` must be used by non-Java programs, such as Python scripts.

   b. If you are connecting from programs or utilities that run outside the watsonx.data client installation, do one of the following:

   i. From your client machine, point or copy the truststore or certificate file present in the ibm-lh-client installation directory. The certificates are at `ibm-lh-client/localstorage/volumes/infra/tls/`. In this location, you can find:

   `truststore.jks`- can be used for Java/JDBC based applications.

   `cabundle.crt` - can be used for non-Java applications.

   `aliases/` - is a subdirectory with individual certificates. For example, `aliases/<Presto server hostname>:443.crt` refers to the certificate for a specific Presto server only.

   ii. Manually import certificates into the truststore. For instructions, see Importing self-signed certificates from a Presto server to a Java truststore.

2. Authenticate with the Presto server.

   ```
   ./presto --server https://<route_name> --truststore-path /<directory>/<certfilename/
   truststore>.jks --truststore-password changeit --catalog iceberg_data --user admin --password
   ```

   Note:

   - The default out-of-the-box user for watsonx.data software is ‘admin’ with a generated password that is accessible by the ibm-lakehouse-manage get-cpd-instance-details command after installation. See Installing the software watsonx.data for more details.
   - `-–truststore-path` and `-–truststore-password` arguments are optional if the certificates were imported into the default JDK/JRE trust store already

   The Administrator of the watsonx.data instance can configure different Identity Providers to authenticate users or grant them appropriate Admin or User Roles. See the “Managing Users” topic for more details.

   In addition to directly using their credentials as username and password properties to connect to the Presto server, users can choose an API Key as well. See Generating API keys for authentication for details on how to generate such API Keys after signing in to watsonx.data from a browser. See, Generating API keys for authentication for details for how to use API Keys or Tokens from client applications instead of credentials.
3. When connecting to the watsonx.data Presto servers, the credential properties to set would then be:

   ```
   username: username, password: <password>
   ```

   Or

   ```
   username: ibmlhapikey, password:<base64 of username:apikey>
   ```

   Or

   ```
   username: ibmlhtoken, password:<platform token> or < instance token >
   ```

Using Java/JDBC – Developer edition

This section provides instructions to connect to a Presto server from a Java JDBC application.

Before you get started, get the hostname and port of the server or workstation where the watsonx.data developer package is installed.

Note: The default Presto port number for the developer edition is 8443. However, if a different port number was picked during the installation, use the appropriate port number.

To connect with watsonx.data Presto server from a Java JDBC appliation, complete the following steps:

1. If required, complete the instructions in Using presto-cli executable (remote) – Developer to get the hostname, port, server certificate, authentication, and test the connectivity.
2. Download and install the `presto-jdbc-0.286.jar` (or later) on the client machine.
3. Add `presto-jdbc-0.286.jar` (or later) to the class path of your Java application.
4. Create a Java application by using a JDBC interface. Following is a sample JDBC code.

   Note:

   - presto_url - Identifies the jdbc URL to the Presto server
   - `SSLTrustStorePath` and `SSLTrustStorePassword` properties are set to identify the location of the truststore and its password (if necessary).
   - Values for user and password properties are the watsonx.data developer edition username and password.

Example: Java snippet

```
import java.sql.*;
import java.util.Properties;

public class PrestoJdbcSample
{

    public static void main(String[] args) throws Exception
    {

        /**
         * example of fetching the location and credentials needed to connect, from 
environment variables
         **/
        String username = System.getenv("ENG_USERNAME");
        String password = System.getenv("ENG_PASSWORD");
        String hostname = System.getenv("ENG_HOST");
        String portnumber = System.getenv("ENG_PORT");

        String presto_url = "jdbc:presto://" + hostname + ":" + portnumber;

        String ts_location =  System.getenv("TRUSTSTORE");
            // example: “ibm-lh-client/localstorage/volumes/infra/tls/truststore.jks";

        String ts_password =  System.getenv("TRUSTSTORE_PASSWORD");
        // example: "changeit";

        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try
        {
            /** load the Presto JDBC Driver class **/
            String driverClass = "com.facebook.presto.jdbc.PrestoDriver";
            Class.forName(driverClass);

            /** Set the connection properties**/
            Properties properties = new Properties();
            properties.setProperty("user", username);
            properties.setProperty("password", password);
            properties.setProperty("SSL", "true");

            /**
             * identify where the java truststore is located.
             * skip if the default JDK truststore already has the Presto Server's certificate
             * or if the Presto server has signed certificates
             **/
            properties.setProperty("SSLTrustStorePath", ts_location);
            properties.setProperty("SSLTrustStorePassword", ts_password);

            /** Connect **/
            connection = DriverManager.getConnection(presto_url, properties);
            /** Issue a Query **/
            String query = "SELECT * FROM tpch.tiny.customer LIMIT 10";
            statement = connection.createStatement();
            resultSet = statement.executeQuery(query);

            /** iterate through the results **/
            while (resultSet.next())
            {
                String phone = resultSet.getString("phone");
                String name = resultSet.getString("name");
                System.out.println("phone = " + phone + ", name = " + name);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            /** clean up at the end always **/
            if (resultSet != null)
            {
                resultSet.close();
            }
            if (statement != null)
            {
                statement.close();
            }
            if (connection != null)
            {
                connection.close();
            }
        }

    }
}
```

Using Java/JDBC – software

This section provides instructions to connect to a Presto server from a Java JDBC application.

Before you get started, ensure the following:

- the Red Hat OpenShift Project namespace administrator or cluster administrator has exposed the route to the Presto server. The administrators should refer to Exposing secure route to Presto server for instructions.
- you have the received the hostname and port (by default 443) associated with that specific Red Hat OpenShift route from the administrator.

To connect with watsonx.data Presto server from a Java JDBC application, complete the following steps:

1. If required, complete the instructions in Using presto-cli executable (remote) - software to get the hostname, port, server certificate, authentication, and test the connectivity.
2. Download and install the `presto-jdbc-0.286.jar` (or later) on the client machine.
3. Add `presto-jdbc-0.286.jar` (or later) to the class path of your Java application.
4. Create a Java application by using JDBC interface. Following is a sample JDBC code.

   Note:

   - presto_url - Identifies the jdbc URL to the Presto server
   - `SSLTrustStorePath` and `SSLTrustStorePassword` properties are set to identify the location of the truststore and its password (if necessary).
   - Values for user and password properties are the watsonx.data software edition username and password.

```
import java.sql.*;
import java.util.Properties;

public class PrestoJdbcSample
{

    public static void main(String[] args) throws Exception
    {

        /**
         * example of fetching the location and credentials needed to connect, from
environment variables
         **/
        String username = System.getenv("ENG_USERNAME");
        String password = System.getenv("ENG_PASSWORD");
        String hostname = System.getenv("ENG_HOST");
        String portnumber = System.getenv("ENG_PORT");

        String presto_url = "jdbc:presto://" + hostname + ":" + portnumber;

        String ts_location =  System.getenv("TRUSTSTORE");
            // example: “ibm-lh-client/localstorage/volumes/infra/tls/truststore.jks";

        String ts_password =  System.getenv("TRUSTSTORE_PASSWORD");
        // example: "changeit";

        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try
        {
            /** load the Presto JDBC Driver class **/
            String driverClass = "com.facebook.presto.jdbc.PrestoDriver";
            Class.forName(driverClass);

            /** Set the connection properties**/
            Properties properties = new Properties();
            properties.setProperty("user", username);
            properties.setProperty("password", password);
            properties.setProperty("SSL", "true");

            /**
             * identify where the java truststore is located.
             * skip if the default JDK truststore already has the Presto Server's certificate
             * or if the Presto server has signed certificates
             **/
            properties.setProperty("SSLTrustStorePath", ts_location);
            properties.setProperty("SSLTrustStorePassword", ts_password);

            /** Connect **/
            connection = DriverManager.getConnection(presto_url, properties);
            /** Issue a Query **/
            String query = "SELECT * FROM tpch.tiny.customer LIMIT 10";
            statement = connection.createStatement();
            resultSet = statement.executeQuery(query);

            /** iterate through the results **/
            while (resultSet.next())
            {
                String phone = resultSet.getString("phone");
                String name = resultSet.getString("name");
                System.out.println("phone = " + phone + ", name = " + name);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            /** clean up at the end always **/
            if (resultSet != null)
            {
                resultSet.close();
            }
            if (statement != null)
            {
                statement.close();
            }
            if (connection != null)
            {
                connection.close();
            }
        }

    }
}
```

Using Python scripts – Developer edition

This section provides instructions to connect to a Presto server from a Python client.

Before you get started, get the hostname and port of the server or workstation where the watsonx.data developer package is installed.

Note: The default Presto port number for the developer edition is 8443. However, if a different port number was picked during the installation, use the appropriate port number.

To connect with watsonx.data Presto server from a Python client, complete the following steps:

1. Install python 3.x (3.10 or later recommended) & pip3 on your client workstation.
2. Install common python modules to connect to Presto.

   ```
   pip3 install SQLAlchemy 'pyhive[presto]' presto-python-client
   ```

3. If required, complete the instructions in "Using presto-cli executable (remote) – Developer edition" on page 451 to get the hostname, port, server certificate, authentication, and test the connectivity.

4. Start the sandbox container for the registered Presto engine.

```
ibm-lh-client/bin/dev-sandbox --engine=demo-b
```

5. In the bash prompt, install the `prestodb` module.

```
export HOME=/tmp
pip3 install SQLAlchemy 'pyhive[presto]' presto-python-client
```

   Important: Steps 4 and 5 are required only if you are not using the Prestodb module in `ibm-lh-client`. Also, currently due to an issue with the Prestodb module in `ibm-lh-client,` you must complete steps 4 and 5.

6. Use the DBAPI interface to query the Presto server. Following is a sample Python script:

   Note:

   - See how the "ENG_" environment variables are examples of how the relevant Presto server locations can be used with the `dbapi.connect` function call.
   - An example of how to point to the location of the Presto server's certificate file is represented by the `cert_location` variable.
   - The `conn._http_session.verify = cert` _location property is used to point to the certificate location (for self-signed cases only).

```
import os
import prestodb

username=os.environ["ENG_USERNAME"]
password=os.environ["ENG_PASSWORD"]
hostname=os.environ["ENG_HOST"]
portnumber=os.environ["ENG_PORT"]
cert_location='./ibm-lh-client/localstorage/volumes/infra/tls/aliases/' + hostname + ':' +
portnumber + '.crt'

with prestodb.dbapi.connect(
host=hostname,
port=portnumber,
user=username,
catalog='tpch',
schema='tiny',
http_scheme='https',
auth=prestodb.auth.BasicAuthentication(username,password)
) as conn:
  conn._http_session.verify = cert_location
  cur = conn.cursor()
  cur.execute('select * from tpch.tiny.customer limit 10')
  rows = cur.fetchall()
  print(rows)
```

Using Python scripts – software

This section provides instructions to connect to a Presto server from a Python client.

Before you get started, ensure the following:

- the Red Hat OpenShift Project namespace administrator or cluster administrator has exposed the route to the Presto Server. The administrators should refer to Exposing secure route to Presto server for instructions.
- you have the received the hostname and port (by default 443) associated with that specific Red Hat OpenShift route from the administrator.

To connect with watsonx.data Presto server from a Python client, complete the following steps:

1. Install python 3.x (3.10 or later recommended) and `pip3` on your client workstation.
2. Install common python modules to connect to Presto.

```
pip3 install SQLAlchemy 'pyhive[presto]' presto-python-client
```

3. If required, complete the instructions in "Using presto-cli executable (remote) - software" on page 452 to get the hostname, port, server certificate, authentication, and test the connectivity.
4. Start the sandbox container for the registered Presto engine.

   ```
   ibm-lh-client/bin/dev-sandbox --engine=demo-b
   ```

5. In the bash prompt, install the `prestodb` module.

   ```
   export HOME=/tmp
   pip3 install SQLAlchemy 'pyhive[presto]' presto-python-client
   ```

   Important: Steps 4 and 5 are required only if you are not using the Prestodb module in `ibm-lh-client`. Also, currently due to an issue with the Prestodb module in `ibm-lh-client,` you must complete steps 4 and 5.

6. Use the DBAPI interface to query the Presto server. Following is a sample Python script:

   Note:

   - See how the "ENG_" environment variables are examples of how the relevant Presto server locations can be used with the `dbapi.connect` function call.
   - An example of how to point to the location of the Presto server's certificate file is represented by the `cert_location` variable.
   - The `conn._http_session.verify = cert` _location property is used to point to the certificate location (for self-signed cases only).

   ```
   import os
   import prestodb

   username=os.environ["ENG_USERNAME"]
   password=os.environ["ENG_PASSWORD"]
   hostname=os.environ["ENG_HOST"]
   portnumber=os.environ["ENG_PORT"]
   cert_location='./ibm-lh-client/localstorage/volumes/infra/tls/aliases/' + hostname + ':' +
   portnumber + '.crt'

   with prestodb.dbapi.connect(
   host=hostname,
   port=portnumber,
   user=username,
   catalog='tpch',
   schema='tiny',
   http_scheme='https',
   auth=prestodb.auth.BasicAuthentication(username,password)
   ) as conn:
     conn._http_session.verify = cert_location
     cur = conn.cursor()
     cur.execute('select * from tpch.tiny.customer limit 10')
     rows = cur.fetchall()
     print(rows)
   ```

Chapter 8. Integrations

You can extend the functionality of the watsonx.data by integrating with other SQL query engines and databases to share data and run your analytics and AI workloads on all of your data across any environment.

watsonx.data on Red Hat OpenShift

Enabling Apache Ranger policy for resources

IBM watsonx.data now supports Apache Ranger policies to allow comprehensive data security on integrating with multiple governance tools and engines.

Before you begin

Ensure you have the following details:

- IBM watsonx.data instance.
- Apache Ranger environment.
- The Presto (Java) JDBC URL and credentials in watsonx.data.
- watsonx.data and Apache Ranger are integrated with LDAP to sync users or groups.

Procedure

1. Complete the following steps to create a service in the Ranger.

    a. Log in to Apache Ranger by using the username and password.

    b. The home page lists all the services that are already configured under different resources. To create a new service, click the **+** icon next to the **PRESTO** resource.

    c. Provide the following details:

Field	Description
Username	admin
Password	UXXXXXXR
jdbc.url	Provide the **JDBC** URL.

    d. The service is successfully added in the **PRESTO** resource list. Click the service name to verify that the default policies are added.

    Note: The testing might fail initially, you can re-test the connection after saving the details since the default policies will be automatically added after saving.

2. Complete the following steps to enable and configure Apache Ranger in watsonx.data.

    a. Log in to watsonx.data console.

    b. From the navigation menu, select **Access control**.

    c. Click the **Integrations** tab.

    d. Click **Integrate service**. The **Integrate service** window opens.

    e. In the **Integrate service** window, provide the following details:

Field	Description
Service	Select **Apache Ranger**.

URL	The URL of Apache Ranger.
Username	The admin credentials.
Password	The admin credentials.
List resources	Click the link to load the resources that are available in the Apache Ranger server.
Resources	Select the resource for which the Apache Ranger policy must be enabled.
Enable data policy within watsonx.data	Select the checkbox to enable data policy along with Apache Ranger policy.

f. Click **Integrate**. The Apache Ranger policy is integrated and listed in the **Access Control** page.

3. Complete the following steps to verify access control :

   a. Log in to watsonx.data instance.

   b. From the navigation menu, click **Query workspace**.

   c. Execute a simple query. The access denied error appears as currently no policies are defined in the Ranger for the user.

4. Complete the following steps to grant permissions to the user:

   a. Log in to Apache Ranger.

   b. Grant the required permission to the test user.

   c. Scroll down to the bottom, click the **Save** button.

   d. Log in to watsonx.data instance and execute a query again. The access is allowed for the user after adding policies in the Ranger.

Limitations

- In Apache Iceberg catalog, an error occurs if a policy is not defined for the snapshots views related to the tables in Ranger. You must manually define policies in Apache Ranger to eliminate the error.
- watsonx.data supports access control for Apache Ranger integration.

Adding row-level filtering policy

Row-level filtering allows users to access a subset of rows in a table.

Before you begin

Ensure you have the following details:

- IBM watsonx.data instance.
- Apache Ranger is provisioned.
- Enable the Apache Ranger policy in watsonx.data. For more information, see "Enabling Apache Ranger policy for resources" on page 459.

Procedure

1. Complete the following steps.

   a. Log in to Apache Ranger by using the username and password.

   b. Go to **Service Manager >** ***service_name*** **policies** page to add a row level filter policy.

   c. In the **Policy Details** section, provide the policy details like name, description.

d. In the **Resources** section, select the catalog, schema, and table for which the policy is applicable.

e. In the **Row Filter Conditions** section, select the user (example, **User1**), set the **Permissions** to **Select** and enter the **Row Level Filter** (example, **c1=1**).

f. Click **Save**.

2. Complete the following steps to verify access control:

   a. Log in to watsonx.data instance as **User1**.

   b. From the navigation menu, click **Query workspace**.

   c. Run a simple query to access the table again, now **User1** can only see rows with column c1 equal to 1 in the result.

Adding column masking policy

Add a column masking policy to restrict users from accessing sensitive data. Users can view data only in masked format.

Before you begin

- IBM watsonx.data instance.
- Apache Ranger is provisioned.
- Enable the Apache Ranger policy in watsonx.data. For more information, see "Enabling Apache Ranger policy for resources" on page 459.

Procedure

1. Complete the following steps.

   a. Log in to Apache Ranger by using the username and password.

   b. Go to **Service Manager >** ***service_name*** **policies** page to add a masking policy.

   c. In the **Policy Details** section, provide the policy details like name, description.

   d. In the **Resources** section, select the catalog, schema, table, and column for which the policy is applicable.

   e. In the **Masking Conditions** section, select the user (example, **User1**), set the **Permissions** to **Select**, set the **Select Masking Option** to **Custom** and enter the masked value as **xxx**.

   f. Click **Save**.

2. Complete the following steps to verify access control:

   a. Log in to watsonx.data instance as **User1**.

   b. From the navigation menu, click **Query workspace**.

   c. Run a simple query to access the table again, now **User1** view see the sensitive data in masked format (xxx) in the target column in the result.

Integrating with Db2

You can extend the functionality of the watsonx.data by seamlessly integrating with Db2 to share data across the products enabling the use of the most appropriate engine for each workload.

watsonx.data on Red Hat OpenShift

You can configure watsonx.data integration with Db2. See Db2.

Integrating with Netezza Performance Server

You can extend the functionality of the watsonx.data by integrating with Netezza Performance Server to connect to the Hive Metastore (HMS) which is a data lake metastore server and query from Apache Iceberg tables that live on your data lake S3 object store.

watsonx.data on Red Hat OpenShift

You can configure watsonx.data integration with Netezza Performance Server. See Netezza Performance Server .

Integrating with IBM Knowledge Catalog

Integrating IBM watsonx.data with IBM Knowledge Catalog provides self-service access to data assets for knowledge workers who need to use those data assets to gain insights.

watsonx.data on Red Hat OpenShift

Before you begin

The IBM Knowledge Catalog integration governs all data in the Presto catalogs that are configured in the **Infrastructure manager**. Import all data assets into the governed catalog before setting up the integration in **Infrastructure manager**.

After integration, you cannot import new data assets into the governed catalogs because watsonx.data doesn't allow accessing ungoverned data.

If you disable the integration, the data protection rules cannot protect the data. Consider the following workaround options to mitigate the security risks when integration is disabled:

- Stop the applications that are connected to watsonx.data temporarily. If you are an administrator, monitor the Presto Query dashboard to make sure that others are not using the system.
- If you are a data steward, you can create rules to deny access for others. For more information, see the data protection rules and setup rules to deny access.
- You can temporarily disable the Presto external route. For more information, see Exposing secure route to Presto server.
- You can define the built-in data policies to prevent others from accessing the data. For more information, see Data policy.

Note: You can select only non-database catalogs. Database catalogs are not supported in watsonx.data.

About this task

IBM Knowledge Catalog provides a secure enterprise catalog management platform that is supported by a data governance framework. A catalog connects people to the data and knowledge that they need.

A catalog is how you share assets across your enterprise:

- Collaborators in a catalog have access to data assets without needing separate credentials or being able to see the credentials.
- An asset in a catalog consists of metadata about data, including how to access the data, the data format, the classification of the asset, which collaborators can access the data and other types of metadata that describe the data

Important: watsonx.data supports adding assets by using watsonx.data connector only. Assets that are brought into Cloud Pak for Data by using Presto connector are governed in Cloud Pak for Data, but not in watsonx.data.

Procedure

To integrate watsonx.data with IBM Knowledge Catalog, complete the following steps:

1. Log in to watsonx.data console.
2. From the navigation menu, select **Access control**.
3. Click the **Integrations** tab.
4. Click **Integrate service**. The **Integrate service** window opens.
5. In the **Integrate service** window, provide the following details:

Field	Description
Service	Select the service (Knowledge Catalog) to be integrated.
Storage catalogs	Select the storage catalogs for Knowledge Catalog governance.
IKC endpoint	Specify the Knowledge Catalog endpoint URL. For example, `https://<instance>.ibm.com`
Zen API key	Specify the Zen API key. For more information, see Generating an API authorization token.

6. Click **Integrate**.

   The service is integrated and listed in the **Access Control** page.

   Note: You can transform or mask data in watsonx.data based on the data protection rules that are defined in the IBM Knowledge Catalog.

   Note: Integrating watsonx.data with IBM Knowledge Catalog is not supported in version 1.1.1.

Enabling IBM Knowledge Catalog

After Integrating IBM watsonx.data with IBM Knowledge Catalog, the next step is to enable IBM Knowledge Catalog (IKC) on watsonx.data

watsonx.data on Red Hat OpenShift

Before you begin

Ensure to have the following prerequisites for enabling IBM Knowledge Catalog on watsonx.data

- Working watsonx.data on-premises (CPD) environment
- Working IKC environment URL (release version - 4.8)
- IKC Admin Zen Api Key

Note:

- The current IKC Integration supports only transformation and masking (redact, obfuscate, and substitute) and denial of data as per the rules defined in IKC.
- Both watsonx.data and IKC must be present in the on-premises (CPD) environment.

What to do next

1. Connect and import the table metadata from watsonx.data to IKC
2. Associate a user to the table asset
3. Configure IBM Knowledge Catalog

Connect and import the asset metadata from watsonx.data to IBM Knowledge Catalog

Do the following steps to connect and import the asset (tables or views) metadata from watsonx.data to IKC.

Procedure

1. Login to the IKC Cloud Pack for Data instance by using admin credentials.
2. From the left pane, go to **Catalogs** > **All catalogs** to view the available catalogs.
3. Click **New catalog**.

   The **New catalog** page appears.
4. In the **New catalog** page, enter the following details and click **Create**.

Field	Description
Name	Enter the catalog name.
Description	Enter the catalog description.
Enforce data protection rules	Select to enable enforcing data protection rules.

   The catalog is created and the catalog page opens.
5. Go to **Add to catalog** > **Connection**.
6. Search and select the IBM watsonx.data connection type and click **Select**.
7. Enter the following details for the connection.

Field	Description
Name	Enter the name of the connection.
Description	Enter a connection description.
Connect to IBM watsonx.data on Cloud Pak for Data	Select the checkbox.
Hostname or IP address	Enter the watsonx.data instance URL.
Port	Enter the port number.
Instance ID	Enter the instance ID. You can get the instance ID from the watsonx.data instance home page (information icon).
Instance name	Enter the watsonx.data instance name.
Credential setting	choose **Personal** for private connection or **Shared** for public connection.
Username	Enter your username of the watsonx.data instance.
Password	Enter your password of the watsonx.data instance.
Mask sensitive credentials retrieved through API calls	Toggle the switch to **On** position to mask sensitive credentials, such as passwords or API keys, when accessing this connection through API calls (for example, when using this connection in a notebook).
Engine is SSL enabled	Select the checkbox. IBM watsonx.data instance is configured to accept SSL connections
Engine's SSL certificate	Download the certificate from the watsonx.data console using a web browser and paste in this field.

Field	Description
Engine's hostname or IP address	Enter the engine hostname available in the watsonx.data console.
Engine ID	Enter the engine ID available in the watsonx.data console.
Engine's port	Enter the engine's port number

8. Optional: Click **Test connection** to test the connection.
9. Click **Create**.

   The connection is added to the catalog.
10. Go to **Add to catalog** > **Connected assets**.
11. Click **Select source**, select an asset, and click **Select**.

    Note: For **Personal** credentials, you have to enter your user name and password to select the asset.
12. Click **Add** to add the asset.

    The asset is successfully added to IKC from watsonx.data.
13. Click the asset name to open the asset page with details.
14. Verify the details in the **Data class** column against all of the other columns assigned during the metadata import.

    Note: You can define rules in IKC based on these data classes.
15. Optional: Click the **Profile** tab and change any data class as per the requirement.

    Note: Changing the data class is not mandatory. Do it only if the populated data class is wrong and needs to be changed.

Associate a user to the asset

The asset (table or a view) with data protection rules has the unmasked data for the owner user. The data is masked for all of the other users. Do the following steps to associate a user to the asset in IKC and assign the ownership.

Procedure

1. Login to the IKC Cloud Pack for Data instance by using admin credentials.
2. From the left pane, go to **Catalogs** > **All catalogs** to view the available catalogs.
3. Select the catalog to open the catalog details page.
4. Click the catalog name and go to the **Access control** tab.
5. Go to **Add collaborators** > **Add user** and select a user role (Admin, Editor, or Viewer).
6. Search and select one or more users from the list and click **Add**.

   The user addition is successful.
7. Go to the **Assets** tab of the catalog details page, click the asset name, and go to the **Access** tab of the asset.
8. Click **Add members**, search for the added user, and click **Add**.

Changing the owner of the asset

Procedure

1. Go to the **Assets** tab of the catalog details page, click the asset name to open the asset details.
2. Click on the edit icon beside **Asset owner** and select a new user from the list.
3. Click, **Apply**.

   The asset owner is changed.

Configure IBM Knowledge Catalog

Do the following steps to associate a user to the asset (tables or views) in IKC and assign the ownership.

Procedure

1. Login to the IKC Cloud Pack for Data instance by using admin credentials.
2. From the watsonx.data home page, go to **Access control**.
3. Go to the **Integration** tab and click **Integrate service**.
4. Enter the following details:

Field	Description
Service	Select `IBM Knowledge Catalog`.
Bucket catalog	Select the applicable bucket catalogs for IKC governance.
WKC endpoint	Enter the IKC endpoint URL.
API key	Enter the Zen API key. For more information, see Generating API keys for authentication.

5. Click **Integrate**.

   The IKC integration is successful.

Supported data types for IBM Knowledge Catalog Integration

IBM Knowledge Catalog Integration with watsonx.data supports the following data types for transformation and masking.

watsonx.data on Red Hat OpenShift

- Varchar
- Bigint
- Boolean
- Date
- Double
- Integer
- Smallint
- Timestamp
- Tinyint

Integrating with Data Virtualization

You can integrate IBM watsonx.data with **`Data Virtualization`** on Cloud Pak for Data to easily join data from different sources in one unified view, without manual changes, data movement, or replication.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Following are the requirements to integrate watsonx.data with **`Data Virtualization`**:

- Cloud Pak for Data with **`Data Virtualization`** installed.
- Cloud Pak for Data with watsonx.data installed.

For more information to integrate and start using **`Data Virtualization`**, see Virtualizing data with Data Virtualization.

Integrating with DataStage

You can integrate IBM watsonx.data with IBM DataStage on Cloud Pak for Data to ingest and to read data from watsonx.data.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Requirements

Following are the requirements to integrate watsonx.data with DataStage:

- Cloud Pak for Data with DataStage (version 5.0.1 or later)
- IBM watsonx.data (version 2.0.0 or later)

  Note: Ingestion is not supported by watsonx.data developer edition in DataStage. The developer edition supports only reading data.
- Use watsonx.data connector in DataStage to read or ingest data into watsonx.data.
- Make sure that the Data Access Service (DAS) is enabled in watsonx.data to enable data ingestion in DataStage.
- Presto engine must be provisioned in watsonx.data to read and ingest data in DataStage.
- Amazon S3 storage must be connected to watsonx.data to enable data ingestion in DataStage.
- The Amazon S3 storage must be connected to the Iceberg catalog, which must be associated to the Presto engine to enable data ingestion in DataStage.

For more information about connecting to a data source in DataStage, see Connecting to a data source in DataStage.

For more information about creating a connection asset in watsonx.data, see IBM watsonx.data connection.

Chapter 9. Working with Spark

You can use watsonx.data to seamlessly integrate with Spark engine to achieve the following use cases:

watsonx.data on Red Hat OpenShift

- Ingesting large volumes of data into watsonx.data tables.

  Note: You can also cleanse and transform data before ingestion.
- Table maintenance operations to enhance performance.
- Complex analytics workloads that are difficult to represent as queries.

External Spark engines

External Spark engines are engines that exist in a different environment from where watsonx.data is deployed. You can deploy them in the following environments.

watsonx.data on Red Hat OpenShift

- Spark instance on Cloud
- Spark on Cloud Pak for Data
- Spark on EMR

Based on the environment where the Spark engine is deployed, select the respective section to connect to watsonx.data:

Working with Spark on Cloud

Integrate watsonx.data with Analytics Engine serverless instance (Spark on Cloud) and run your Spark workloads.

watsonx.data on Red Hat OpenShift

Before you begin

- Install IBM watsonx.data and ensure that the instance is up and running. For more information, see Installing watsonx.data on Red Hat OpenShift.
- Provision an IBM Analytics Engine serverless instance. For more information, see Provisioning an IBM Analytics Engine serverless instance.
- Ensure that you have **Admin** or **Metastore Admin** privilege to submit spark jobs to watsonx.data. For more information, see Managing access to metastore. This is required at the time of configuring Spark Cloud instance.
- Ensure to associate a bucket with Hive Metastore (HMS). For more information, see Adding a bucket-catalog pair. This is required at the time of configuring Spark Cloud instance.

Procedure

1. Expose the Hive Metastore (HMS) by using a `NodePort` service. For more information, see Expose HMS.
2. Import a self-signed certificate. For more information, see Importing self-signed certificates from a Hive metastore server to a Java truststore.
3. Create an Analytics Engine serverless library set and upload the `truststore.jks` file into the library set to make it available to the Analytics Engine Spark application. Complete the following steps to create a library set by using the Analytics Engine serverless customization script:
   a) Create a Python file named `customization_script.py` with the following content. Analytics Engine's customization component looks for a Python module with this name.

```
import cos_utils
import os
import sys

def customize(install_path, params):
  print("inside base install_misc_package(), override this to implement your own
implementation.")
  endpoint = params[0]
  bucket_name = params[1]
  truststore_file = params[2]  # Change parameter name to truststore_file
  access_key = params[3]
  secret_key = params[4]
  cos = cos_utils.get_cos_object(access_key, secret_key, endpoint)

  retCode = cos_utils.download_file(truststore_file, cos, bucket_name, "{}/
{}".format(install_path,truststore_file))  # Download the truststore file
  if retCode != 0:
    print("Non-zero return code while downloading file: {}".format(str(retCode)))
    sys.exit(retCode)
  else:
    print("Successfully downloaded truststore file.")
```

b) Store the truststore and `customization_script.py` files in IBM Cloud Object Storage. For example, `ibmcloud cos upload --bucket --key --file`.

c) Run the following curl command for submitting the application to create the library set by using a payload.

```
curl -X POST  https://api.<CHANGEME_REGION>.ae.cloud.ibm.com/v3/analytics_engines/
<INSTANCE_ID>/spark_applications  --header "Authorization: Bearer {IAM_TOKEN}"  -H
"Content-Type: application/json"  -d@payload.json
```

Example of payload:

```
{
  "application_details": {
  "application": "/opt/ibm/customization-scripts/customize_instance_app.py",
    "arguments": ["{\"library_set\":{\"action\":\"add\",\"name\":\"wxd-
library\",\"script\":{\"source\":\"py_files\",\"params\":[\"https://
s3.direct.<CHANGEME_REGION>.cloud-object-
storage.appdomain.cloud\",\"<CHANGEME_BUCKET_NAME>\",\"truststore.jks\",\"<CHANGEME_ACCESS
_KEY>\",\"<CHANGEME_SECRET_KEY>\"]}}}"],
    "conf": {
      "spark.hadoop.fs.cos.dev-cos.endpoint":"https://
s3.publicendpoint.<CHANGEME_REGION>.cloud-object-storage.appdomain.cloud",
      "spark.hadoop.fs.cos.dev-cos.access.key":"<CHANGEME_ACCESS_KEY>",
      "spark.hadoop.fs.cos.dev-cos.secret.key":"<CHANGEME_SECRET_KEY>",
      "spark.submit.pyFiles":"cos://<CHANGEME_BUCKET_NAME>.dev-cos/
customization_script.py"
    }
  }
}
```

4. Configure your Analytics Engine instance with your watsonx.data instance. For more information, see Configuring an Analytics Engine

Configuring Analytics Engine

You can configure IBM Analytics Engine instance to connect to the IBM watsonx.data instance. Set watsonx.data configurations and Spark related configuration as the default configuration for the IBM Analytics Engine instance.

watsonx.data on Red Hat OpenShift

You can configure Analytics Engine instance with default settings in one of the following ways:

- Configure by using IBM Cloud console
- Configure by using the Analytics Engine API
- Configure by using the Analytics Engine CLI

Configuring an Analytics Engine instance by using IBM Cloud console

To configure your Analytics Engine instance from the IBM Cloud Resource list, complete the following steps:

1. Log in to your IBM Cloud account.
2. Access the IBM Cloud Resource list.
3. Find your Analytics Engine instance and click the instance to see the details.
4. Click **Manage** > **Configuration** to view the configuration.
5. In the **Default Spark configuration** section, click **Edit**.
6. Add the following configuration to the **Default Spark configuration** section.

```
{
"spark.sql.catalogImplementation": "hive",
"spark.sql.extensions": "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
"spark.sql.iceberg.vectorization.enabled": "false",
"spark.sql.catalog.lakehouse": "org.apache.iceberg.spark.SparkCatalog",
"spark.sql.catalog.lakehouse.type": "hive",
"spark.sql.catalog.lakehouse.uri": "<public_IP_address>:<nodeport>",
"spark.hive.metastore.client.auth.mode": "PLAIN",
"spark.hive.metastore.client.plain.username": "<metastore_admin_user>",
"spark.hive.metastore.client.plain.password": "<metastore_admin_password>",
"spark.hive.metastore.use.SSL": "true",
"spark.hive.metastore.truststore.type": "JKS",
"spark.hive.metastore.truststore.path": "file:///home/spark/shared/user-libs/wxd-library/
custom/truststore.jks",
"spark.hive.metastore.truststore.password": "<trustsore_password>"
}
```

Parameter values:

`<public_IP_address>:<nodeport>` - public_IP address and nodeport for the watsonx.data instance.

`<metastore_admin_user>` - watsonx.data cluster metastore-admin user.

Note:

To submit spark jobs to watsonx.data , you must have **Admin** or **Metastore admin** privilege in watsonx.data. The preferred one is **Metastore admin**. For more information, see Managing access to metastore.

`<metastore_admin_password>` - watsonx.data cluster metastore-admin password.

`<trustsore_password>` - `truststore.jks` password.

Configuring an Analytics Engine instance by using Analytics Engine API

To configure your IBM Analytics Engine instance from the Analytics Engine API, complete the following steps:

1. Generate an IAM token to connect to the Analytics Engine API. For more information about how to generate an IAM token, see Granting permissions to users.
2. Run the API to set instance default configuration:

```
curl -X PATCH --location --header "Authorization: Bearer {IAM_TOKEN}" --header "Accept:
application/json" --header "Content-Type: application/merge-patch+json" --data '{
<CONFIGURATION_DETAILS>
}' "{BASE_URL}/v3/analytics_engines/{INSTANCE_ID/default_configs"
```

Parameter values:

`IAM_TOKEN` - The API token generated for the Analytics Engine API.

`INSTANCE_ID` - The Analytics Engine instance ID. For more information, see Obtaining the service endpoints using the IBM Cloud CLI.

CONFIGURATION_DETAILS

```
{
"spark.sql.catalogImplementation": "hive",
"spark.sql.extensions": "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
"spark.sql.iceberg.vectorization.enabled": "false",
"spark.sql.catalog.lakehouse": "org.apache.iceberg.spark.SparkCatalog",
"spark.sql.catalog.lakehouse.type": "hive",
"spark.sql.catalog.lakehouse.uri": "<public_IP_address>:<nodeport>",
"spark.hive.metastore.client.auth.mode": "PLAIN",
"spark.hive.metastore.client.plain.username": "<metastore_admin_user>",
"spark.hive.metastore.client.plain.password": "<metastore_admin_password>",
"spark.hive.metastore.use.SSL": "true",
"spark.hive.metastore.truststore.type": "JKS",
"spark.hive.metastore.truststore.path": "file:///home/spark/shared/user-libs/wxd-library/
custom/truststore.jks",
"spark.hive.metastore.truststore.password": "<trustsore_password>"
}
```

Configuring an Analytics Engine instance by using Analytics Engine CLI

To specify the configuration settings for your IBM Analytics Engine instance from CLI, complete the following steps:

Run the following command:

```
ibmcloud analytics-engine-v3 instance default-configs-update [--id INSTANCE_ID] --body BODY
```

Parameter values:

- `INSTANCE_ID` - The Analytics Engine instance ID. For more information, see Obtaining the service endpoints.
- BODY - Copy and paste the following configuration information:

```
{
"spark.sql.catalogImplementation": "hive",
"spark.sql.extensions": "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
"spark.sql.iceberg.vectorization.enabled": "false",
"spark.sql.catalog.lakehouse": "org.apache.iceberg.spark.SparkCatalog",
"spark.sql.catalog.lakehouse.type": "hive",
"spark.sql.catalog.lakehouse.uri": "<public_IP_address>:<nodeport>",
"spark.hive.metastore.client.auth.mode": "PLAIN",
"spark.hive.metastore.client.plain.username": "<metastore_admin_user>",
"spark.hive.metastore.client.plain.password": "<metastore_admin_password>",
"spark.hive.metastore.use.SSL": "true",
"spark.hive.metastore.truststore.type": "JKS",
"spark.hive.metastore.truststore.path": "file:///home/spark/shared/user-libs/wxd-library/
custom/truststore.jks",
"spark.hive.metastore.truststore.password": "<trustsore_password>"
}
```

After you configure Analytics Engine, you can submit the Spark application. For more information, see Run Spark use case.

Running Spark application

You can run Spark use cases for watsonx.data by using Python samples. All the samples are written by using Spark Python APIs.

watsonx.data on Red Hat OpenShift

Before you begin

To enable your Spark application to work with the watsonx.data catalog and storage, you must have `Metastore admin` role. Without `Metastore admin` privilege, you cannot ingest data to storage using

Native Spark engine. To enable your Spark application to work with the watsonx.data catalog and storage, add the following configuration to your application payload:.

```
spark.hive.metastore.client.plain.username=ibmlhapikey
spark.hive.metastore.client.plain.password=<api-key-of-the-user-which-has-metastore-admin-role>
spark.hadoop.wxd.apiKey=Basic base64(ibmlhapikey_ibmcloudid:apikey)
```

About this task

About the sample use case

The sample file demonstrates the following functions:

- **Accessing tables from** watsonx.data

  : The **Create a database in Lakehouse catalog** section from the sample use case creates a database **demodb** in the configured watsonx.data instance with a catalog named **lakehouse**. **demodb** is configured to store all the data and metadata under the Cloud Object Storage bucket **lakehouse-bucket**. It also creates an iceberg table **testTable** and accesses it.
- **Ingesting data to** watsonx.data

  : The **Ingest parquet data into a lakehouse table** section from the sample use case allows you to ingest data in parquet and CSV format from a **spark-vol** into a watsonx.data table. Sample data in parquet format is inserted from the volume into the watsonx.data table **yellow_taxi_2022** . The **Ingest parquet data into a lakehouse table** section from the sample python file also shows ingesting data in CSV format from the **spark-vol** into the table **zipcode** in the database **demodb**. For more information, see Inserting sample data into the Cloud Object Storage.
- **Modifying the schema in** watsonx.data

  : The **Schema evolution** section from the sample use case allows you to modify data in watsonx.data.
- **Table maintenance activities in** watsonx.data

  : Table maintenance helps in keeping the watsonx.data table performance. Iceberg provides table maintenance procedures out of the box that allows performing powerful table optimizations in a declarative fashion. The following sample demonstrates how to do some table maintenance operations by using Spark. For more information about the Iceberg Spark table maintenance operations, see Table Operations.

To run Spark use cases for watsonx.data by using sample Python files:

Procedure

1. To insert data to Cloud Object Storage, for the following steps.

   Inserting sample data into the Cloud Object Storage

   a. Create a Cloud Object Storage (for example, source-bucket) to store sample data to be ingested into watsonx.data instance. For information about creating storage volume, see Getting started with IBM Cloud Object Storage.

      As a user of Object Storage (storage volume) , you not only need to know the API key or the HMAC keys to configure Object Storage, but also the IBM Analytics Engine service endpoints to connect to Object Storage. See Selecting regions and endpoints for more information on the endpoints to use based on your Object Storage bucket type, such as regional versus cross-regional. You can also view the endpoints across regions for your Object Storage service by selecting the service on your IBM Cloud dashboard and clicking **Endpoint** in the navigation pane.

   b. Download sample CSV file (for example, zipcodes.csv) and parquet sample data (for example, six months taxi data for year 2022) from the following links.

      - Sample parquet file

- Sample CSV file

c. Install IBM Cloud Object Storage plug-in. For more information about how to install plug-in, see IBM Cloud Object Storage CLI.

d. Use the COS CLI to upload the sample data into Cloud Object Storage bucket.

```
ibmcloud cos upload --bucket <cos_bucket_name> --key <source_file_name> --file
<path_to_source_file>
```

Parameter values:

- <cos_bucket_name>: name of the storage volume.

- <source_file_name>: the name of the sample data file that you downloaded. Here, `key zipcodes.csv` is the file name (see the following example).

- <path_to_source_file>: the path to the location in your machine where the file resides. Here, `path/zipcodes.csv` is the file path (see the following example).

For example:

```
ibmcloud cos upload --bucket source-bucket --key zipcodes.csv --file <path/
zipcodes.csv>
```

2. Save the following sample as a Python file:

```
from pyspark.sql import SparkSession
import os

def init_spark():
    spark = SparkSession.builder \
        .appName("lh-hms-cloud") \
        .config("spark.hadoop.fs.s3a.bucket.lakehouse-
bucket.endpoint" ,"<cos_bucket_endpoint>") \
        .config("spark.hadoop.fs.s3a.bucket.lakehouse-bucket.access.key" ,"<access_key>") \
        .config("spark.hadoop.fs.s3a.bucket.lakehouse-bucket.secret.key" ,"<secret_key>") \
        .enableHiveSupport() \
        .getOrCreate()
    return spark

def create_database(spark):
    # Create a database in the lakehouse catalog
    spark.sql("create database if
not exists lakehouse.demodb LOCATION 's3a://lakehouse-bucket/'")

def list_databases(spark):
    # list the database under lakehouse catalog
    spark.sql("show databases from lakehouse").show()

def basic_iceberg_table_operations(spark):
    # demonstration: Create a basic Iceberg table, insert some data and then query table
    spark.sql("create table if not exists lakehouse.demodb.testTable(id
INTEGER, name VARCHAR(10), age INTEGER, salary DECIMAL(10, 2)) using iceberg").show()
    spark.sql("insert into lakehouse.demodb.testTable
values(1,'Alan',23,3400.00),(2,'Ben',30,5500.00),(3,'Chen',35,6500.00)")
    spark.sql("select * from lakehouse.demodb.testTable").show()

def create_table_from_parquet_data(spark):
    # load parquet data into dataframe
    df = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-01.parquet")
    # write the dataframe into an Iceberg table
    df.writeTo("lakehouse.demodb.yellow_taxi_2022").create()
    # describe the table created
    spark.sql('describe table lakehouse.demodb.yellow_taxi_2022').show(25)
    # query the table
    spark.sql('select * from lakehouse.demodb.yellow_taxi_2022').count()

def ingest_from_csv_temp_table(spark):
    # load csv data into a dataframe
    csvDF = spark.read.option("header",True).csv("file:///spark-vol/zipcodes.csv")
    csvDF.createOrReplaceTempView("tempCSVTable")
    # load temporary table into an Iceberg table
    spark.sql('create or replace
table lakehouse.demodb.zipcodes using iceberg as select * from tempCSVTable')
    # describe the table created
    spark.sql('describe table lakehouse.demodb.zipcodes').show(25)
    # query the table
    spark.sql('select * from lakehouse.demodb.zipcodes').show()

def ingest_monthly_data(spark):
    df_feb = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-02.parquet")
    df_march = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-03.parquet")
    df_april = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-04.parquet")
    df_may = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-05.parquet")
    df_june = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-06.parquet")
    df_q1_q2 = df_feb.union(df_march).union(df_april).union(df_may).union(df_june)
    df_q1_q2.write.insertInto("lakehouse.demodb.yellow_taxi_2022")

def perform_table_maintenance_operations(spark):
    # Query the metadata files table to list underlying data files
    spark.sql("SELECT file_path, file_size_in_bytes
FROM lakehouse.demodb.yellow_taxi_2022.files").show()
    # There are many smaller files compact them into files of 200MB each using the
    # `rewrite_data_files` Iceberg Spark procedure
    spark.sql(f"CALL lakehouse.system.rewrite_data_files(table
=> 'demodb.yellow_taxi_2022', options => map('target-file-size-bytes','209715200'))").show()
    # Again, query the
metadata files table to list underlying data files; 6 files are compacted
    # to 3 files
    spark.sql("SELECT file_path, file_size_in_bytes
FROM lakehouse.demodb.yellow_taxi_2022.files").show()
    # List all the snapshots
    # Expire earlier snapshots. Only latest one with compacted data is required
    # Again, List all the snapshots to see only 1 left
    spark.sql("SELECT committed_at, snapshot_id,
operation FROM lakehouse.demodb.yellow_taxi_2022.snapshots").show()
    #retain only the latest one
    latest_snapshot_committed_at = spark.sql("SELECT committed_at, snapshot_id,
operation FROM lakehouse.demodb.yellow_taxi_2022.snapshots").tail(1)[0].committed_at
    print (latest_snapshot_committed_at)
    spark.sql(f"CALL lakehouse.system.expire_snapshots(table
=> 'demodb.yellow_taxi_2022',older_than => TIMESTAMP
'{latest_snapshot_committed_at}',retain_last => 1)").show()
    spark.sql("SELECT committed_at, snapshot_id,
operation FROM lakehouse.demodb.yellow_taxi_2022.snapshots").show()
    # Removing Orphan data files
    spark.sql(f"CALL lakehouse.system.remove_orphan_files(table
=> 'demodb.yellow_taxi_2022')").show(truncate=False)
    # Rewriting Manifest Files
    spark.sql(f"CALL lakehouse.system.rewrite_manifests('demodb.yellow_taxi_2022')").show()

def evolve_schema(spark):
    # demonstration: Schema evolution
    # Add column fare_per_mile to the table
    spark.sql('ALTER TABLE lakehouse.demodb.yellow_taxi_2022
ADD COLUMN(fare_per_mile double)')
    # describe the table
    spark.sql('describe table lakehouse.demodb.yellow_taxi_2022').show(25)

def clean_database(spark):
    # clean-up the demo database
    spark.sql('drop table if exists lakehouse.demodb.testTable purge')
    spark.sql('drop table if exists lakehouse.demodb.zipcodes purge')
    spark.sql('drop table if exists lakehouse.demodb.yellow_taxi_2022 purge')
    spark.sql('drop database if exists lakehouse.demodb cascade')

def main():
    try:
        spark = init_spark()
        create_database(spark)
        list_databases(spark)
        basic_iceberg_table_operations(spark)
        # demonstration: Ingest parquet and csv data into a watsonx.data Iceberg table
        create_table_from_parquet_data(spark)
        ingest_from_csv_temp_table(spark)
        # load data for
the month of February to June into the table yellow_taxi_2022 created above
        ingest_monthly_data(spark)
        # demonstration: Table maintenance
        perform_table_maintenance_operations(spark)
        # demonstration: Schema evolution
        evolve_schema(spark)
    finally:
        # clean-up the demo database
        clean_database(spark)
        spark.stop()

if __name__ == '__main__':
main()
```

Parameter values:

- <cos_bucket_endpoint> : Provide the **Metastore host** value. For more information, see storage details.
- <access_key> : Provide the `access_key_id`. For more information, see storage details.
- <secret_key> : Provide the `secret_access_key`. For more information, see storage details.

3. Upload the Python file to the Cloud Object Storage (You application file reside in Cloud Object Storage). For more information about uploading data, see Upload data.
4. Generate the IAM token for the IBM Analytics Engine token. For more information about how to generate an IAM token, see IAM token.
5. Run the following curl command to submit the Spark application.

```
curl https://api.<region>.ae.cloud.ibm.com/v3/analytics_engines/<iae-instance-guid>/
spark_applications -H "Authorization: Bearer <iam-bearer-token>" -X POST -d '{
  "application_details": {
      "application": "s3a://<application_bucket>/lakehouse-hms-test-cloud-doc-sample.py",
      "conf": {
          "spark.hadoop.fs.s3a.bucket.<application-bucket>.endpoint": "https://
s3.publicendpoint.us-south.cloud-object-storage.appdomain.cloud",
          "spark.hadoop.fs.s3a.bucket.<application-bucket>.access.key":
"<hmac_access_key_for_application-bucket>",
          "spark.hadoop.fs.s3a.bucket.<application-bucket>.secret.key":
"<hmac_secret_key_for_application-bucket>"
      }
  }
  }'
```

Parameter values:

- <region> : Region where Spark instance is provisioned.
- <iae-instance-guid>: The unique ID of the Spark engine. You can get the instance id from the Spark engine details window in the UI.
- <iam-bearer-token> : Generate IAM token. For more information about how to generate an IAM token, see IAM token
- <application_bucket> : Name of the Object Storage bucket. For more information, see storage details.
- <application_bucket_endpoint>: Public endpoint of the Object Storage bucket. For more information, see Service credentials.
- <hmac_access_key_for_application-bucket> : Access key for the Cloud Object Storage. For more information, see Using HMAC credentials.
- <hmac_secret_key_for_application-bucket> : Secret key for the Cloud Object Storage. For more information, see Using HMAC credentials.

Running Spark notebook from Watson Studio on Cloud

The topic provides the procedure to run a sample Spark application by using Watson Studio notebooks. The notebook resides in a Watson Studio project that is available in IBM Cloud.

You can download and run the Spark use case sample in Watson Studio to explore the following functions in watsonx.data:

- Accessing tables
- Loading data
- Modifying schema

- Performing table maintenance activities

Note: Watson Studio provides sample note books that allow to run small pieces of code that process your data, and immediately view the results of your computation. The notebook includes a sample use case that the users can readily download and start working on.

watsonx.data on Red Hat OpenShift

Before you begin

- Install Watson Studio on watsonx.data cluster (if your Watson Studio project is on watsonx.data cluster).
- Download the sample notebook.
- **Retrieve watsonx.data credentials**

  Get the following information from watsonx.data:

  - `<wxd_hms_endpoint>` : Thrift endpoint. For example, `thrift://<id>.databases.appdomain.cloud:32683`. To get the details, log in to your watsonx.data instance, Click on the **Iceberg data** catalog from **Infrastructure manager**. In the **Details** tab, copy **Metastore host**, which is your <wxd_hms_endpoint>.
  - `<wxd_hms_username>` : Username for watsonx.data instance. For IBM Cloud, the username is by default `ibmlhapikey`.
  - `<wxd_hms_password>` : Hive Metastore (HMS) password. Get the API key from the watsonx.data administrator.
- Source bucket details: If you bring your own Jupiter notebook, you must require the following details of your source bucket where data resides.
  - `<source_bucket_endpoint>` : Endpoint of the source bucket. For example, for a source bucket in Dallas region, the endpoint is `s3.direct.us-south.cloud-object-storage.appdomain.cloud`. Use public endpoint.
  - `<source_bucket_access_key>` : Access key of the source bucket.
  - `<source_bucket_secret_key>` : Secret key of the source bucket.

About this task

To run the Spark sample notebook, follow the steps:

Procedure

1. Integrate watsonx.data and your Watson Studio Cloud. To do that:
   a. Expose the Hive Metastore (HMS) by using a `NodePort` service. For more information, see ../../wxd/admin/expose_hms.dita.
   b. Import a self-signed certificate. For more information, see Importing self-signed certificates from a Hive metastore server to a Java truststore.
2. Log in to your Cloud Watson Studio account.
3. Create a project. For more information, see Creating a project.
4. Select the project and add the Jupyter Notebook.
5. Click **New Assets** to create a new asset of Jupyter Notebook. The **New Assets** page opens. For more information, see Creating notebooks.
6. Click **Code editors**.
7. Search and select **Jupyter Notebook editor**. The **New notebook** page opens.
8. Specify the following details:
   a) Name: Type the name of the notebook.
   b) Select the **Spark runtime**. It must be Spark 3.3 or Spark 3.4 with Python 10.8.

9. Upload and run IBM published Spark notebook. Follow the steps:
   a. From the left window, click **Local file**.
   b. In the **Notebook file** field, drag the IBM Spark notebook file (that is provided by IBM) from your local computer.
   c. Update the watsonx.data credentials, source bucket and catalog bucket details in the **Configuring IBM Analytics Engine** section in the notebook.
10. Click **Create**. The uploaded notebook opens.
11. You can step through the notebook execution cell by cell, by selecting **Shift-Enter** or you can run the entire notebook by clicking **Run All** from the menu.

Working with Spark on Cloud Pak for Data

The topic provides the procedure to integrate watsonx.data with Spark with on Cloud Pak for Data and submit a sample application.
You can use Spark that is available on the same Cloud Pak for Data cluster where watsonx.data is installed or on a different Cloud Pak for Data cluster.

Note: If you use Spark on a different CPD cluster, make sure that you do the additional procedure.

watsonx.data on Red Hat OpenShift

Before you begin

- Install Cloud Pak for Data with Spark.
- If you are using Spark that belongs to a different Cloud Pak for Data cluster (from watsonx.data cluster), you must do the following:
  1. Expose the Hive Metastore (HMS) by using a `NodePort` service. For more information, see Expose HMS.
  2. Import a self-signed certificate. For more information, see Importing self-signed certificates from a Hive metastore server to a Java truststore.

About this task

To configure an Analytics Engine powered by Spark instance for watsonx.data:

Procedure

1. Configure your Analytics Engine powered by Apache Spark instance with your watsonx.data instance:
   a. Generate an access token to set the Analytics Engine powered by Apache Spark instance default configuration. See Generating an authorization token or API key.
   b. Run the API to set instance default configuration:

```
curl -X PATCH --location --header "Authorization: Bearer {ACCESS_TOKEN}" --header
"Accept: application/json" --header "Content-Type: application/merge-patch+json" --data '{
<CONFIGURATION_DETAILS>
}' "<https://<CloudPakforData_URL>/v4/analytics_engines/<INSTANCE_ID>/default_configs"
```

2. CONFIGURATION_DETAILS: Copy the following configuration details and substitute the following values:
   - `<hms-thrift-endpoint-from-watsonx.data>`: The HMS endpoint from the watsonx.data instance. HMS URL from watsonx.data. For example (thrift://<id>.databases.appdomain.cloud:32683). To get the details, log in to your watsonx.data instance, Click on the **Iceberg data** catalog from **Infrastructure manager**. In the **Details** tab, copy **Metastore host**, which is your <hms-thrift-endpoint-from-watsonx.data>.
   - `<hms-user-from-watsonx.data>`: Cloud Pak for Data cluster admin username . Cloud Pak for Data cluster admin user.

- `<hms-password-from-watsonx.data>`: Cloud Pak for Data cluster admin password. Cloud Pak for Data cluster admin password.

```
{
"spark.sql.catalogImplementation": "hive",
"spark.driver.extraClassPath": "/opt/ibm/connectors/iceberg-lakehouse/iceberg-3.3.2-1.2.1-
hms-4.0.0-shaded.jar",
"spark.sql.extensions": "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
"spark.sql.iceberg.vectorization.enabled": "false",
"spark.sql.catalog.lakehouse": "org.apache.iceberg.spark.SparkCatalog",
"spark.sql.catalog.lakehouse.type": "hive",
"spark.sql.catalog.lakehouse.uri": "thrift://<hms-thrift-endpoint-from-watsonx.data>",
"spark.hive.metastore.client.auth.mode": "PLAIN",
"spark.hive.metastore.client.plain.username": "<hms-user-from-watsonx.data>",
"spark.hive.metastore.client.plain.password": "<hms-password-from-watsonx.data>",
"spark.hive.metastore.use.SSL": "true",
"spark.hive.metastore.truststore.type": "JKS",
"spark.hive.metastore.truststore.path" : "file:///opt/ibm/jdk/lib/security/cacerts"
"spark.hive.metastore.truststore.password" : "changeit"
}
```

3. After you configure Analytics Engine, you can submit the Spark application. For more information, see Run Spark use case.

Running Spark application

You can run Spark use cases for watsonx.data by using Python samples. All the samples are written by using Spark Python APIs.

Before you begin

Before you can run Spark use cases for watsonx.data, you must:

- Install IBM watsonx.data instance.
- Establish connection with Spark .
- To enable your Spark application to work with the watsonx.data catalog and storage, you must have `Metastore admin` role. Without `Metastore admin` privilege, you cannot ingest data to storage using Native Spark engine. For more information about the Spark configuration, see Working with the watsonx.data catalog and storage.

About this task

About the sample use case

The sample file demonstrates the following functions:

- **Accessing tables from** watsonx.data

  : The **Create a database in Lakehouse catalog** section from the sample use case creates a database **demodb** in the configured watsonx.data instance with a catalog named **lakehouse**. **demodb** is configured to store all the data and metadata under the Cloud Object Storage bucket **lakehouse-bucket**. It also creates an iceberg table **testTable** and accesses it.

- **Ingesting data to** watsonx.data

  : The **Ingest parquet data into a lakehouse table** section from the sample use case allows you to ingest data in parquet and CSV format from a **spark-vol** into a watsonx.data table. Sample data in parquet format is inserted from the volume into the watsonx.data table **yellow_taxi_2022** . The **Ingest parquet data into a lakehouse table** section from the sample python file also shows ingesting data in CSV format from the **spark-vol** into the table **zipcode** in the database **demodb**. For more information, see Inserting sample data into the Cloud Object Storage.

- **Modifying the schema in** watsonx.data

  : The **Schema evolution** section from the sample use case allows you to modify data in watsonx.data.

- **Table maintenance activities in** watsonx.data

: Table maintenance helps in keeping the watsonx.data table performance. Iceberg provides table maintenance procedures out of the box that allows performing powerful table optimizations in a declarative fashion. The following sample demonstrates how to do some table maintenance operations by using Spark. For more information about the Iceberg Spark table maintenance operations, see Table Operations.

To run Spark use cases for watsonx.data by using sample Python files:

Procedure

1. To insert data to Cloud Object Storage, for the following steps.

   Inserting sample data into the Cloud Object Storage

   a. Create a Cloud Object Storage (for example, source-bucket) to store sample data to be ingested into watsonx.data instance. For information about creating storage volume, see Getting started with IBM Cloud Object Storage.

      As a user of Object Storage (storage volume) , you not only need to know the API key or the HMAC keys to configure Object Storage, but also the IBM Analytics Engine service endpoints to connect to Object Storage. See Selecting regions and endpoints for more information on the endpoints to use based on your Object Storage bucket type, such as regional versus cross-regional. You can also view the endpoints across regions for your Object Storage service by selecting the service on your IBM Cloud dashboard and clicking **Endpoint** in the navigation pane.

   b. Download sample CSV file (for example, zipcodes.csv) and parquet sample data (for example, six months taxi data for year 2022) from the following links.

      - Sample parquet file

      - Sample CSV file

   c. Install IBM Cloud Object Storage plug-in. For more information about how to install plug-in, see IBM Cloud Object Storage CLI.

   d. Use the COS CLI to upload the sample data into Cloud Object Storage bucket.

      ```
      ibmcloud cos upload --bucket <cos_bucket_name> --key <source_file_name> --file
      <path_to_source_file>
      ```

      Parameter values:

      - <cos_bucket_name>: name of the storage volume.

      - <source_file_name>: the name of the sample data file that you downloaded. Here, `key zipcodes.csv` is the file name (see the following example).

      - <path_to_source_file>: the path to the location in your machine where the file resides. Here, `path/zipcodes.csv` is the file path (see the following example).

      For example:

      ```
      ibmcloud cos upload --bucket source-bucket --key zipcodes.csv --file <path/
      zipcodes.csv>
      ```

2. Save the following sample as a Python file:

   ```
   from pyspark.sql import SparkSession
   import os

   def init_spark():
       spark = SparkSession.builder \
           .appName("lh-hms-cloud") \
           .config("spark.hadoop.fs.s3a.bucket.lakehouse-
   bucket.endpoint" ,"<cos_bucket_endpoint>") \
           .config("spark.hadoop.fs.s3a.bucket.lakehouse-bucket.access.key" ,"<access_key>") \
           .config("spark.hadoop.fs.s3a.bucket.lakehouse-bucket.secret.key" ,"<secret_key>") \
           .enableHiveSupport() \
           .getOrCreate()
   ```

```
    return spark

def create_database(spark):
    # Create a database in the lakehouse catalog
    spark.sql("create database if
not exists lakehouse.demodb LOCATION 's3a://lakehouse-bucket/'")

def list_databases(spark):
    # list the database under lakehouse catalog
    spark.sql("show databases from lakehouse").show()

def basic_iceberg_table_operations(spark):
    # demonstration: Create a basic Iceberg table, insert some data and then query table
    spark.sql("create table if not exists lakehouse.demodb.testTable(id
INTEGER, name VARCHAR(10), age INTEGER, salary DECIMAL(10, 2)) using iceberg").show()
    spark.sql("insert into lakehouse.demodb.testTable
values(1,'Alan',23,3400.00),(2,'Ben',30,5500.00),(3,'Chen',35,6500.00)")
    spark.sql("select * from lakehouse.demodb.testTable").show()

def create_table_from_parquet_data(spark):
    # load parquet data into dataframe
    df = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-01.parquet")
    # write the dataframe into an Iceberg table
    df.writeTo("lakehouse.demodb.yellow_taxi_2022").create()
    # describe the table created
    spark.sql('describe table lakehouse.demodb.yellow_taxi_2022').show(25)
    # query the table
    spark.sql('select * from lakehouse.demodb.yellow_taxi_2022').count()

def ingest_from_csv_temp_table(spark):
    # load csv data into a dataframe
    csvDF = spark.read.option("header",True).csv("file:///spark-vol/zipcodes.csv")
    csvDF.createOrReplaceTempView("tempCSVTable")
    # load temporary table into an Iceberg table
    spark.sql('create or replace
table lakehouse.demodb.zipcodes using iceberg as select * from tempCSVTable')
    # describe the table created
    spark.sql('describe table lakehouse.demodb.zipcodes').show(25)
    # query the table
    spark.sql('select * from lakehouse.demodb.zipcodes').show()

def ingest_monthly_data(spark):
    df_feb = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-02.parquet")
    df_march = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-03.parquet")
    df_april = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-04.parquet")
    df_may = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-05.parquet")
    df_june = spark.read.option("header",True).parquet("file:///
spark-vol/yellow_tripdata_2022-06.parquet")
    df_q1_q2 = df_feb.union(df_march).union(df_april).union(df_may).union(df_june)
    df_q1_q2.write.insertInto("lakehouse.demodb.yellow_taxi_2022")

def perform_table_maintenance_operations(spark):
    # Query the metadata files table to list underlying data files
    spark.sql("SELECT file_path, file_size_in_bytes
FROM lakehouse.demodb.yellow_taxi_2022.files").show()
    # There are many smaller files compact them into files of 200MB each using the
    # `rewrite_data_files` Iceberg Spark procedure
    spark.sql(f"CALL lakehouse.system.rewrite_data_files(table
=> 'demodb.yellow_taxi_2022', options => map('target-file-size-bytes','209715200'))").show()
    # Again, query the
metadata files table to list underlying data files; 6 files are compacted
    # to 3 files
    spark.sql("SELECT file_path, file_size_in_bytes
FROM lakehouse.demodb.yellow_taxi_2022.files").show()
    # List all the snapshots
    # Expire earlier snapshots. Only latest one with compacted data is required
    # Again, List all the snapshots to see only 1 left
    spark.sql("SELECT committed_at, snapshot_id,
operation FROM lakehouse.demodb.yellow_taxi_2022.snapshots").show()
    #retain only the latest one
    latest_snapshot_committed_at = spark.sql("SELECT committed_at, snapshot_id,
operation FROM lakehouse.demodb.yellow_taxi_2022.snapshots").tail(1)[0].committed_at
    print (latest_snapshot_committed_at)
    spark.sql(f"CALL lakehouse.system.expire_snapshots(table
=> 'demodb.yellow_taxi_2022',older_than => TIMESTAMP
'{latest_snapshot_committed_at}',retain_last => 1)").show()
    spark.sql("SELECT committed_at, snapshot_id,
operation FROM lakehouse.demodb.yellow_taxi_2022.snapshots").show()
    # Removing Orphan data files
    spark.sql(f"CALL lakehouse.system.remove_orphan_files(table
=> 'demodb.yellow_taxi_2022')").show(truncate=False)
    # Rewriting Manifest Files
    spark.sql(f"CALL lakehouse.system.rewrite_manifests('demodb.yellow_taxi_2022')").show()

def evolve_schema(spark):
    # demonstration: Schema evolution
    # Add column fare_per_mile to the table
    spark.sql('ALTER TABLE lakehouse.demodb.yellow_taxi_2022
ADD COLUMN(fare_per_mile double)')
    # describe the table
    spark.sql('describe table lakehouse.demodb.yellow_taxi_2022').show(25)

def clean_database(spark):
    # clean-up the demo database
    spark.sql('drop table if exists lakehouse.demodb.testTable purge')
    spark.sql('drop table if exists lakehouse.demodb.zipcodes purge')
    spark.sql('drop table if exists lakehouse.demodb.yellow_taxi_2022 purge')
    spark.sql('drop database if exists lakehouse.demodb cascade')

def main():
    try:
        spark = init_spark()
        create_database(spark)
        list_databases(spark)
        basic_iceberg_table_operations(spark)
        # demonstration: Ingest parquet and csv data into a watsonx.data Iceberg table
        create_table_from_parquet_data(spark)
        ingest_from_csv_temp_table(spark)
        # load data for
the month of February to June into the table yellow_taxi_2022 created above
        ingest_monthly_data(spark)
        # demonstration: Table maintenance
        perform_table_maintenance_operations(spark)
        # demonstration: Schema evolution
        evolve_schema(spark)
    finally:
        # clean-up the demo database
        clean_database(spark)
        spark.stop()

if __name__ == '__main__':
main()
```

Parameter values:

- <cos_bucket_endpoint> : Provide the **Metastore host** value. For more information, see storage details.
- <access_key> : Provide the `access_key_id`. For more information, see storage details.
- <secret_key> : Provide the `secret_access_key`. For more information, see storage details.

3. Upload the Python file to the Cloud Object Storage (You application file reside in Cloud Object Storage). For more information about uploading data, see Upload data.
4. Generate the API authorization token for the IBM Analytics Engine. For more information about how to generate the token, see Generating an API authorization token.
5. Run the following curl command to submit the Spark application.

```
curl -k -X POST <V4_JOBS_API_ENDPOINT> -H "Authorization: ZenApiKey ${TOKEN}" -d '{
{
  "application_details": {
    "application": "cos://<BUCKET_NAME>.<COS_SERVICE_NAME>/<OBJECT_NAME>",
    "arguments": [
      "cos://<BUCKET_NAME>.<COS_SERVICE_NAME>/<OBJECT_NAME>"
    ],
    "class": "<main_class>",
    "conf": {
      "spark.app.name": "MyJob",
      "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.endpoint": "<COS_ENDPOINT>",
      "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.secret.key": "<COS_SECRET_KEY>",
      "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.access.key": "<COS_ACCESS_KEY>"
    }
  }
}
```

Parameter values:

- <V4_JOBS_API_ENDPOINT>: The endpoint for the instance that you want to use to submit your Spark job. To get the Spark jobs endpoint for your provisioned instance, see Administering the service instance.
- <TOKEN>: To get the access token for your service instance, see Generating an API authorization token.
- <OBJECT_NAME>: The IBM Cloud Object Storage name.
- <BUCKET_NAME>: The storage bucket where the application file resides.
- <COS_SERVICE_NAME>: The Cloud object Storage service name.

Working with the watsonx.data catalog and storage

To enable your Spark application to work with the watsonx.data catalog and storage, add the following configuration to your application payload:

```
spark.hive.metastore.client.plain.username=ibmlhapikey
spark.hive.metastore.client.plain.password=<api-key-of-the-user-which-has-metastore-
admin-role>
spark.hadoop.wxd.apiKey=Basic base64(ibmlhapikey_ibmcloudid:apikey)
```

Running Spark notebook from Watson Studio on Cloud Pak for Data

The topic provides the procedure to run a sample Spark application by using Watson Studio notebooks. The notebook resides in a Watson Studio project that is available IBM Cloud Pak for Data (CPD) cluster.

You can download and run the Spark use case sample in Watson Studio to explore the following functions in watsonx.data:

- Accessing tables
- Loading data
- Modifying schema
- Performing table maintenance activities

Note: Watson Studio provides sample note books that allow to run small pieces of code that process your data, and immediately view the results of your computation. The notebook includes a sample use case that the users can readily download and start working on.

watsonx.data on Red Hat OpenShift

Before you begin

- Install Watson Studio on the CPD cluster.
- **Retrieve watsonx.data credentials**

  Get the following information from watsonx.data:

  - <wxd_hms_endpoint> : Thrift endpoint. For example, thrift://81823aaf-8a88-4bee-a0a1-6e76a42dc833.cfjag3sf0s5o87astjo0.databases.appdomain.cloud:32683. To get the details, log in to your watsonx.data instance, Click on the **Iceberg data** catalog from **Infrastructure manager**. In the **Details** tab, copy **Metastore host**, which is your <wxd_hms_endpoint>.
  - <wxd_hms_username> : Username for watsonx.data instance. For IBM Cloud Pak for Data, the username of the user with Metastore admin role. For more information, see Managing access to the Hive Metastore.
  - <wxd_hms_password> : Hive Metastore (HMS) password. Get the password from the watsonx.data administrator.
- Source bucket details: If you bring your own Jupiter notebook, you must require the following details of your source bucket where data resides.

  - `<source_bucket_endpoint>` : Endpoint of the source bucket. For example, for a source bucket in Dallas region, the endpoint is `s3.direct.us-south.cloud-object-storage.appdomain.cloud`. Use public endpoint.
  - `<source_bucket_access_key>` : Access key of the source bucket.
  - `<source_bucket_secret_key>` : Secret key of the source bucket.
- Download the sample notebook.

About this task

To run the Spark sample notebook, follow the steps:

Procedure

1. Log in to your Watson Studio account in IBM Cloud Pak for Data cluster.
2. Create a project. For more information, see Creating a project.
3. Select the project and add the Jupyter Notebook.
4. Click **New Assets** to create a new asset of Jupyter Notebook. The **New Assets** page opens. For more information, see Creating notebooks.
5. Click **Code editors**.
6. Search and select **Jupyter Notebook editor**. The **New notebook** page opens.
7. Specify the following details:
   a) Name: Type the name of the notebook.
   b) Select the **Spark runtime**. It must be Spark 3.3 or Spark 3.4 with Python 10.8.
8. Upload and run IBM published Spark notebook. Follow the steps:
   a. From the left window, click **Local file**.
   b. In the **Notebook file** field, drag the IBM Spark notebook file (that is provided by IBM) from your local computer.
   c. Update the watsonx.data credentials, source bucket and catalog bucket details in the **Configuring IBM Analytics Engine** section in the notebook.
9. Click **Create**. The uploaded notebook opens.
10. You can step through the notebook execution cell by cell, by selecting **Shift-Enter** or you can run the entire notebook by clicking **Run All** from the menu.

Working with Apache Hudi catalog

The topic describes the procedure to run a Spark application that ingests data into an Apache Hudi catalog.

watsonx.data on Red Hat OpenShift

Procedure

1. Create a storage with Apache Hudi catalog to store data used in the Spark application. To create storage with Apache Hudi catalog, see Adding a storage-catalog pair.
2. Associate the storage with the external Spark engine. For more information, see Associating a catalog with an engine.
3. Create Cloud Object Storage (COS) to store the Spark application. To create Cloud Object Storage and a bucket, see Creating a storage bucket. You can also use other storage types such as IBM Storage Ceph, Amazon S3, or Min IO.
4. Register the Cloud Object Storage in watsonx.data. For more information, see Adding a storage-catalog pair.
5. Save the following Spark application (Python file) to your local machine. Here, `hudi_demo.py`.

   The Python Spark application demonstrates the following functionality:

- It creates a database inside the Apache Hudi catalog (that you created to store data). Here, `hudi_db`.
- It creates a table inside the `hudi_db` database, namely `hudi_table`.
- It inserts data into the `hudi_table` and does SELECT query operation.
- It drops the table and schema after use.

```
from pyspark.sql import SparkSession

def init_spark():
    spark = SparkSession.builder \
        .appName("CreateHudiTableInCOS") \
        .enableHiveSupport() \
        .getOrCreate()
    return spark

def main():

    try:
        spark = init_spark()
        spark.sql("show databases").show()
        spark.sql("create database if not exists spark_catalog.hudi_db LOCATION 's3a://hudi-
connector-test/'").show()
        spark.sql("create table if not exists spark_catalog.hudi_db.hudi_table (id
bigint, name string, location string) USING HUDI OPTIONS ('primaryKey' 'id',
hoodie.write.markers.type= 'direct', hoodie.embed.timeline.server= 'false')").show()
        spark.sql("insert into hudi_db.hudi_table VALUES (1, 'Sam','Kochi'), (2,
'Tom','Bangalore'), (3, 'Bob','Chennai'), (4, 'Alex','Bangalore')").show()
        spark.sql("select * from spark_catalog.hudi_db.hudi_table").show()
        spark.sql("drop table spark_catalog.hudi_db.hudi_table").show()
        spark.sql("drop schema spark_catalog.hudi_db CASCADE").show()

    finally:
        spark.stop()

if __name__ == '__main__':
    main()
```

6. Upload the Spark application to the COS, see Uploading data.
7. To submit the Spark application with data residing in Cloud Object Storage, specify the parameter values and run the following curl command.

```
curl -k -X POST <V4_JOBS_API_ENDPOINT> -H "Authorization: ZenApiKey $<TOKEN>" -d '{ '{
    "application_details": {
        "conf": {
            "spark.serializer" : "org.apache.spark.serializer.KryoSerializer",
            "spark.hadoop.fs.s3a.bucket.<data_storage_name>.endpoint" : "<bucket_endpoint>",
            "spark.hadoop.fs.s3a.bucket.<data_storage_name>.access.key" : "<access_key>",
            "spark.hadoop.fs.s3a.bucket.<data_storage_name>.secret.key" : "<secret_key>",
            "spark.hadoop.fs.s3a.path.style.access" : "true",
            "spark.hadoop.fs.s3a.impl" : "org.apache.hadoop.fs.s3a.S3AFileSystem",
            "spark.hive.metastore.uris" : "<thrift_url_catalog>",
            "spark.hive.metastore.use.SSL" : "true",
            "spark.hive.metastore.truststore.path" : "file:///opt/ibm/jdk/lib/security/
cacerts",
            "spark.hive.metastore.truststore.password" : "changeit",
            "spark.hive.metastore.truststore.type" : "JKS",
            "spark.hive.metastore.client.auth.mode" : "PLAIN",
            "spark.hive.metastore.client.plain.username" : "<wxd_log_in_username>",
            "spark.hive.metastore.client.plain.password" : "<wxd_log_in_password>",
            "spark.driver.extraJavaOptions" :
"-Dcom.sun.jndi.ldap.object.disableEndpointIdentification=true
-Djdk.tls.trustNameService=true",
            "spark.executor.extraJavaOptions" :
"-Dcom.sun.jndi.ldap.object.disableEndpointIdentification=true
-Djdk.tls.trustNameService=true",
            "spark.hadoop.hive.metastore.schema.verification" : "false",
            "spark.hadoop.hive.metastore.schema.verification.record.version" : "false",
        "spark.sql.extensions": "org.apache.spark.sql.hudi.HoodieSparkSessionExtension",
           "spark.kryo.registrator": "org.apache.spark.HoodieSparkKryoRegistrar",
           "spark.sql.catalog.spark_catalog.type": "hudi",
            "spark.sql.catalog.spark_catalog":
"org.apache.spark.sql.hudi.catalog.HoodieCatalog"

        },
        "application": "s3a://<data_storage_name>/hudi_final_cas.py"
    }
}
```

Parameter values:

- *<V4_JOBS_API_ENDPOINT>*: The endpoint for the instance that you want to use to submit your Spark job. To get the Spark jobs endpoint for your provisioned instance, see Administering the service instance.
- *<TOKEN>*: To get the access token for your service instance, see Generating an API authorization token.
- *<data_storage_name>*: The name of the storage bucket that stores data.
- *<bucket_endpoint>* : Provide the **Metastore host** value of the storage.
- *<access_key>* : Provide the `access_key_id`. Provide the `access_key_id` of the storage.
- *<secret_key>* : Provide the `secret_access_key`. Provide the `secret_access_key` of the storage.
- *<wxd-user-name>*: Your user name credential for the watsonx.data cluster. Note: You must have `Metastore Admin` access on Hive Metastore. For more information, see Managing access to the Hive Metastore (HMS).
- *<wxd-user-password>*: Your password for the watsonx.data cluster.

Working with Delta Lake catalog

The topic describes the procedure to run a Spark application that ingests data into a Delta Lake catalog.

watsonx.data on Red Hat OpenShift

Procedure

1. Create a storage with Delta Lake catalog to ingest and manage data in delta table format. To create storage with Delta Lake catalog, see Adding a storage-catalog pair.“Adding a storage-catalog pair” on page 306.
2. Associate the storage with the external Spark engine. For more information, see Associating a catalog with an engine.
3. Create Cloud Object Storage (COS) to store the Spark application. To create Cloud Object Storage and a bucket, see Creating a storage bucket. You can also use other storage types such as IBM Storage Ceph, Amazon S3, or Min IO.
4. Register the Cloud Object Storage in watsonx.data. For more information, see Adding a storage-catalog pair.
5. Save the following Spark application (Python file) to your local machine. Here, `delta_demo.py`.

   The Python Spark application demonstrates the following functionality:

   - It creates a database inside the Delta Lake catalog (that you created to store data). Here, `iae`.
   - It creates a table inside the `iae` database, namely `employee`.
   - It inserts data into the `employee` and does SELECT query operation.
   - It drops the table and schema after use.

```
from pyspark.sql import SparkSession
import os

def init_spark():
    spark = SparkSession.builder.appName("lh-hms-cloud")\
    .enableHiveSupport().getOrCreate()

    return spark

def main():
    spark = init_spark()
    spark.sql("show databases").show()
    spark.sql("create database if not exists spark_catalog.iae LOCATION 's3a://delta-
connector-test/'").show()
    spark.sql("create table if not exists spark_catalog.iae.employee (id bigint, name
string, location string) USING DELTA").show()
    spark.sql("insert into spark_catalog.iae.employee VALUES (1, 'Sam','Kochi'), (2,
'Tom','Bangalore'), (3, 'Bob','Chennai'), (4, 'Alex','Bangalore')").show()
    spark.sql("select * from spark_catalog.iae.employee").show()
    spark.sql("drop table spark_catalog.iae.employee").show()
    spark.sql("drop schema spark_catalog.iae CASCADE").show()
    spark.stop()

if __name__ == '__main__':
    main()
```

6. Upload the Spark application to the COS, see Uploading data.
7. To submit the Spark application with data residing in Cloud Object Storage, specify the parameter values and run the following curl command.

```
curl -k -X POST <V4_JOBS_API_ENDPOINT> -H "Authorization: ZenApiKey $<TOKEN>" -d '{ {
    "application_details": {
    "conf": {
            "spark.hadoop.fs.s3a.bucket.<data_storage_name>.access.key" : "<access_key>",
            "spark.hadoop.fs.s3a.bucket.<data_storage_name>.secret.key" : "<secret_key>",
            "spark.hadoop.fs.s3a.bucket.<data_storage_name>.endpoint": "<bucket_endpoint>",
            "spark.sql.catalogImplementation" : "hive",
            "spark.sql.extensions" : "io.delta.sql.DeltaSparkSessionExtension",
            "spark.serializer" : "org.apache.spark.serializer.KryoSerializer",
            "spark.hadoop.hive.metastore.schema.verification" : "false",
            "spark.hadoop.hive.metastore.schema.verification.record.version" : "false",
            "spark.hadoop.datanucleus.schema.autoCreateTables" : "false",
            "spark.sql.catalog.spark_catalog" :
"org.apache.spark.sql.delta.catalog.DeltaCatalog",
            "spark.sql.catalog.spark_catalog.type" : "hive",
            "spark.hive.metastore.uris" : "thrift://ibm-lh-lakehouse-hive-metastore-svc.cpd-
instance.svc.cluster.local:9083",
            "spark.hive.metastore.use.SSL" : "true",
            "spark.hive.metastore.truststore.path" : "file:///opt/ibm/jdk/lib/security/
cacerts",
            "spark.hive.metastore.truststore.password" : "changeit",
            "spark.hive.metastore.truststore.type" : "JKS",
            "spark.hive.metastore.client.auth.mode" : "PLAIN",
            "spark.hive.metastore.client.plain.username" : "cpadmin",
            "spark.hive.metastore.client.plain.password" :
"QQJM8GWsiU1m9XAwLoYiaZIMLdqC8q5Z",
            "spark.hadoop.fs.s3a.path.style.access" : "true"
    },

    "application": "s3a://delta-connector-test/delta_demo.py"
    }
}
```

- *<V4_JOBS_API_ENDPOINT>*: The endpoint for the instance that you want to use to submit your Spark job. To get the Spark jobs endpoint for your provisioned instance, see Administering the service instance.
- *<TOKEN>*: To get the access token for your service instance, see Generating an API authorization token.
- *<data_storage_name>*: The name of the storage bucket that stores data.
- *<bucket_endpoint>* : Provide the **Metastore host** value of the storage.
- *<access_key>* : Provide the `access_key_id`. Provide the `access_key_id` of the storage.
- *<secret_key>* : Provide the `secret_access_key`. Provide the `secret_access_key` of the storage.
- *<wxd-user-name>*: Your user name credential for the watsonx.data cluster. Note: You must have `Metastore Admin` access on Hive Metastore. For more information, see Managing access to the Hive Metastore (HMS).
- *<wxd-user-password>*: Your password for the watsonx.data cluster.

Working with Spark on AWS EMR

The topic provides the procedure to run Spark applications from Amazon Web Services Elastic MapReduce (AWS EMR) to achieve the watsonx.data
Spark use cases:

- Data ingestion
- Data querying
- Table maintenance

watsonx.data on Red Hat OpenShift

Before you begin

- Create a catalog with S3 bucket inside watsonx.data.
- Get S3 bucket credentials.
- Set up EMR cluster on AWS. For more information, see Setting up an EMR Cluster.
- Fetch the following information from watsonx.data:
  - HMS URL from watsonx.data. For example (thrift://81823aaf-8a88-4bee-a0a1-6e76a42dc833.cfjag3sf0s5o87astjo0.databases.appdomain.cloud:32683).
  - HMS Credentials from watsonx.data. For more information about getting the HMS credentials, see Managing access to the Hive Metastore.

About this task

To work with source data that resides in AWS S3 buckets, use one of the following options:

- Option 1: Setup watsonx.data on AWS.
- Option 2: Configure watsonx.data to include an AWS S3 bucket-based catalog.

The watsonx.data query engines can run queries on data from AWS S3 buckets. In both cases, you can run the data ingestion and schema maintenance operations by using AWS EMR Spark.

Follow the steps to run the Spark sample python file.

Procedure

1. Expose the Hive Metastore (HMS) by using a `NodePort` service. For more information, see ../../wxd/admin/expose_hms.dita.
2. Import a self-signed certificate. For more information, see Importing self-signed certificates from a Hive metastore server to a Java truststore.
3. Connect to the AWS EMR cluster. For more information about using SSH to connect to the EMR cluster, see Setting up Amazon EMR.
4. Save the following sample python file.

```
from pyspark.sql import SparkSession
import os

def init_spark():
spark = SparkSession.builder.appName("lh-hms-cloud")\
.enableHiveSupport().getOrCreate()
return spark

def create_database(spark):
    # Create a database in the lakehouse catalog
    spark.sql("create database if not exists lakehouse.amazonschema LOCATION 's3a://
lakehouse-bucket-amz/'")

def list_databases(spark):
    # list the database under lakehouse catalog
    spark.sql("show databases from lakehouse").show()

def basic_iceberg_table_operations(spark):
    # demonstration: Create a basic Iceberg table, insert some data and then query table
    spark.sql("create table if not exists lakehouse.amazonschema.testTable(id INTEGER, name
VARCHAR(10), age INTEGER, salary DECIMAL(10, 2)) using iceberg").show()
    spark.sql("insert into lakehouse.amazonschema.testTable values(1,'Alan',23,3400.00),
(2,'Ben',30,5500.00),(3,'Chen',35,6500.00)")
    spark.sql("select * from lakehouse.amazonschema.testTable").show()
```

```
def create_table_from_parquet_data(spark):
    # load parquet data into dataframce
    df = spark.read.option("header",True).parquet("s3a://source-bucket-amz/nyc-taxi/
yellow_tripdata_2022-01.parquet")
    # write the dataframe into an Iceberg table
    df.writeTo("lakehouse.amazonschema.yellow_taxi_2022").create()
    # describe the table created
    spark.sql('describe table lakehouse.amazonschema.yellow_taxi_2022').show(25)
    # query the table
    spark.sql('select * from lakehouse.amazonschema.yellow_taxi_2022').count()

def ingest_from_csv_temp_table(spark):
    # load csv data into a dataframe
    csvDF = spark.read.option("header",True).csv("s3a://source-bucket-amz/zipcodes.csv")
    csvDF.createOrReplaceTempView("tempCSVTable")
    # load temporary table into an Iceberg table
    spark.sql('create or replace table lakehouse.amazonschema.zipcodes using iceberg as
select * from tempCSVTable')
    # describe the table created
    spark.sql('describe table lakehouse.amazonschema.zipcodes').show(25)
    # query the table
    spark.sql('select * from lakehouse.amazonschema.zipcodes').show()

def ingest_monthly_data(spark):
    df_feb = spark.read.option("header",True).parquet("s3a://source-bucket-amz//nyc-taxi/
yellow_tripdata_2022-02.parquet")
    df_march = spark.read.option("header",True).parquet("s3a://source-bucket-amz//nyc-taxi/
yellow_tripdata_2022-03.parquet")
    df_april = spark.read.option("header",True).parquet("s3a://source-bucket-amz//nyc-taxi/
yellow_tripdata_2022-04.parquet")
    df_may = spark.read.option("header",True).parquet("s3a://source-bucket-amz//nyc-taxi/
yellow_tripdata_2022-05.parquet")
    df_june = spark.read.option("header",True).parquet("s3a://source-bucket-amz//nyc-taxi/
yellow_tripdata_2022-06.parquet")

    df_q1_q2 = df_feb.union(df_march).union(df_april).union(df_may).union(df_june)
    df_q1_q2.write.insertInto("lakehouse.amazonschema.yellow_taxi_2022")

def perform_table_maintenance_operations(spark):
    # Query the metadata files table to list underlying data files
    spark.sql("SELECT file_path, file_size_in_bytes FROM
lakehouse.amazonschema.yellow_taxi_2022.files").show()

    # There are many smaller files compact them into files of 200MB each using the
    # `rewrite_data_files` Iceberg Spark procedure
    spark.sql(f"CALL lakehouse.system.rewrite_data_files(table =>
'amazonschema.yellow_taxi_2022', options => map('target-file-size-
bytes','209715200'))").show()

    # Again, query the metadata files table to list underlying data files; 6 files are
compacted
    # to 3 files
    spark.sql("SELECT file_path, file_size_in_bytes FROM
lakehouse.amazonschema.yellow_taxi_2022.files").show()

    # List all the snapshots
    # Expire earlier snapshots. Only latest one with comacted data is required
    # Again, List all the snapshots to see only 1 left
    spark.sql("SELECT committed_at, snapshot_id, operation FROM
lakehouse.amazonschema.yellow_taxi_2022.snapshots").show()
    #retain only the latest one
    latest_snapshot_committed_at = spark.sql("SELECT committed_at, snapshot_id, operation
FROM lakehouse.amazonschema.yellow_taxi_2022.snapshots").tail(1)[0].committed_at
    print (latest_snapshot_committed_at)
    spark.sql(f"CALL lakehouse.system.expire_snapshots(table
=> 'amazonschema.yellow_taxi_2022',older_than => TIMESTAMP
'{latest_snapshot_committed_at}',retain_last => 1)").show()
    spark.sql("SELECT committed_at, snapshot_id, operation FROM
lakehouse.amazonschema.yellow_taxi_2022.snapshots").show()

    # Removing Orphan data files
    spark.sql(f"CALL lakehouse.system.remove_orphan_files(table =>
'amazonschema.yellow_taxi_2022')").show(truncate=False)

    # Rewriting Manifest Files
    spark.sql(f"CALL
lakehouse.system.rewrite_manifests('amazonschema.yellow_taxi_2022')").show()


def evolve_schema(spark):
    # demonstration: Schema evolution
    # Add column fare_per_mile to the table
    spark.sql('ALTER TABLE lakehouse.amazonschema.yellow_taxi_2022 ADD COLUMN(fare_per_mile
double)')
    # describe the table
    spark.sql('describe table lakehouse.amazonschema.yellow_taxi_2022').show(25)


def clean_database(spark):
    # clean-up the demo database
    spark.sql('drop table if exists lakehouse.amazonschema.testTable purge')
    spark.sql('drop table if exists lakehouse.amazonschema.zipcodes purge')
    spark.sql('drop table if exists lakehouse.amazonschema.yellow_taxi_2022 purge')
    spark.sql('drop database if exists lakehouse.amazonschema cascade')

def main():
    try:
        spark = init_spark()
        clean_database(spark)

        create_database(spark)
        list_databases(spark)

        basic_iceberg_table_operations(spark)

        # demonstration: Ingest parquet and csv data into a wastonx.data Iceberg table
        create_table_from_parquet_data(spark)
        ingest_from_csv_temp_table(spark)

        # load data for the month of Feburary to June into the table yellow_taxi_2022
created above
        ingest_monthly_data(spark)

        # demonstration: Table maintenance
        perform_table_maintenance_operations(spark)

        # demonstration: Schema evolution
        evolve_schema(spark)
    finally:
        # clean-up the demo database
        #clean_database(spark)
        spark.stop()

if __name__ == '__main__':
  main()
```

5. Run the following commands to download the Hive metastore JAR file from the location to your workstation:

   Note: The JAR file must be present in the /home/hadoop location on all nodes of the cluster. Make a note of the `spark.driver.extraClassPath` and `spark.executor.extraClassPath`.

```
wget https://github.com/IBM-Cloud/IBM-Analytics-Engine/raw/master/wxd-connectors/hms-
connector/hive-exec-2.3.9-core.jar
wget https://github.com/IBM-Cloud/IBM-Analytics-Engine/raw/master/wxd-connectors/hms-
connector/hive-common-2.3.9.jar
wget https://github.com/IBM-Cloud/IBM-Analytics-Engine/raw/master/wxd-connectors/hms-
connector/hive-metastore-2.3.9.jar
```

6. Configure the HMS connection details in the AWS EMR cluster to connect to the watsonx.data Hive Meta Store (HMS). A sample command to use spark-submit from an EMR-6.12.0 (Spark 3.4.1) based cluster is as follows:

   Run the command from EMR on the EC2 cluster to submit the Sample spark job.

```
spark-submit \
--deploy-mode cluster \
--jars https://repo1.maven.org/maven2/org/apache/iceberg/iceberg-
spark-runtime-3.4_2.12/1.4.0/iceberg-spark-runtime-3.4_2.12-1.4.0.jar,/usr/lib/hadoop/hadoop-
aws.jar,/usr/share/aws/aws-java-sdk/aws-java-sdk-bundle*.jar,/usr/lib/hadoop-lzo/lib/* \
--conf spark.sql.catalogImplementation=hive \
--conf spark.driver.extraClassPath=/home/hadoop/hive-common-2.3.9.jar:/home/hadoop/hive-
metastore-2.3.9.jar:/home/hadoop/hive-exec-2.3.9-core.jar \
--conf spark.executor.extraClassPath=/home/hadoop/hive-common-2.3.9.jar:/home/hadoop/hive-
metastore-2.3.9.jar:/home/hadoop/hive-exec-2.3.9-core.jar \
--conf
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions \
--conf spark.sql.iceberg.vectorization.enabled=false \
--conf spark.sql.catalog.lakehouse=org.apache.iceberg.spark.SparkCatalog \
--conf spark.sql.catalog.lakehouse.type=hive \
--conf spark.hive.metastore.uris==<<change_endpoint>> \
--conf spark.hive.metastore.client.auth.mode=PLAIN \
--conf spark.hive.metastore.client.plain.username=ibmlhapikey \
--conf spark.hive.metastore.client.plain.password=<<change_pswd>> \
--conf spark.hive.metastore.use.SSL=true \
--conf spark.hive.metastore.truststore.type=JKS \
--conf spark.hive.metastore.truststore.path=file:///etc/pki/java/cacerts \
--conf spark.hive.metastore.truststore.password=changeit \
amazon-lakehouse.py
```

Parameter values:

- *<<change_endpoint>>* : The Hive meta store URI endpoint to access the meta store. For more information on getting the HMS credentials, see Managing access to the Hive Metastore.
- *<<change_pswd>>* : The password to access the meta store. For more information on getting the HMS credentials, see Managing access to the Hive Metastore.

Note:

To run the Spark python file by using EMR-6.11.1 (Spark 3.3.2) cluster, download the iceberg jars from the location and follow the same procedure.

Native Spark engine

Introduction to native Spark engine

Native Spark engine is a compute engine in IBM watsonx.data. You can use native Spark engine to submit applications that involves complex analytical operations.

In watsonx.data, native Spark engine is used to achieve the following use cases:

- Ingest large volumes of data into watsonx.data tables.
- Handle complex analytical workload.
- Table maintenance operation to enhance watsonx.data performance of the table
- Develop, run, and debug applications written in Python, R and Scala.

Note: You can also register an external Spark engine from the **Add engine** window in watsonx.data UI. External Spark engines reside in an external network or a different namespace to where watsonx.data is deployed. For more information, see Registering an engine.

Provisioning native Spark engine

IBM watsonx.data allows you to provision native Spark engine to run complex analytical workloads.

About this task

To provision a native Spark engine, complete the following steps.

Procedure

1. Log in to watsonx.data console.
2. From the navigation menu, select **Infrastructure Manager**.
3. To provision an engine, click **Add component** and select **IBM Spark**.
4. Click **Next**.
5. In the **Add component - IBM Spark** window, enter the **Display name** for your Spark engine.
6. Select **Create a native Spark engine**, and do the following:
    a. Specify the storage volume that is considered as **Engine home**, which stores the Spark events and logs that are generated while running spark applications. You can either select an existing storage volume or specify details to create a new storage volume. Choose one of the following options:

Note: To store Spark application, create a different storage volume. To create storage volume , see Creating a storage volume.

- **Option1**: Select an existing volume. To do that, specify the following fields:
  - **Existing volume**: Select the option to associate a storage volume that is already available in the cluster. To create storage volume , see Creating a storage volume.
  - **Select volume**: To use an existing volume, select the storage volume from the list.
- **Option2** : Create a new storage volume. To do that, specify the following fields:
  - **New Volume**: Select the option to create a new storage volume and use it.
  - **Volume name**: Enter a name for the new storage volume.
  - **Storage Class**: Select the class to which the storage volume belongs.
  - **Size of the new storage volume**: Slide to select the volume size in GB. You can select values between 5 GB and 1024 GB.

  Restriction: Use storage classes that provision file storage rather than block storage. If you try to use a storage class that provisions block storage, you might encounter an error when you try to create storage volumes.

  Note: You must have user role with the **Create service instances** permission in Cloud Pak for Data to create Storage volumes. If you do not have the permission, the Administrator must create a storage volume and grant you write access permission. To create storage volume and grant access permission, see Creating a storage volume.

b. Select the **Spark runtime version** that must be considered for processing the applications.

c. Select the catalogs that must be associated with the engine from the **Associated catalogs(optional)** field.

7. Click **Create**. An acknowledgment message is displayed.

Managing native Spark engine details

In IBM watsonx.data, you can view and edit the details of a native Spark engine. You can also manage user access and monitor the status of the applications that are submitted.

View and edit native Spark engine details

You can use the IBM watsonx.data UI or API to view and edit the native Spark details.

About this task

Viewing native Spark engine details

You can view the native Spark engine details in list and topology views.

1. Log in to the watsonx.data cluster. Go to the **Infrastructure manager** page.
2. Click the name of Spark engine (either from list or topology view). Engine information window opens.
3. In the **Details** tab, you can view the following details:

Field	Description
Display name	The Spark engine name.
Engine ID	The unique identifier of the Spark instance.
Description	The description of the engine.
Tags	The tag that is specified at the time of registering an engine.

Field	Description
Default Spark version	The Spark runtime version that is used by default for any application that is submitted to the Spark engine.
Volume	The storage volume associated with the Spark engine.
Type	The engine type. Here, IBM Analytics Engine (Spark).
watsonx.data application endpoint	The endpoint is used at the time of application submission. To submit an application by using API, see API Docs.
Spark engine endpoint	The native Spark endpoint.
Default Spark Configuration	The Spark configuration properties that are applied to any application that is submitted to the Spark engine.

Editing Spark details from Console UI

You can edit the Spark details in list and topology views.

1. Log in to the watsonx.data cluster. Go to the **Infrastructure manager** page.
2. Click the name of Spark engine (either from list or topology view). Engine information window opens.
3. In the **Details** tab, click **Edit**.
4. In the **Display name** field, enter the display name for the Spark engine.
5. In the **Description** field, enter the description of the engine or edit the existing description.
6. In the **Tags** field, select the tags from the list or start typing to define a new tag.
7. In the **Default Spark version** field, select the Spark runtime version that must be considered for processing the applications.
8. In the **Default Spark configuration** field, click **Edit configuration** link to update the default Spark configuration. For more information about different properties, see Available Properties.
   - Enter the key-value pair for the Spark configuration that applies to all applications.
   - Click **Apply**.
9. Click **Save** and click the name of Spark engine (either from list or topology view). Engine information window opens.

Editing Spark details by using API

1. Use the following curl command to edit the default Spark configuration and Spark version.

   Note: You can update the curl command to include or modify the description and tag also.

```
curl --request PATCH \
  --url https://<cpd_host_name>/lakehouse/api/v2/spark_engines/<spark_engine_id> \
  --header 'Authorization: Bearer <token>' \
  --header 'Content-Type: application/merge-patch+json' \
  --header 'LhInstanceId: <instance_id>' \
  --data '{
  "engine_details": {
    "default_config": <map_of_spark_properties>,
    "default_version": "<spark_version>"
  }
}'
```

   Use the curl command to update the Spark engine details like tags, description, default configuration, and Spark version.

Following are the details of the parameter values to be used in the curl command.

- *<cpd_host_name>*: The hostname of your Cloud Pak for Data cluster.
- *<spark_engine_id>* : The **Engine ID** of the native Spark engine.
- *<token>* : The bearer token. For more information about generating the token, see Generating a bearer token.
- *<instance_id>* : The instance ID from the watsonx.data cluster instance URL. For example, `1709968977177454`.
- *<map_of_spark_properties>* : Specify the Spark properties in the form of key-value pair ("*<property_name>*": "*<property_value>*") separated by comma.
  - *<property_name>*: The default configuration property name. For more information about different properties, see Available Properties.
  - *<property_value>*: The value that must be configured for the property. For more information about different properties, see Available Properties.
- *<spark_version>*: The Spark runtime. Possible values are 3.3, and 3.4.

Example:

```
curl --request PATCH \
  --url https://<cpd_host_name>/lakehouse/api/v2/spark_engines/<spark_engine_id> \
  --header 'Authorization: Bearer <token>' \
  --header 'Content-Type: application/merge-patch+json' \
  --header 'LhInstanceId: 1709968977177454' \
  --data '{
  "engine_details": {
    "default_config": {
      "spark.driver.cores": "1",
      "spark.driver.memory": "4g"
    },
    "default_version": "3.4"
  }
}'
```

Note: To add new properties to the `default_config` parameter, or update existing properties, specify the property name and value. To delete a property, specify the property name and the set the value as NULL.

Associating or dissociating catalogs

In IBM watsonx.data, you can associate (or dissociate an already associated catalog) a catalog with the Native Spark engine.

About this task

To associate a catalog with an engine, see "Associating a catalog with an engine" on page 302.

Note: When you associate a catalog with a Spark engine, connection properties for the associated catalog are added to the Spark engine's default configuration. Do not overwrite these properties in default configuration manually.

To dissociate a catalog with an engine, see "Dissociating a catalog from an engine" on page 303.

Managing the user access for native Spark engine

You can manage access for a native Spark engine.

About this task

Managing user access

You can edit the Spark details in list and topology views.

1. Log in to the watsonx.data cluster. Go to the **Infrastructure manager** page.

2. Click the name of Spark engine (either from list or topology view). Engine information window opens.
3. In the **Access control** tab, click **Add access**.
4. In the **Add access** window, provide the name of the user and select the role.
5. Click **Add**. The user is added and assigned the role.

   Note: To remove a role, click the overflow menu for the selected user and then select **Remove**.

Infrastructure access

The access control at the infrastructure level allows you to grant access to the engines, catalogs, buckets, and databases. The following table explains the permitted actions for each role.

Table 40. Roles and privileges for a native Spark engine

Privileges	Administrator	Manager	User
Delete an engine	Y	N	N
Grant access	Y	N	N
Revoke access	Y	N	N
Update Spark engine details (tags, description, version and configuration)	Y	Y	N
View an engine	Y	Y	Y
Run workloads against the engine	Y	Y	Y
Start Spark history server	Y	Y	Y
Stop Spark history server	Y	Y	Y
View Spark history UI	Y	Y	Y
View Spark UI	Y	Y	Y
Associate catalog	Y	Y	N
Disassociate catalog	Y	Y	N

Procedure

1. Log in to the watsonx.data console.
2. From the navigation menu, select **Access control**. Under the **Infrastructure** tab, the different components (Engine, Catalog, Bucket, and Database) are displayed in the table.
3. Click the overflow icon in the components row and then click **Manage access**. Alternatively, you can click the **Display name** of the component. The selected component page opens.
4. Under the **Access control** tab, click **Add access**.
5. In the **Add access** window, provide the following details.

Field	Description
Name	You can select individual users or a user group.

Field	Description
Role	Select the role from the drop-down list. You can assign roles based on the component type. For more information, see Roles and privileges.

6. Click **Add**. The user is added and assigned the role.
7. To change the role that is assigned to a user, complete the following steps:
   a) Under the **Infrastructure** tab, click the **Display name** of the component in the table.
   b) The **Access control** tab for selected component opens.
   c) Click the overflow menu for the selected user and then select **Change role**.
   d) In the **Change role** window, select the role from the drop-down list.
   e) Click **Save**.
8. To remove a user for a component, complete the following steps:
   a) Under the **Infrastructure** tab, click the **Display name** of the component in the table.
   b) The **Access control** tab for the selected component opens.
   c) Click the overflow menu for the selected user and then select **Remove**.
   d) In the **Confirm removal** window, click **Remove**.

View and manage applications

You can monitor the status of the applications that are submitted in the IBM watsonx.data instance.

Procedure

Log in to the watsonx.data cluster. Go to the **Infrastructure manager** page.

View details of the submitted Spark applications in watsonx.data

a. In the **Applications** tab, you can view the list of all applications that are submitted to watsonx.data. The tab also displays the details such as the application status, Spark version, creation time, start time, and finish time.

   Note: Application can have one of the following statuses.

   - ACCEPTED
   - FINISHED
   - FAILED
   - ERROR
   - RUNNING
   - SUBMITTED
   - STOPPED
   - WAITING

b. Click the arrow to the left of an application ID in the result list, to view more details like **Spark application ID** and **Application name**.

Note: You can also filter the applications based on status using the **Filter** icon.

Stopping applications in watsonx.data

Important: You can stop only the applications that are in `RUNNING` state.

a. In the **Applications** tab, select the application that you want to stop.
b. Click the overflow menu and select **Stop**. The application status changes to `STOPPED`.

Spark user interface

The Spark user interface (Spark UI) allows you to track the following aspects of a running Spark application.

- Current running stage
- Number of tasks in a stage
- Reason for a longer running stage
- Whether the executors in the application are used optimally
- Inspect memory consumption of the driver and executor

Use the Spark history server to inspect stages of a completed Spark application. To access the Spark history server, see the Access Spark history server.

Procedure

1. Log in to the watsonx.data console.
2. From the navigation menu, select **Infrastructure manager**.
3. Click the name of the Spark engine (from list view or topology view). The engine information window opens.
4. In the **Applications** tab, select an application and click the application **ID** link. The **Spark UI** opens. You can view the following details:
   - The **Event Timeline** displays a graphical view of the timeline and the events.
   - The **Active Jobs** displays the details such as job ID, duration and tasks involved in the application.

Accessing the Spark history server

The Spark history server allows you to view the stages of running and completed Spark applications on a watsonx.data instance.

Procedure

1. Log in to the watsonx.data console.
2. From the navigation menu, select **Infrastructure manager**.
3. Click the name of the Spark engine (from list view or topology view). The engine information window opens.
4. In the **Spark history** tab, click **Start history server**.
5. By default, the Spark history server consumes 1 CPU core and 4 GB of memory while running. To use more resources, enter the values for the **Cores** and **Memory**.
6. Click **Start**.

   The Spark history server is started.
7. Click **View Spark history**. The **History Server** page opens.

   From the **History Server** page you can:
   - View the list of completed Spark application and details such as the application ID, duration, and event log for each application.
   - Download the event log of a Spark application. Click the **Download** link inside the **Event Log** column.
   - View the details of an application. Click the application **ID** link. The **Spark Jobs** page opens. It has the following details:

     –
   - such as the different stages of execution, the storage used, the Spark environment and executor (memory and driver) details.

Stopping the Spark history server

Procedure

1. To stop the history server, go to the **Spark history** tab and click **Stop history server**.
   A confirmation box appears.
2. Click **Stop** to confirm.

Managing resource quota

When you submit Spark applications, the resources that are required by the applications can sometimes exceed the resources that are available for the watsonx.data instance. You can control the maximum CPU and memory resources that are used in a single Spark engine by specifying the quota limits.

For a Spark instance in watsonx.data, you can:

- Set a resource quota for the Spark engine
- View the existing resource quota limits
- View the resource quota usage
- Disable the resource quota limits

Note: When the cumulative resource quota required to process a Spark application exceeds the quota limit, the submitted application is queued until the resource quota becomes available for execution.

watsonx.data on Red Hat OpenShift

Before you begin

Important: The quota limits must be set by the Red Hat OpenShift administrator or Red Hat OpenShift project administrator.

1. Install the **Scheduling service** in the watsonx.data cluster. See Installing shared cluster components for IBM Cloud Pak for Data. Do this at the initial stage of setting up the watsonx.data cluster."
2. Enable the resource quota API in the Analytics Engine service by using the following command:

   ```
   oc patch AnalyticsEngine analyticsengine-sample --namespace ${PROJECT_CPD_INST_OPERANDS} \
   --type merge --patch '{"spec":{"serviceConfig":{"schedulerForQuotaAndQueuing":"ibm-cpd-
   scheduler"}}}'
   ```

   Important: You can set resource quota limits only after the Scheduling service configuration is complete and is available for use with watsonx.data and Analytics Engine services.

Setting resource quota limits for native Spark engine
Using API

1. To set your resource quota limits, use the following command:

   ```
   curl --request PATCH \
     --url https://<wxd_host>/lakehouse/api/v2/spark_engines/<spark_engine_id> \
     --header 'AuthInstanceId: <wxd_instance_id>' \
     --header 'Authorization: Bearer <token>' \
     --header 'Content-Type: application/json' \
     --data '{
     "engine_details": {
       "resource_limit_enabled": true,
       "resource_limits": {
         "cores": "10",
         "memory": "50G"
       }
     }
   }'
   ```

   Here, `cores` and `memory` is the maximum number of cores and memory the Spark engine can consume at any given time.

Using UI

1. Log in to the watsonx.data cluster. Go to the **Infrastructure manager** page.
2. Click the name of the Spark engine (either from list or topology view). Engine information window opens.
3. In the **Details** tab, click **Edit**.
4. Use the **Resource quota limits** toggle switch to enable the setting limits for Spark resource quota.
5. Set the following resource quota information:
   a. In the **CPU limit** field, enter the number of cores required.
   b. In the **Memory limit (GB)** field, enter the amount of memory for the instance.
6. Click **Save**. The engine information window opens. After you set the resource quota information, you can view the **CPU consumption** and **Memory consumption** progress widgets.

Disabling the resource quota

Disable the instance resource quota limits to remove the resource limits on the Spark engine and the Spark resources can run without any restrictions.

Using API

1. To remove your resource quota limits, use the following command:

```
curl --request PATCH \
  --url https://<wxd_host>/lakehouse/api/v2/spark_engines/<spark_engine_id> \
  --header 'AuthInstanceId: <wxd_instance_id>' \
  --header 'Authorization: Bearer <token>' \
  --header 'Content-Type: application/json' \
  --data '{
  "engine_details": {
    "resource_limit_enabled": false
  }
}'
```

Using UI

1. Log in to the watsonx.data cluster. Go to the **Infrastructure manager** page.
2. Click the name of Spark engine (either from list or topology view). Engine information window opens.
3. In the **Details** tab, click **Edit**.
4. Use the **Resource quota limits** toggle switch to disable the resource quota limits.
5. Click **Save** and click the name of Spark engine (either from list or topology view). Engine information window opens.

Viewing the resource quota limits and consumption in the Spark engine

You can retrieve the resource quota limits and the current resource utilization of a Spark engine.

Using API

1. Use the following request command:

```
curl --request GET \
  --url https://<wxd_host>/lakehouse/api/v2/spark_engines/<spark_engine_id> \
  --header 'AuthInstanceId: <wxd_instance_id>' \
  --header 'Authorization: Bearer <token>'
```

Sample response:

```
{
  "actions": [
    "view",
    "use",
    "manage_history_server",
    "view_history_ui",
    "view_ui",
    "update",
    "associate",
    "disassociate",
    "grant",
    "revoke",
    "delete"
  ],
  "associated_catalogs": [],
  "created_by": "cpadmin",
  "created_on": 1717484070,
  "description": "",
  "engine_details": {
    "default_config": {
      "ae.spark.history-server.cores": "1",
      "ae.spark.history-server.memory": "4G",
      "spark.hadoop.wxd.cas.endpoint": "https://ibm-lh-lakehouse-cas-svc.cpd-
instance.svc.cluster.local:3500/cas/v1/signature",
      "spark.hadoop.wxd.instanceId": "1716445297707321",
      "spark.hive.metastore.client.auth.mode": "PLAIN",
      "spark.hive.metastore.truststore.password": "ch****it",
      "spark.hive.metastore.truststore.path": "file:///opt/ibm/jdk/lib/security/cacerts",
      "spark.hive.metastore.truststore.type": "JKS",
      "spark.hive.metastore.uris": "thrift://ibm-lh-lakehouse-hive-metastore-svc.cpd-
instance.svc.cluster.local:9083",
      "spark.hive.metastore.use.SSL": "true",
      "spark.serializer": "org.apache.spark.serializer.KryoSerializer",
      "spark.sql.catalogImplementation": "hive",
      "spark.sql.extensions":
"org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions,io.delta.sql.DeltaSparkSes
sionExtension,org.apache.spark.sql.hudi.HoodieSparkSessionExtension",
      "spark.sql.iceberg.vectorization.enabled": "false",
      "spark.wxd.api.endpoint": "https://lhconsole-api-svc.cpd-
instance.svc.cluster.local:3333"
    },
    "default_version": "3.3",
    "endpoints": {
      "wxd_application_endpoint": "https://cpd-cpd-instance.apps.p500.cp.ibm.com/
lakehouse/api/v2/spark_engines/spark205/applications",
      "wxd_engine_endpoint": "https://cpd-cpd-instance.apps.p500.cp.ibm.com/lakehouse/api/v2/
spark_engines/spark205",
      "wxd_history_server_endpoint": "https://cpd-cpd-instance.apps.p500.cp.ibm.com/
lakehouse/api/v2/spark_engines/spark205/history_server",
      "wxd_history_server_ui_endpoint": "https://cpd-cpd-instance.apps.p500.cp.ibm.com/
lakehouse/api/v2/1716445297707321/spark_engines/spark205/history_server/ui"
    },
    "engine_home_volume": "cpd-instance::ae-vol1",
    "resource_limit_enabled": true,
    "resource_limits": {
      "cores": "10",
      "memory": "50G"
    },
    "resource_utilisation": {
      "cores": "6000m",
      "memory": "6144Mi"
    }
  },
  "engine_display_name": "nat-spark1",
  "engine_id": "spark205",
  "origin": "native",
  "status": "running",
  "tags": [],
  "type": "spark"
}
```

Using UI

1. Log in to the watsonx.data cluster. Go to the **Infrastructure manager** page.
2. Click the name of Spark engine (either from list or topology view). Engine information window opens.
3. In the **Details** tab.
4. You can view the current resource quota limits from the **CPU limit** and **Memory limit (GB)** fields.
5. You can view the current resource consumption of a Spark engine from the **CPU consumption** and **Memory consumption** widgets.

Submitting Spark application by using native Spark engine

You can submit a Spark application by running a CURL command. Complete the following steps to submit a Python application.

Before you begin

To enable your Spark application to work with the watsonx.data catalog and storage, you must have `Metastore admin` role. Without `Metastore admin` privilege, you cannot ingest data to storage using Native Spark engine. To enable your Spark application to work with the watsonx.data catalog and storage, add the following configuration to your application payload:.

```
spark.hive.metastore.client.plain.username=ibmlhapikey
spark.hive.metastore.client.plain.password=<api-key-of-the-user-which-has-metastore-admin-role>
spark.hadoop.wxd.apiKey=Basic base64(ibmlhapikey_ibmcloudid:apikey)
```

Procedure

1. Create a storage volume to store the Spark application and related output.
   - **Option1**: Create a storage volume in Cloud Pak for Data cluster. To create storage volume in Cloud Pak for Data cluster, see Creating a storage volume.
   - **Option2**: Create Cloud Object Storage. To create Cloud Object Storage and a bucket, see Creating a storage bucket.
2. If you use Cloud Object Storage, register the Cloud Object Storage in watsonx.data, register Cloud Object Storage bucket. To register Cloud Object Storage bucket, see Adding bucket catalog pair.
3. Upload the Spark application to the storage volume.
   - If you use Cloud Pak for Data storage volume, see Uploading data.
   - If you use Cloud Object Storage, see Uploading data.
4. If your Spark application resides in Cloud Pak for Data storage volume, specify the parameter values and run the following CURL command to submit the application.

   ```
   curl --request POST \
     --url https://<cpd_host_name>/lakehouse/api/v2/spark_engines/<spark_engine_id>/
   applications \
     --header 'Authorization: Bearer <token>' \
     --header 'Content-Type: application/json' \
     --header 'LhInstanceId: <instance_id>' \
     --data '{
     "application_details": {
       "application": "/myapp/<python file name>"
     },
     "volumes": [
       {
         "name": "cpd-instance::my-vol-1",
         "mount_path": "/myapp"
       }
     ]
   }'
   ```

   Parameter values:

   - *<cpd_host_name>*: The hostname of your Cloud Pak for Data cluster.
   - *<spark_engine_id>* : The **Engine ID** of the native Spark engine.
   - *<token>* : The bearer token. For more information about generating the token, see Generating a bearer token.
   - *<instance_id>* : The instance ID from the watsonx.data cluster instance URL. For example, 1609968577169454.
   - *<python file name>* : The Spark application file name. It must be available in the storage volume.
   - *<my-vol-1>* : The display name of the storage volume.
   - *<python file name>* : The Spark application file name.

Example 2:

Run the following curl command to submit the `word count` application.

```
curl --request POST \
  --url https://<cpd_host_name>/lakehouse/api/v2/spark_engines/<spark_engine_id>/
applications \
  --header 'Authorization: Bearer <token>' \
  --header 'Content-Type: application/json' \
  --header 'LhInstanceId: <instance_id>' \
  --data '{
  "application_details": {
    "application": "/opt/ibm/spark/examples/src/main/python/wordcount.py",
    "arguments": [
"/opt/ibm/spark/examples/src/main/resources/people.txt"
    ]
  }
}'
```

Parameters:

- *<instance_id>* : The instance ID from the watsonx.data cluster instance URL. For example, `1609968577169454`.
- *<cpd_host_name>* : The host name of your watsonx.data cluster.
- *<spark_engine_id>* : The engine ID of the Spark engine.
- *<token>* : The bearer token. For more information about generating the token, see Generating a bearer token.

Example 3:

Run the following curl command to customize the cluster hardware sizes:

```
curl -k -X POST \
--url https://<cpd_host_name>/lakehouse/api/v2/spark_engines/<spark_engine_id>/applications \
 -H "Authorization: ZenApiKey ${TOKEN}" -d '{
    "application_details": {
        "application": "/opt/ibm/spark/examples/jars/spark-examples*.jar",
        "arguments": ["1"],
        "class": "org.apache.spark.examples.SparkPi",
        "conf": {
            "spark.driver.memory": "4G",
            "spark.driver.cores": "1",
            "spark.executor.memory": "4G",
            "spark.executor.cores": "1",
            "ae.spark.executor.count": "1"
        }
    }
}'
```

5. If your Spark application resides in Cloud Object Storage, specify the parameter values and run the following curl command. The following example shows the command to submit `read.py` application.

   Example 1:

```
curl --request POST \
  --url https://<cpd_host_name>/lakehouse/api/v2/spark_engines/<spark_engine_id>/
applications \
  --header 'Authorization: Bearer <token>' \
  --header 'Content-Type: application/json' \
  --header 'LhInstanceId: <instance_id>' \
  --data '{
  "application_details": {
    "application": "s3a://<s3_bucket_name>/cos-read.py",
    "conf": {
        "spark.hadoop.fs.s3a.bucket.<s3_bucket_name>.endpoint": "<cos_endpoint>",
        "spark.hadoop.fs.s3a.bucket.<s3_bucket_name>.access.key": "<s3 bucket HMAC access
key>",
        "spark.hadoop.fs.s3a.bucket.<s3_bucket_name>.secret.key": "<s3 bucket  HMAC secret
key>",
        "spark.app.name": "reader-app"
    }
  }
}'
```

Parameter values:

- *<cpd_host_name>*: The hostname of your Cloud Pak for Data cluster.
- *<spark_engine_id>* : The **Engine ID** of the native Spark engine.
- *<token>* : The bearer token. For more information about generating the token, see Generating a bearer token.
- *<instance_id>* : The instance ID from the watsonx.data cluster instance URL. For example, `1609968977179454`.
- *<COS_bucket_name>* : The name of the Cloud Object Storage.
- *<cos_endpoint>*: The public endpoint of the Cloud Object Storage bucket. For example, `s3.direct.us-south.cloud-object-storage.appdomain.cloud`.
- *<Cloud Object storage HMAC access key>* : The access key for Cloud Object storage. For more information, see Create HMAC credentials using the CLI.
- *<Cloud Object storage HMAC secret key>* : The secret key for Cloud Object storage. For more information, see Create HMAC credentials using the CLI.

6. If your Spark application resides in ADLS, specify the parameter values and run the following curl command. The following example shows the command to submit `read.py` application.

   Example 1 :

```
curl --request POST \
  --url https://<cpd_host_name>/lakehouse/api/v2/spark_engines/<spark_engine_id>/
applications \
  --header 'Authorization: Bearer <token>' \
  --header 'Content-Type: application/json' \
  --header 'LhInstanceId: <instance_id>' \
  --data '{
  "application_details": {
    "application": "abfss://<storage_account>@<storage_container>.dfs.core.windows.net/adls-
read.py",
    "conf": {
        "spark.hadoop.fs.azure.account.auth.type.<storage_account>.dfs.core.windows.net",
"OAuth",

"spark.hadoop.fs.azure.account.oauth.provider.type.<storage_account>.dfs.core.windows.net",
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",

"spark.hadoop.fs.azure.account.oauth2.client.id.<storage_account>.dfs.core.windows.net",
"<application_id>",

"spark.hadoop.fs.azure.account.oauth2.client.secret.<storage_account>.dfs.core.windows.net","
<secret>",

"spark.hadoop.fs.azure.account.oauth2.client.endpoint.<storage_account>.dfs.core.windows.net"
, "https://login.microsoftonline.com/<directory_id>/oauth2/token",
        "spark.hadoop.fs.azure.createRemoteFileSystemDuringInitialization", "false",
        "spark.app.name": "reader-app",
    }
  }
}'
```

   Parameter values:

- *<cpd_host_name>*: The hostname of your Cloud Pak for Data cluster.
- *<spark_engine_id>* : The **Engine ID** of the native Spark engine.
- *<token>* : The bearer token. For more information about generating the token, see Generating a bearer token.
- *<instance_id>* : The instance ID from the watsonx.data cluster instance URL. For example, `1609968977179454`.
- *<storage_account>* : The name of the azure storage account.
- *<storage_container>* : The name of the Azure storage container.
- *<application_id>* : The Application ID of the ServicePrincipal.

- *<secret>*: The Client Secret of the ServicePrincipal. For more information, see Create a service principal.
- *<directory_id>* : The Directory ID of the ServicePrincipal. For more information, see Create a service principal.

Example 2 :

Application code for Gen1 ADLS:

```
from pyspark.sql import SparkSession
import json
import time

spark = SparkSession.builder \
    .appName("ADLS_ICEBERG") \
    .config("spark.hadoop.wxd.apikey", "ZenApiKey
Y3BhZG1pbjowT2VFV0o1WmtVYVhqR0dtOVFxSkJrZzRjeTlvREQzZ0RaU2JDZGFO") \
    .config("fs.wasb.impl", "org.apache.hadoop.fs.azure.IbmlhcasAzureFileSystem") \
    .config("fs.wasbs.impl", "org.apache.hadoop.fs.azure.IbmlhcasAzureFileSystem$Secure") \
    .config("fs.azure.secure.mode", True) \
    .config("fs.azure.local.sas.key.mode", True) \
    .config("fs.azure.saskey.usecontainersaskeyforallaccess", False) \
    .config("spark.sql.catalogImplementation", "hive") \
    .config("spark.sql.catalog.new_catalog_gen1", "org.apache.iceberg.spark.SparkCatalog") \
    .config("hive.metastore.uris", "thrift://ibm-lh-lakehouse-hive-metastore-svc.cpd-
instance.svc.cluster.local:9083") \
    .config("spark.sql.catalog.new_catalog_gen1.type" ,"hive") \
    .config("spark.sql.iceberg.vectorization.enabled" ,"false") \
    .config("spark.hadoop.hive.metastore.schema.verification", "false") \
    .config("spark.hadoop.hive.metastore.schema.verification.record.version", "false") \
    .config("spark.hadoop.datanucleus.schema.autoCreateTables", "false") \
    .config("spark.hive.metastore.use.SSL", "true") \
    .config("spark.hive.metastore.truststore.path", "file:///opt/ibm/jdk/lib/security/
cacerts") \
    .config("spark.driver.extraJavaOptions", "-
Dcom.sun.jndi.ldap.object.disableEndpointIdentification=true
-Djdk.tls.trustNameService=true") \
    .config("spark.executor.extraJavaOptions", "-
Dcom.sun.jndi.ldap.object.disableEndpointIdentification=true
-Djdk.tls.trustNameService=true") \
    .config("spark.hive.metastore.truststore.password", "changeit") \
    .config("spark.hive.metastore.truststore.type", "JKS") \
    .config("hive.metastore.client.auth.mode", "PLAIN") \
    .config("spark.sql.extensions",
"org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions") \
    .config("spark.sql.iceberg.vectorization.enabled" ,"false") \
    .config("spark.hive.metastore.client.plain.username", "cpadmin") \
    .config("spark.hive.metastore.client.plain.password",
"ueTAKG1GX2eo8HafqiAsA4CprTotR9YO") \
    .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
    .config("fs.azure.createRemoteFileSystemDuringInitialization", "true") \
    .enableHiveSupport() \
    .getOrCreate()

# time.sleep(300)
# sc = spark.sparkContext
# sc.setLogLevel("DEBUG")
#Example Spark SQL query
#Execute the SQL query
spark.sql("CREATE database if not exists new_catalog_gen1.new_database_gen1 LOCATION
'wasbs://lhcasblob2@lhcastest2.blob.core.windows.net/new_database_gen1'")

# Execute the SQL query
spark.sql("CREATE TABLE IF NOT EXISTS new_catalog_gen1.new_database_gen1.adls_aa_sam4(col1
INT, col2 STRING)").show()

spark.sql("show tables from new_catalog_gen1.new_database_gen1").show()

spark.sql('describe schema new_catalog_gen1.new_database_gen1').show(25)

spark.sql("insert into new_catalog_gen1.new_database_gen1.adls_aa_sam4 values (1,'Alan'),
(2,'Ben'),(3,'Chen')").show()

spark.sql("select * from new_catalog_gen1.new_database_gen1.adls_aa_sam4").show()

# Stop SparkSession
spark.stop()
```

Example 3 :

Application code for Gen2 ADLS:

```
from pyspark.sql import SparkSession
import json
import time

spark = SparkSession.builder \
    .appName("ADLS_ICEBERG") \
    .config("spark.hadoop.wxd.apikey", "ZenApiKey
Y3BhZG1pbjowT2VFV0o1WmtVYVhqR0dtOVFxSkJrZzRjeTlvREQzZ0RaU2JDZGFO") \
    .config("fs.azure.account.auth.type.annfeng.dfs.core.windows.net", "SAS") \
    .config("fs.azure.sas.token.provider.type.annfeng.dfs.core.windows.net",
"org.apache.hadoop.fs.azurebfs.sas.IbmlhcasSASTokenProvider") \
    .config("spark.sql.catalogImplementation", "hive") \
    .config("spark.sql.catalog.new_catalog", "org.apache.iceberg.spark.SparkCatalog") \
    .config("hive.metastore.uris", "thrift://ibm-lh-lakehouse-hive-metastore-svc.cpd-
instance.svc.cluster.local:9083") \
    .config("spark.sql.catalog.new_catalog.type" ,"hive") \
    .config("spark.sql.iceberg.vectorization.enabled" ,"false") \
    .config("spark.hadoop.hive.metastore.schema.verification", "false") \
    .config("spark.hadoop.hive.metastore.schema.verification.record.version", "false") \
    .config("spark.hadoop.datanucleus.schema.autoCreateTables", "false") \
    .config("spark.hive.metastore.use.SSL", "true") \
    .config("spark.hive.metastore.truststore.path", "file:///opt/ibm/jdk/lib/security/
cacerts") \
    .config("spark.driver.extraJavaOptions", "-
Dcom.sun.jndi.ldap.object.disableEndpointIdentification=true
-Djdk.tls.trustNameService=true") \
    .config("spark.executor.extraJavaOptions", "-
Dcom.sun.jndi.ldap.object.disableEndpointIdentification=true
-Djdk.tls.trustNameService=true") \
    .config("spark.hive.metastore.truststore.password", "changeit") \
    .config("spark.hive.metastore.truststore.type", "JKS") \
    .config("hive.metastore.client.auth.mode", "PLAIN") \
    .config("spark.sql.extensions",
"org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions") \
    .config("spark.sql.iceberg.vectorization.enabled" ,"false") \
    .config("spark.hive.metastore.client.plain.username", "cpadmin") \
    .config("spark.hive.metastore.client.plain.password",
"ueTAKG1GX2eo8HafqiAsA4CprTotR9YO") \
    .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
    .config("fs.azure.createRemoteFileSystemDuringInitialization", "true") \
    .enableHiveSupport() \
    .getOrCreate()

time.sleep(300)
# sc = spark.sparkContext
# sc.setLogLevel("DEBUG")
#Example Spark SQL query
#Execute the SQL query
spark.sql("CREATE database if not exists new_catalog.new_database LOCATION 'abfss://
castest@annfeng.dfs.core.windows.net/aarush_cas_test/new_database'")

# Execute the SQL query
spark.sql("CREATE TABLE IF NOT EXISTS new_catalog.new_database.adls_aa_sam4(col1 INT, col2
STRING)").show()

spark.sql("show tables from new_catalog.new_database").show()

spark.sql('describe schema new_catalog.new_database').show(25)

spark.sql("insert into new_catalog.new_database.adls_aa_sam4 values (1,'Alan'),(2,'Ben'),
(3,'Chen')").show()

spark.sql("select * from new_catalog.new_database.adls_aa_sam4").show()

# Stop SparkSession
spark.stop()
```

7. If your Spark application resides in Google Cloud Storage, specify the parameter values and run the following curl command. The following example shows the command to submit `gcs-read.py` application.

   Example :

```
curl --request POST   --url https://<region>.lakehouse.cloud.ibm.com/lakehouse/api/v2/
spark_engines/<spark_engine_id>/applications   --header 'Authorization: Bearer <token>'   --
```

```
header 'Content-Type: application/json'   --header 'AuthInstanceID: <crn_instance>'   --data
'{
  "application_details": {
    "application": "gs://{bucket_name}//gcs-read.py",
    "conf": {

"spark._jsc.hadoopConfiguration().set("google.cloud.auth.service.account.json.keyfile","<json
_keyfile>")
        "spark.app.name":"GCSFilesRead"
    }
  }
}
```

Parameter values:

- *<region>*: The region where the Spark instance is provisioned..
- *<spark_engine_id>* : The **Engine ID** of the native Spark engine.
- *<token>* : The bearer token. For more information about generating the token, see Generating a bearer token.
- *<crn_instance>* : The instance ID from the watsonx.data cluster instance URL. For example, 1609968977179454.
- *<json_keyfile>* : The path to the json keyfile generated.

8. After you submit the Spark application, you receive a confirmation message with the application ID and Spark version. Save it for reference.
9. Log in to the watsonx.data cluster, access the **Engine details** page. In the **Applications** tab, use the application ID to list the application and you can track the stages. For more information, see View and manage applications.

Submitting Spark application by using DAS to access data

You can submit Spark application by accessing watsonx.data data without object store credentials. If the Spark application (that you want to submit) uses data that resides in an Object Store (COS, AWS S3 and Minio), you can use Data Access Service (DAS) to access the data (without using the object store credentials) and submit the application.

Before you begin

1. Create Cloud Object Storage to store data used in the Spark application. To create Cloud Object Storage and a bucket, see Creating a storage bucket.
2. Register Cloud Object Storage bucket in watsonx.data. For more information, see Adding bucket catalog pair.
3. Upload the Spark application to the storage, see Uploading data.

Procedure

1. To submit the Spark application with data residing in Cloud Object Storage (without accessing object store credentials), specify the parameter values and run the following curl command. The following example shows the command to submit `iceberg.py` application.

   Example 1:

```
curl --request POST \
  --url https://<wxd-host-name>/lakehouse/api/v2/spark_engines/<spark_engine_id>/
applications \
  --header 'Authorization: Bearer <token>' \
  --header 'Content-Type: application/json' \
  --header 'LhInstanceId: <instance-id>' \
  --data '{
  "application_details": {
     "conf": {
        "spark.hadoop.fs.s3a.bucket.<wxd-data-bucket-name>.endpoint": "<wxd-data-bucket-
endpoint>",
        "spark.hadoop.fs.s3a.bucket.<wxd-data-bucket-
name>.aws.credentials.provider":"com.ibm.iae.s3.credentialprovider.WatsonxCredentialsProvider
",
        "spark.hadoop.fs.s3a.bucket.<wxd-data-bucket-
name>.custom.signers":"WatsonxAWSV4Signer:com.ibm.iae.s3.credentialprovider.WatsonxAWSV4Signe
r",
        "spark.hadoop.fs.s3a.bucket.<wxd-data-bucket-name>.s3.signing-
algorithm":"WatsonxAWSV4Signer",
        "spark.sql.catalog.<wxd-bucket-catalog-
name>":"org.apache.iceberg.spark.SparkCatalog",
        "spark.sql.catalog.<wxd-bucket-catalog-name>.type":"hive",
        "spark.sql.catalog.<wxd-bucket-catalog-name>.uri":"thrift://ibm-lh-lakehouse-hive-
metastore-svc.cpd-instance.svc.cluster.local:9083",
        "spark.hadoop.fs.s3a.bucket.<user-application-bucket-name>.endpoint": "<user-
application-bucket-endpoint>",
        "spark.hadoop.fs.s3a.bucket.<user-application-bucket-name>.access.key": "<user-
application-bucket-accesskey>",
        "spark.hadoop.fs.s3a.bucket.<user-application-bucket-name>.secret.key": "<user-
application-bucket-secretkey>",
        "spark.hive.metastore.client.plain.username":"<wxd-user-name>",
        "spark.hive.metastore.client.plain.password":"<wxd-user-password>",
        "spark.hadoop.wxd.apikey":"<user-authentication-string>"
    },
    "application": "s3a://<user-application-bucket-name>/iceberg.py"
  }
}
```

Parameter values:

- *<wxd_host_name>*: The hostname of your watsonx.data or Cloud Pak for Data cluster.
- *<instance_id>* : The instance ID from the watsonx.data cluster instance URL. For example, `1609968977179454`.
- *<spark_engine_id>* : The **Engine ID** of the native Spark engine.
- *<token>* : The bearer token. For more information about generating the token, see Generating a bearer token.
- *<wxd-data-bucket-name>*: The name of the Object storage bucket that contains the data, which the application uses.

  Note: You must register this bucket with watsonx.data, associate a catalog and you must also have access to both the bucket and catalog.
- *<wxd-data-bucket-endpoint>*: The host name of the endpoint for accessing the data bucket mentioned above. Example, `s3.us-south.cloud-object-storage.appdomain.cloud` for a Cloud Object storage bucket in us-south region.
- *<wxd-bucket-catalog-name>*: The name of the catalog associated with the data bucket.
- *<user-application-bucket-name>*: The name of the Object storage bucket that contains the application script.
- *<user-application-bucket-endpoint>*: The hostname of the endpoint for accessing the your bucket containing the application script.
- *<wxd-user-name>*: Your user name credential for the watsonx.data cluster. Note: You must have `Metastore Admin` access on Hive Metastore. For more information, see Managing access to the Hive Metastore (HMS).
- *<wxd-user-password>*: Your password for the watsonx.data cluster.
- *<user-authentication-string>* : The value must be in the format : `echo -n "<username>:<your Zen API key>" | base64`. The `Zen API Key` here is the API key of the user accessing the Object store bucket. To generate API key, log in into the watsonx.data console and navigate to `Profile > Profile and Settings > API Keys` and generate a new API key.

  Note: If you generate a new API key, your old API key becomes invalid.

Python application

The following is the sample Python application, `iceberg.py` that fetch data from watsonx.data bucket and perform some operations.

```
from pyspark.sql import SparkSession
import os
from datetime import datetime
```

```
def init_spark():
    spark = SparkSession.builder.appName("lh-hms-
cloud").enableHiveSupport().getOrCreate()
    return spark

def create_database(spark,bucket_name,catalog):
    spark.sql(f"create database if not exists {catalog}.ivttestdb LOCATION 's3a://
{bucket_name}/'")

def list_databases(spark,catalog):
    # list the database under lakehouse catalog
    spark.sql(f"show databases from {catalog}").show()

def basic_iceberg_table_operations(spark,catalog):
    # demonstration: Create a basic Iceberg table, insert some data and then query table
    print("creating table")
    spark.sql(f"create table if not exists {catalog}.ivttestdb.testTable(id INTEGER,
name VARCHAR(10), age INTEGER, salary DECIMAL(10, 2)) using iceberg").show()
    print("table created")
    spark.sql(f"insert into {catalog}.ivttestdb.testTable values(1,'Alan',23,3400.00),
(2,'Ben',30,5500.00),(3,'Chen',35,6500.00)")
    print("data inserted")
    spark.sql(f"select * from {catalog}.ivttestdb.testTable").show()



def clean_database(spark,catalog):
    # clean-up the demo database
    spark.sql(f'drop table if exists {catalog}.ivttestdb.testTable purge')
    spark.sql(f'drop database if exists {catalog}.ivttestdb cascade')

def main():
    try:
        spark = init_spark()

        create_database(spark,"<wxd-data-bucket-name>","<wxd-data-bucket-catalog-name>")
        list_databases(spark,"<wxd-data-bucket-catalog-name>")
        basic_iceberg_table_operations(spark,"<wxd-data-bucket-catalog-name>")


    finally:
        # clean-up the demo database
        clean_database(spark,"<wxd-data-bucket-catalog-name>")
        spark.stop()

if __name__ == '__main__':
    main()
```

2. After you submit the Spark application, you receive a confirmation message with the application ID and Spark version. Save it for reference.
3. Log in to the watsonx.data cluster, access the **Engine details** page. In the **Applications** tab, use the application ID to list the application and you can track the stages. For more information, see View and manage applications.

Enabling application autoscaling

It is not always possible to estimate the resource requirements (number of Spark executors) of a Spark application upfront, because these requirements might vary based on the size of the input data set. To assist you in this situation, you can submit a Spark application with auto-scaling, which will automatically determine the number of executors required by an application based on the application's demand.

You can enable basic autoscaling for a Spark application by adding the configuration setting `ae.spark.autoscale.enable=true` to the existing application configuration.

watsonx.data on Red Hat OpenShift

Submitting an application with basic autoscaling

The steps to submit an application with autoscaling enabled is the same as the steps to submit an application without autoscaling. The only difference is that you need to add the configuration setting `ae.spark.autoscale.enable=true` to the application payload.

1. Submit a Spark application.
2. Use the following sample JSON payload as an example to enable basic autoscaling:

```
{
  "application_details": {
    "application": "/opt/ibm/spark/examples/src/main/python/wordcount.py",
    "arguments": ["/opt/ibm/spark/examples/src/main/resources/people.txt"]
   },
   "conf": {
     "ae.spark.autoscale.enable":"true"
   }
}
```

Autoscaling application configurations

You can use the following configuration settings to further control autoscaling the number of executors.

Configuration settings	Description	Default
`ae.spark.autoscale.enable`	Signals that the application will autoscale based on the application's demand and any other autoscaling configurations that are set. If specified at instance level, all applications in the instance will autoscale.	false
`spark.dynamicAllocation.initialExecutors`	Specifies the initial number of executors to be created, irrespective of any demand made by the Spark application	0
`spark.dynamicAllocation.minExecutors`	Specifies the minimum number of executors to be maintained, irrespective of any demand by the Spark application	0
`spark.dynamicAllocation.maxExecutors`	Specifies the maximum number of executors to be created irrespective of any demand by the Spark Application	2
`ae.spark.autoscale.scaleupDemandPercentage`	Specifies the percentage of the executors demanded by Spark Application that should be fulfilled by the application auto-scaler. For example, if at a certain stage a Spark application requests 10 executors and the value for this configuration is set to 50 percent, the application auto-scaler will scale up the number of executors by 5 executors. The default value of this configuration is 100 percent which essentially means that any number of executors requested by an application's demand can be added by the application auto-scaler.	100

Configuration settings	Description	Default
`ae.spark.autoscale.scaledownExcessPercentage`	Specifies the percentage of the idle executors held up by the Spark application that are to be removed by the application auto-scaler. For example, if after completing a certain stage in a Spark application, there are 20 idle executors held up by the application and the value for this configuration is set to 25 percent, the application auto-scaler will scale down the executors in the application by 4 executors. The default value for this configuration is 100 percent which means that the application auto-scaler can scale down all idle executors.	100
`ae.spark.autoscale.frequency`	The frequency in seconds at which the application auto-scaler scales up executors or scales down executors in an application.	10s

Example of an auto-scaling application payload showing the lower and upper bounds on the number of executors that the application can scale up or scale down to.

```
{
  "application_details": {
    "application": "cos://<application-bucket-name>.<cos-reference-name>/
my_spark_application.py",
    "conf": {
      "ae.spark.autoscale.enable": "true",
      "spark.dynamicAllocation.initialExecutors": "0",
      "spark.dynamicAllocation.minExecutors": "1",
      "spark.dynamicAllocation.maxExecutors": "10",
      "spark.hadoop.fs.cos.<cos-reference-name>.endpoint": "s3.direct.us-south.cloud-object-
storage.appdomain.cloud",
      "spark.hadoop.fs.cos.<cos-reference-name>.iam.api.key": "<iam-api-key-of-application-
bucket>"
    }
  }
}
```

Enabling autoscaling on an instance

By default, application autoscaling is disabled. If you want to enable automatic scaling for all applications at the instance level, you can explicitly set "ae.spark.autoscale.enable": "true" as a default Spark configuration when you create the instance. See Default Spark configuration.

Submitting Spark jobs to access components in a remote Hadoop cluster

This topic describes how to a run Spark job that access data available in a remote Hadoop cluster from IBM watsonx.data.

You can access the data in Hadoop cluster, which is either secure (kerberized) and insecure (non-rubberized). watsonx.data now support Cloudera Distribution for Hadoop (CDH) based Hadoop clusters.

Before you begin

- Install IBM watsonx.data instance.

- Provision native Spark engine. For more information, see "Provisioning native Spark engine" on page 490.
- The Project Administrator must assign Developer role to the watsonx.data instance.
- To access an insecure remote Hadoop cluster:
  - Ensure that both Hadoop cluster and watsonx.data belongs to same subnet.
  - Set the following Hadoop configuration to allow communication on the Hadoop cluster:

    ```
    hosts=<WXD_PRIVATE_CLOUD_SUBNET>
    ```

    Note: The above configuration ensures that the Spark can access the Hadoop components through the RPC port. Specifically, HDFS Namenode, HDFS Datanode and Hive Metastore (HMS) must be accessible.
- To access an secure remote Hadoop cluster:
  - Ensure that both Hadoop cluster and watsonx.data belongs to same subnet.
  - Set the following Hadoop configuration to allow communication on the Hadoop cluster:

    ```
    hadoop.rpc.protection=privacy
    hadoop.proxyuser.hive.users=<USER_ACCESSING_FROM_SPARK>
    hadoop.proxyuser.hive.hosts=<WXD_PRIVATE_CLOUD_SUBNET>
    hadoop.proxyuser.hive.groups=<REQUIRED_GROUPS_USED_IN_JOBS>
    ```

    Note: The above configuration ensures that the Spark can access the Hadoop components through the RPC port. Specifically, HDFS Namenode, HDFS Datanode and Hive Metastore (HMS) must be accessible.
  - Ensure that the **keytab** file of the user submitting the Spark job on the edge node of the remote Hadoop cluster is readily available. The **keytab** file is required for secure mode of HDFS access.

About this task

The steps to submit Spark jobs that access Hadoop components on a Hadoop cluster vary depending on whether the Hadoop cluster is secure or insecure. The main steps include:

- Running a Spark job on an insecure Hadoop cluster
- Running a Spark job on a secure Hadoop cluster

Running a Spark job on an insecure Hadoop cluster

Specify the parameter values and run the following curl command. The following example shows the command to submit `remoteHadoopAccessSample.py`

```
curl --request POST \
  --url https://<wxd-host-name>/lakehouse/api/v2/spark_engines/<spark_engine_id>/
applications \
  --header 'Authorization: Bearer <token>' \
  --header 'Content-Type: application/json' \
  --header 'LhInstanceId: <instance-id>' \
  --data '{
  "application_details":
        { "application": "cos://<BUCKET_NAME>.<COS_SERVICE_NAME>/
remoteHadoopAccessSample.py",
          "application_arguments": ["hdfs://<namenode-server>:<namenode-rpc-port>/subpath-
to-access>"],
          "conf":
            { "spark.app.name": "RemoteHadoopAccessSample",
              "ae.spark.remoteHadoop.isSecure": "false",
              "ae.spark.remoteHadoop.services": "HDFS,HMS",
              "spark.hadoop.hive.metastore.uris": "thrift://<hms-server>:<hms-port>",
              "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.endpoint": "<COS_ENDPOINT>",
              "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.secret.key": "<COS_SECRET_KEY>",
              "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.access.key": "<COS_ACCESS_KEY>"
            }
        }
}
```

Parameter values:

- *<wxd_host_name>*: The hostname of your watsonx.data or Cloud Pak for Data cluster.
- *<instance_id>* : The instance ID from the watsonx.data cluster instance URL. For example, 1609968977179454.
- *<spark_engine_id>* : The **Engine ID** of the native Spark engine.
- *<token>* : The bearer token. For more information about generating the token, see Generating a bearer token.
- *<bucket-name>*: The name of the Object storage bucket that contains the data, which the application uses.

  Note: You must register this bucket with watsonx.data, associate a catalog and you must also have access to both the bucket and catalog.
- *<COS_SERVICE_NAME>*:
- *<hms-server>:<hms-port>*: The thrift URL of the hive metastore.
- *<cos-endpoint>*: The hostname of the endpoint for accessing the your bucket containing the application script.
- *<COS_ACCESS_KEY>*: Your user name credential for the watsonx.data cluster. Note: You must have `Metastore Admin` access on Hive Metastore.
- *<COS_SECRET_KEY>*: Your password for the watsonx.data cluster.

Running a Spark job on a secure Hadoop cluster

Generate a delegation token on an edge node of the Hadoop cluster. Add this token to the job's payload with the other Hadoop required configurations. Submitting the Spark job. The main steps include:

1. Generate a delegation token. Download the delegation token generation utility based on the version of your Hadoop cluster. For CDH Cluster (Hive standalone 2.1 and later), use Hadoop delegation token generator.
2. Extract the ZIP file by using the following shell command.

```
unzip HadoopDelegationTokenGenerator-0.0.1-SNAPSHOT.zip
Archive: HadoopDelegationTokenGenerator-0.0.1-SNAPSHOT.zip
inflating: HadoopDelegationTokenGenerator-0.0.1-SNAPSHOT.jar
inflating: delegation-token-generator.sh
```

3. Add the JAR file to the classpath. For example, CDH 7.1.7 cluster:

```
export classpath="/opt/cloudera/parcels/CDH-7.1.7-1.cdh7.1.7.p2000.37147774/lib/
hadoop/*:/opt/cloudera/parcels/CDH
-7.1.7-1.cdh7.1.7.p2000.37147774/lib/hive/lib/*:/opt/cloudera/parcels/CDH
-7.1.7-1.cdh7.1.7.p2000.37147774/lib/hadoop-hdfs/*:/opt/cloudera/parcels/CDH
-7.1.7-1.cdh7.1.7.p2000.37147774/lib/hadoop-hdfs/lib/*:/opt/cloudera/parcels/CDH
-7.1.7-1.cdh7.1.7.p2000.37147774/lib/hadoop-mapreduce/*:/etc/hadoop/conf:/etc/hive/
conf:/opt/cloudera/parcels/CDH
-7.1.7-1.cdh7.1.7.p2000.37147774/lib/spark/*:/opt/cloudera/parcels/CDH
-7.1.7-1.cdh7.1.7.p2000.37147774/lib/search/lib/*:/root/pratham/
HadoopDelegationTokenGenerator
-0/HadoopDelegationTokenGenerator-0.0.1-SNAPSHOT.jar"
```

4. If you are using both Hive and HMS token, export the configurations for Hive. Example:

```
export hive_kerberos_principal="hive/abc.xyz.com@EXAMPLE.COM";
export hive_metastore_uri="thrift://<thrift-server>:9083";
```

5. Export your HADOOP_HOME. Example:

```
export HADOOP_HOME="/opt/cloudera/parcels/CDH-7.1.7-1.cdh7.1.7.p2000.37147774/lib/
hadoop/"
```

6. Generate the Kerberos Ticket Granting Ticket (TGT). Let's use the user `hdfs` on a CDH cluster as an example:

```
kinit -kt /<path to hdfs keytab>/hdfs.keytab hdfs/abc.xyz.com@EXAMPLE.COM
```

7. Execute the shell script from the downloaded ZIP file:

   ```
   sh delegation-token-generator.sh hdfs/abc.xyz.com@EXAMPLE.COM /mytok.dt HDFS HMS
   ```

   The token is fetched. The last two parameters in the command are the components for which the token needs to be generated. If you only need HDFS, omit HMS or vice versa. The script prints the delegation token in base64 encoded format as a string. Note down this string for later use.
8. Prepare the job payload by adding the following Hadoop specific configurations. If the remote Hadoop cluster is kerberized, set the following parameter:

   ```
   "ae.spark.remoteHadoop.isSecure" : "true"

   If your Spark job accesses services, list those services:

   "ae.spark.remoteHadoop.services": "HDFS,HMS"

   The above example shows accessing HDFS and HMS.
   ```
9. Export all the hadoop configurations files on the remote Cluster in this case CDH and mount all those conf files to your spark application. Example of configuration files needed. See Creating a Storage Volumes

   ```
   core-site.xml
   hadoop-env.sh
   hdfs-site.xml
   log4j.properties
   mapred-site.xml
   ssl-client.xml
   topology.map
   ```
10. Add the mount location to extraClasspath of both driver and executor and set the HADOOP_CONF_DIR to this path. Example:

    ```
    "spark.driver.extraClassPath":"/mnts/remote-hadoop-test/"
    "spark.executor.extraClassPath":"/mnts/remote-hadoop-test/"

    "HADOOP_CONF_DIR": "/mnts/remote-hadoop-test/"
    ```
11. To enable access to the kerberized Hadoop cluster from Spark, add the delegation token you noted down:

    ```
    "ae.spark.remoteHadoop.delegationToken": "<token>"
    ```
12. If you are accessing HMS from Spark, add the Hive Metastore Kerberos principal and the URI to access the Hive Metastore:

    ```
    "spark.hadoop.hive.metastore.kerberos.principal" : "hive/abc.xyz.com@EXAMPLE.COM"
    "spark.hadoop.hive.metastore.uris":"thrift://<thrift-server>:<thrift-port>"
    ```
13. Here is a sample payload for an application called `remoteHadoopAccessSample.py`:

    ```
    {
      "application_details":
      { "application": "cos://<BUCKET_NAME>.<COS_SERVICE_NAME>/
    remoteHadoopAccessSample.py",
        "application_arguments": ["hdfs://<namenode-server>:<namenode-rpc-port>/<subpath-to-
    access>"],
        "conf":
        { "spark.app.name": "RemoteHadoopAccessSample",
          "ae.spark.remoteHadoop.isSecure": "true",
          "ae.spark.remoteHadoop.services": "HDFS,HMS",
          "ae.spark.remoteHadoop.delegationToken": "<base64-encoded-delegation-token>",
          "spark.hadoop.hive.metastore.kerberos.principal": "<hms-kerberos-principal>",
          "spark.hadoop.hive.metastore.uris": "thrift://<hms-server>:<thrift-port>",
          "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.endpoint": "<COS_ENDPOINT>",
          "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.secret.key": "<COS_SECRET_KEY>",
          "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.access.key": "<COS_ACCESS_KEY>",
          "spark.hadoop.hadoop.security.authentication": "kerberos",
          "spark.driver.extraClassPath":"/mnts/<vol-mount-path>/",
          "spark.executor.extraClassPath":"/mnts/<vol-mount-path>/"
        },
    ```

```
    "env": {
        "HADOOP_CONF_DIR": "/mnts/<vol-mount-path>/ "
        }
        },
  "volumes": [{
    "name": "<volume-name>",
    "mount_path": "/mnts/<vol-mount-path>"
  }]
}
```

14. Here is an example of a Spark application (remoteHadoopAccessSample.py in the previous sample payload) that shows you how to access HDFS and HMS:

```
import sys
from pyspark.sql import SparkSession
if __name__ == "__main__":
spark = SparkSession.builder.appName("secureHadoop").enableHiveSupport().getOrCreate()
path = "{}/{} ".format(sys.argv[1], sys.argv[2])
print("Path accessed in HDFS is : {}".format(path));
df = spark.read.format("csv").option("timestampFormat", "yyyy/MM/dd HH:mm:ss
ZZ").load(path);
df.show()
sqlDF1 = spark.sql("show tables")
sqlDF1.show()
tablename = "securehadoop"
createsql = "create external table {} (name string,id string) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '{}/{}'".format(tablename,
sys.argv[1], sys.argv[2])
print("SQL executed for HMS : {}".format(createsql));
sqlDFCr = spark.sql(createsql)

insertsql = "insert into {} values('newvalue','123456')".format(tablename)
print("SQL executed for insert in HMS :{}".format(insertsql));
sqlDF2 = spark.sql(insertsql)
sqlDF = spark.sql("SELECT * FROM {}".format(tablename))
sqlDF.show()
spark.stop()
```

15. Submit your application as a Spark job.

Debug the Spark application

The Engine home (storage volume that you create at the time of provisioning the Spark engine) enables you to debug the Spark application and monitor the logs that the Spark application generates. The administrator can access the engine home and debug the Spark application.

Procedure

1. Log in to Cloud Pack for Data. Click **Administration** > **Storage volumes**. The **Engine home** (storage volume that you create at the time of provisioning the Spark engine) is already listed.
2. Click the **File browser** tab. Search for the engine by using the Spark ID (generated at the time of provisioning Spark engine). The log folder with the Spark ID is listed.
3. Open the folder to see the logs associated with the application that you submitted. You can search by using the application ID that you received at the time of application submission.
4. To view the results of the application, open the Spark driver log file. You can also download the file to view the details of execution.

Working with Apache Hudi catalog

The topic describes the procedure to run a Spark application that ingests data into an Apache Hudi catalog.

Before you begin

To enable your Spark application to work with the watsonx.data catalog and storage, you must have `Metastore admin` role. Without `Metastore admin` privilege, you cannot ingest data to storage using

Native Spark engine. To enable your Spark application to work with the watsonx.data catalog and storage, add the following configuration to your application payload:.

```
spark.hive.metastore.client.plain.username=ibmlhapikey
spark.hive.metastore.client.plain.password=<api-key-of-the-user-which-has-metastore-admin-role>
spark.hadoop.wxd.apiKey=Basic base64(ibmlhapikey_ibmcloudid:apikey)
```

Procedure

1. Create a storage with Apache Hudi catalog to store data used in the Spark application. To create storage with Apache Hudi catalog, see Adding a storage-catalog pair.
2. Associate the storage with the Native Spark engine. For more information, see Associating a catalog with an engine.
3. Create Cloud Object Storage (COS) to store the Spark application. To create Cloud Object Storage and a bucket, see Creating a storage bucket.
4. Register the Cloud Object Storage in watsonx.data. For more information, see Adding a storage-catalog pair.
5. Save the following Spark application (Python file) to your local machine. Here, `hudi_demo.py`.

   The Python Spark application demonstrates the following functionality:

   - It creates a database inside the Apache Hudi catalog (that you created to store data). Here, `hudi_db`.
   - It creates a table inside the `hudi_db` database, namely `hudi_table`.
   - It inserts data into the `hudi_table` and does SELECT query operation.
   - It drops the table and schema after use.

   ```
   from pyspark.sql import SparkSession

   def init_spark():
       spark = SparkSession.builder \
           .appName("CreateHudiTableInCOS") \
           .enableHiveSupport() \
           .getOrCreate()
       return spark

   def main():

       try:
           spark = init_spark()
           spark.sql("show databases").show()
           spark.sql("create database if not exists spark_catalog.hudi_db LOCATION 's3a://hudi-
   connector-test/'").show()
           spark.sql("create table if not exists spark_catalog.hudi_db.hudi_table (id
   bigint, name string, location string) USING HUDI OPTIONS ('primaryKey' 'id',
   hoodie.write.markers.type= 'direct', hoodie.embed.timeline.server= 'false')").show()
           spark.sql("insert into hudi_db.hudi_table VALUES (1, 'Sam','Kochi'), (2,
   'Tom','Bangalore'), (3, 'Bob','Chennai'), (4, 'Alex','Bangalore')").show()
           spark.sql("select * from spark_catalog.hudi_db.hudi_table").show()
           spark.sql("drop table spark_catalog.hudi_db.hudi_table").show()
           spark.sql("drop schema spark_catalog.hudi_db CASCADE").show()

       finally:
           spark.stop()

   if __name__ == '__main__':
       main()
   ```

6. Upload the Spark application to the COS, see Uploading data.
7. To submit the Spark application with data residing in Cloud Object Storage, specify the parameter values and run the following curl command.

   ```
   curl --request POST \
     --url https://<wxd_host_name>/lakehouse/api/v2/spark_engines/<spark_engine_id>/
   applications \
     --header 'Authorization: Bearer <token>' \
     --header 'Content-Type: application/json' \
     --header 'LhInstanceId: <instance_id>' \
   ```

```
   --data '{
     "application_details": {
         "conf": {
         "spark.serializer" : "org.apache.spark.serializer.KryoSerializer",
         "spark.hadoop.fs.s3a.path.style.access" : "true",
         "spark.hive.metastore.client.plain.username":"<wxd-user-name>",
         "spark.hive.metastore.client.plain.password":"<wxd-user-password>",
         "spark.driver.extraJavaOptions" :
"-Dcom.sun.jndi.ldap.object.disableEndpointIdentification=true
-Djdk.tls.trustNameService=true",
         "spark.executor.extraJavaOptions" :
"-Dcom.sun.jndi.ldap.object.disableEndpointIdentification=true
-Djdk.tls.trustNameService=true",

         "spark.kryo.registrator": "org.apache.spark.HoodieSparkKryoRegistrar",
         "spark.sql.catalog.spark_catalog.type": "hudi",
         "spark.sql.catalog.spark_catalog": "org.apache.spark.sql.hudi.catalog.HoodieCatalog",

         "spark.hadoop.wxd.cas.endpoint":"https://cas-
ussouth.lakehouse.dev.appdomain.cloud/cas/v1/signature",
         "spark.hadoop.wxd.cas.apiKey":"<user-authentication-string> "

         },
         "application": "s3a://hudi-connector-test/hudi_demo.py"
     }
}
```

Parameter values:

- *<wxd_host_name>*: The hostname of your watsonx.data or Cloud Pak for Data cluster.
- *<instance_id>* : The instance ID from the watsonx.data cluster instance URL. For example, 1609968977179454.
- *<spark_engine_id>* : The **Engine ID** of the native Spark engine.
- *<token>* : The bearer token. For more information about generating the token, see Generating a bearer token.
- *<wxd-user-name>*: Your user name credential for the watsonx.data cluster. Note: You must have `Metastore Admin` access on Hive Metastore. For more information, see Managing access to the Hive Metastore (HMS).
- *<wxd-user-password>*: Your password for the watsonx.data cluster.
- *<user-authentication-string>* : The value must be in the format : `echo -n "<username>:<your Zen API key>" | base64`. The `Zen API Key` here is the API key of the user accessing the Object store bucket. To generate API key, log in into the watsonx.data console and navigate to `Profile > Profile and Settings > API Keys` and generate a new API key.

  Note: If you generate a new API key, your old API key becomes invalid.

8. After you submit the Spark application, you receive a confirmation message with the application ID and Spark version. Save it for reference.
9. Log in to the watsonx.data cluster, access the **Engine details** page. In the **Applications** tab, use application ID to list the application and track the stages. For more information, see View and manage applications.

Working with Apache Delta Lake catalog

The topic describes the procedure to run a Spark application that ingests data into a Delta Lake catalog.

Before you begin

To enable your Spark application to work with the watsonx.data catalog and storage, you must have `Metastore admin` role. Without `Metastore admin` privilege, you cannot ingest data to storage using Native Spark engine. To enable your Spark application to work with the watsonx.data catalog and storage, add the following configuration to your application payload:.

```
spark.hive.metastore.client.plain.username=ibmlhapikey
spark.hive.metastore.client.plain.password=<api-key-of-the-user-which-has-metastore-admin-role>
spark.hadoop.wxd.apiKey=Basic base64(ibmlhapikey_ibmcloudid:apikey)
```

Procedure

1. Create a storage with Delta Lake catalog to ingest and manage data in delta table format. To create storage with Delta Lake catalog, see Adding a storage-catalog pair.
2. Associate the storage with the Native Spark engine. For more information, see Associating a catalog with an engine.
3. Create Cloud Object Storage (COS) to store the Spark application. To create Cloud Object Storage and a bucket, see Creating a storage bucket.
4. Register the Cloud Object Storage in watsonx.data. For more information, see Adding a storage-catalog pair.
5. Save the following Spark application (Python file) to your local machine. Here, `delta_demo.py`.

   The Python Spark application demonstrates the following functionality:

   - It creates a database inside the Delta Lake catalog (that you created to store data). Here, `iae`.
   - It creates a table inside the `iae` database, namely `employee`.
   - It inserts data into the `employee` and does SELECT query operation.
   - It drops the table and schema after use.

```
from pyspark.sql import SparkSession
import os

def init_spark():
    spark = SparkSession.builder.appName("lh-hms-cloud")\
    .enableHiveSupport().getOrCreate()

    return spark

def main():
    spark = init_spark()
    spark.sql("show databases").show()
    spark.sql("create database if not exists spark_catalog.iae LOCATION 's3a://delta-
connector-test/'").show()
    spark.sql("create table if not exists spark_catalog.iae.employee (id bigint, name
string, location string) USING DELTA").show()
    spark.sql("insert into spark_catalog.iae.employee VALUES (1, 'Sam','Kochi'), (2,
'Tom','Bangalore'), (3, 'Bob','Chennai'), (4, 'Alex','Bangalore')").show()
    spark.sql("select * from spark_catalog.iae.employee").show()
    spark.sql("drop table spark_catalog.iae.employee").show()
    spark.sql("drop schema spark_catalog.iae CASCADE").show()
    spark.stop()

if __name__ == '__main__':
    main()
```

6. Upload the Spark application to the COS, see Uploading data.
7. To submit the Spark application with data residing in Cloud Object Storage, specify the parameter values and run the following curl command.

```
curl --request POST \
  --url https://<wxd_host_name>/lakehouse/api/v2/spark_engines/<spark_engine_id>/
applications \
  --header 'Authorization: Bearer <token>' \
  --header 'Content-Type: application/json' \
  --header 'LhInstanceId: <instance_id>' \
  --data '{
    "application_details": {
    "conf": {
            "spark.sql.catalog.spark_catalog" :
"org.apache.spark.sql.delta.catalog.DeltaCatalog",
            "spark.sql.catalog.spark_catalog.type" : "hive",
            "spark.hive.metastore.client.plain.username" : "ibmlhapikey",
            "spark.hive.metastore.client.plain.password" :
"IYWWVLfNv5v0mY_LOxhkWjLaPI1YEDZ2S5dQLdITdY3k",
            "spark.hadoop.wxd.cas.endpoint":"https://cas-
ussouth.lakehouse.dev.appdomain.cloud/cas/v1/signature",
            "spark.hadoop.wxd.cas.apiKey":"Basic
aWJtbGhhcGlrZXlfZGhhcmphaW5AaW4uaWJtLmNvbTpJWVdXVkxmTnY1djBtWV9MT3hoa1dqTGFQSTFZRURaMlM1ZFFMZ
ElUZFkzaw=="
    },
    "application": "s3a://delta-connector-test/delta_demo.py"
    }
}
```

Parameter values:

- *<wxd_host_name>*: The hostname of your watsonx.data or Cloud Pak for Data cluster.
- *<instance_id>* : The instance ID from the watsonx.data cluster instance URL. For example, 1609968977179454.
- *<spark_engine_id>* : The **Engine ID** of the native Spark engine.
- *<token>* : The bearer token. For more information about generating the token, see Generating a bearer token.
- *<wxd-user-name>*: Your user name credential for the watsonx.data cluster. Note: You must have `Metastore Admin` access on Hive Metastore. For more information, see Managing access to the Hive Metastore (HMS).
- *<wxd-user-password>*: Your password for the watsonx.data cluster.
- *<user-authentication-string>* : The value must be in the format : `echo -n "<username>:<your Zen API key>" | base64`. The `Zen API Key` here is the API key of the user accessing the Object store bucket. To generate API key, log in into the watsonx.data console and navigate to `Profile > Profile and Settings > API Keys` and generate a new API key.

  Note: If you generate a new API key, your old API key becomes invalid.

8. After you submit the Spark application, you receive a confirmation message with the application ID and Spark version. Save it for reference.
9. Log in to the watsonx.data cluster, access the **Engine details** page. In the **Applications** tab, use application ID to list the application and track the stages. For more information, see View and manage applications.

VS Code development environment - Spark labs

The VS Code development environment is a Spark-based development environment that enables you to interactively program, debug, submit, and test Spark applications on a Spark cluster running on the Spark engine. It is available as a Visual Studio Code extension and you can install it in your local system to access Spark IDE using Visual Studio Code. It reduces the time for development and increases usability.

Before you begin

1. Install a desktop version of Visual Studio Code.
2. Install watsonx.data extension from VS Code Marketplace.
3. Ensure that you have public-private SSH key pair to establish an SSH connection with the Spark lab. For more information about generating the key, open watsonx.data extension in Visual Studio Code, go to **Details** tab, see **Set up SSH on your machine** section.
4. Install the extension `Remote -` SSH from Visual Studio Code marketplace.

Important: As Spark labs are ephemeral in nature, you must back up the data stored periodically to prevent potential data loss during upgrades or a Spark master crash.

About this task

Setting up the Spark labs

1. Open Visual Studio Code. You view the watsonx.data icon in the left navigation window. Click the icon. The **Settings** window opens.
2. Configure the watsonx.data extension.

   Click **Go to Settings**. Configure the following items:

   a. • Environment Type: Select Cloud Pak for Data as the environment type.

    - watsonx-data.host: Hostname of the watsonx.data console. Get it from the address bar of your browser when you log in to Cloud Pak for Data or IBM Cloud. For example, `` `cpd.xxx.xxx.xxx.com` ``.
    - watsonx-data.username: The username of your Cloud Pak for Data cluster.
    - watsonx-data.privateKeyPath: Path to your private SSH key file.
    - watsonx-data.publicKeyPath: Path to your public SSH key file.

   b. Click **Refresh**. The window prompts for Cloud Pak for Data API key. To get the API key, log in into the watsonx.data console and navigate to `Profile > Profile and Settings > API Keys`.

   c. Provide the API key and press **Enter**. Refresh your window to view the Spark engine in the left pane of Visual Studio Code application.

3. Create a Spark lab.

   a. To create a new Spark lab, click the + icon. The **Create Spark Lab** window opens. Specify your **public SSH key** and the **public SSH keys** of the users whom you want to grant access to Spark lab. Specify each public SSH key on a new line.

   b. Click **Create**. Click **Refresh** to see the Spark lab in the left window. This is the dedicated Spark cluster for application development.

   c. Open the Spark lab to access the file system, terminal, and work with it.

   d. In the **Explorer** window, you can view the file system, where you can upload the files, and view logs.

   Note: To delete an already running Spark lab, hover the mouse over the name of the Spark lab in the watsonx.data left navigation pane and click on Delete icon.

Developing a Spark application

Develop a Spark application in the Spark lab. You can work with a Spark application in one of the following ways:

- Create your own Python file
- Create Jupyter Notebooks

Create your own Python file

1. Create, upload or drag the Python application file to the **Explorer** window. The file opens in the right pane of Visual Studio Code application.
2. Run the following command in the terminal. This initiates a Python session and you can see the acknowledgment message in the terminal.

```
python <filename>
```

Create Jupyter Notebooks

1. Browse for the `Jupyter` extension from the VS Code Marketplace and install the extension.
2. You can either create a new Jupyter Notebook file with the extension `.ipynb` or drag and drop the existing notebook to the **Explorer** window.
3. From the **Explorer** window, double-click to open the Jupyter Notebook.
4. From the Jupyter Notebook, click the **Change Kernel** link to select the **Python Environment**.
5. The Jupyter Notebook is now ready to use. You can write your code and execute it cell by cell.

Monitoring and debugging Spark applications from Spark labs

Before you begin

1. Install a desktop version of Visual Studio Code.
2. Install watsonx.data extension from VS Code Marketplace. (for version 1.1.4)

3. Ensure that you have public-private SSH key pair to establish an SSH connection with the Spark lab. For more information about generating the key, open watsonx.data extension in Visual Studio Code, go to **Details** tab, see **Set up SSH on your machine** section.
4. Install the extension `Remote - SSH` from Visual Studio Code marketplace.

About this task

Debugging the Spark application from Spark labs

The Spark lab also allows you to debug the Spark application that you submit. To do that:

1. Go to Visual Studio Code > **Extensions**.
2. Browse for the debugging tool to debug the code. For each Spark application type (Python, Java, Scalar, R), you need to choose the official extension to debug. For example, if you submit a Spark application that is written in Python language, install **Python** extension.
3. After you install the debugging tool extension, open the file that you want to debug and click **Run and Debug** from the Visual Studio Code.
4. The Visual Studio Code window prompts for the language of the Spark application code and default configuration.
5. Select the language and provide the default configuration based on your Spark application type (Python, Java, Scalar, R).
6. Click **Run and Debug**. The debugging process starts and you can view the result in the **Terminal**.

Accessing Spark UI from Spark labs

The Spark user interface (UI) allows you to monitor various aspects of running a Spark application. For more information, see Spark user interface. Expose Spark UI to access it from Spark labs. To do that:

1. Go to Visual Studio Code > **Terminal** and select the **Ports** tab.
2. Click **Forward a Port**. Type `4040` and press **Enter**. You can now access Spark UI from Spark labs. Open a web browser and enter the URL in the format - `localhost:4040`. The Spark UI opens, which allows you to inspect Spark applications in the Spark labs. You can view the following details :
   - The **Event timeline** displays a graphical view of the timeline and the events.
   - Different stages of execution, the storage used, the Spark environment, and executor (memory and driver) details.

Monitoring Spark application runs by using Databand

Databand integration with Spark enhances monitoring capabilities by providing insights that extend beyond Spark UI and Spark History.

watsonx.data on Red Hat OpenShift

Databand improves Spark application monitoring by the following:

Advanced monitoring

Databand’s task annotations enable you to tag and track crucial stages of your Spark application, offering a more meaningful level of monitoring compared to Spark jobs, stages, or tasks.

Dataset tracking

Databand monitors the datasets that are accessed and modified during your Spark application run, providing enhanced visibility into your data flows.

Custom alerts

You can configure alerts for specific stages of your application or track key dataset metrics, allowing you to identify and address potential issues early.

Before you begin

To get started with Databand, you must have an active Databand subscription. You can obtain this by either requesting a Databand cloud application (SaaS) instance, which is deployed by the Databand team, or by opting for a self-hosted (on-premises) installation. For integrating databand with your watsonx.data instance, you must have the following credentials:

- **Environment address**: The URL for your Databand environment (example: `yourcompanyname.databand.ai`).
- **Access token**: A Databand access token that is needed to connect to the environment. You can generate and manage tokens through the Databand UI as required. For more details, visit: Managing personal access tokens.

What to do next

- Track Spark applications
- Invoke Spark Applications

Track Spark applications

You can track your Spark applications by using Databand through the methods that are mentioned in this section.

watsonx.data on Red Hat OpenShift

Databand listener

This method automatically tracks dataset operations. With an agent, you can activate Databand listeners without incorporating them into your project or making further code modifications. The agent's `FatJar` (`-all.jar` file) includes all necessary Databand code. You can specify this jar dependency when submitting your Spark applications. Download the latest version of the jar from Maven Repository.

Databand decorators and logging API

This approach facilitates manual tracking, allowing Databand to deliver insights into code errors, metrics, and logging details. To utilize this method, you need to import the dbnd module, which involves modifying your code. You can persist packages to use in Spark applications in a service volume instance.

To configure and use a volume in watsonx.data, do the following steps:

Procedure

1. Create a new volume named `appvol` in your Cloud Pak for Data instance. Set the `storageClass` to `managed-nfs-storage` and specify an appropriate mount path. For more information about creating volumes, see Create a Cloud Pak for Data storage volume.
2. Within the `appvol` volume, create a subfolder named `pippackages` and upload the necessary Python package (dbnd) from your local machine.
   a. Install Python and a virtual environment tool (`venv` or `virtualenv`). If you are using `virtualenv`, install it using `pip install virtualenv`.
   b. Run `python -m venv myenv` or `virtualenv myenv` to create a virtual environment.
   c. Activate the virtual environment by using the command: `myenv\Scripts\activate` (Windows) or `source myenv/bin/activate` (mac OS/Linux).
   d. After activating the virtual environment, install the dbnd package by using the command: `pip install dbnd`.
   e. You can verify the installation of dbnd package by using the command : `pip show dbnd`.

f. Create a compressed archive of all the contents in the site-packages directory using the following command to prepare the dependencies required for the dbnd library upload.

```
tar -czvf dbnd.tar.gz -C /path/to/your/myenv/lib/python{version}/
      site-packages
```

g. Upload the compressed archive to the `pippackages` folder inside `appvol`. Ensure that all options are selected to extract all files and make them executable during the upload process.

h. After you finish working with the virtual environment, deactivate it by using the command : `deactivate`.

3. Upload your `PySpark` application to the `customApps` subfolder in `appvol`. Ensure that all checkboxes are selected to extract all files and make them executable upon upload.
4. Submit your `PySpark` application that utilizes the packages that are uploaded to the `appvol` volume in the `pippackages` folder as a Spark job by using the following curl command:

```
curl -ivk -X POST -d @payload.json -H "Content-Type: application/json" -H "LhInstanceId:
<InstanceId>" -H "Authorization: ZenApiKey <Token>" "<watsonx_application_endpoint>"
```

Note: For information about generating token, see Authentication.

Sample of `payload.json`

```
{
  "application_details": {
    "application": "/<mount_path>/customApps/<application_name>.py",
    "conf": {
      "spark.app.name": "MyJob",
      "spark.eventLog.enabled": "true"
    },
    "env": {
      "PYTHONPATH": "/<mount_path>/pippackages/python:/home/spark/space/assets/
data_asset:/home/spark/user_home/python-3:/cc-home/_global_/python-3:/home/spark/shared/
user-libs/python:/home/spark/shared/conda/envs/python/lib/python/site-packages:/opt/ibm/
conda/miniconda/lib/python/site-packages:/opt/ibm/third-party/libs/python3:/opt/ibm/
image-libs/python3:/opt/ibm/image-libs/spark2/metaindexmanager.jar:/opt/ibm/image-libs/
spark2/stmetaindexplugin.jar:/opt/ibm/spark/python:/opt/ibm/spark/python/lib/py4j-0.10.7-
src.zip",
      "DBND__CORE__DATABAND_URL": "<Databand Server url>",
      "DBND__CORE__DATABAND_ACCESS_TOKEN": "Personal access token for your account",
      "DBND__TRACKING": "True",
      "DBND__ENABLE__SPARK_CONTEXT_ENV": "True",
      "DBND__TRACKING__PROJECT": "<project_name_for_your_Databand_pipeline>",
      "DBND__TRACKING__JOB": "<pipeline_name_for_your_Databand_run>",
      "DBND__RUN_INFO__NAME": "<run_name>"
    }
  },
  "volumes": [
    {
      "name": "cpd-instance::appvol",
      "mount_path": "/<mount_path>",
      "source_sub_path": ""
    }
  ]
}
```

Alternatively, you can make the Python packages visible to your `PySpark` application by adding the following lines to the top of your `PySpark` script:

```
import sys sys.path.append('/<mount_path>/pippackages/')
```

Invoke Spark applications

This section provides examples of submitting your Spark application while integrating Databand.

watsonx.data on Red Hat OpenShift

End-to-End Example of using Databand listener

Your PySpark script can benefit from automatic tracking of data set operations without requiring any code changes.

1. Create a JSON file for submitting your Spark job as follows. Example: `submit-application.son`.

```
{
    "application_details": {
      "application": "/<mount_path>/customApps/<application_name>.py",
      "jars": "/<mount_path>/dbnd-agent-<version>-all.jar",
      "conf": {
        "spark.app.name": "<spark-app-name>",
        "spark.eventLog.enabled": "true",
        "spark.sql.queryExecutionListeners":
"ai.databand.spark.DbndSparkQueryExecutionListener"
      },
      "env": {
        "DBND__CORE__DATABAND_URL": "<Databand Server url>",
        "DBND__CORE__DATABAND_ACCESS_TOKEN": "Personal access token for your account",
        "DBND__TRACKING": "True",
        "DBND__ENABLE__SPARK_CONTEXT_ENV": "True",
        "DBND__TRACKING__PROJECT": "<project_name_for_your_Databand_pipeline>",
        "DBND__TRACKING__JOB": "<pipeline_name_for_your_Databand_run>",
        "DBND__RUN_INFO__NAME": "<run_name>"
      }
    },
   "volumes": [{
        "name": "cpd-instance::appvol",
        "mount_path": "/<mount_path>",
        "source_sub_path": ""
    }]
  }
```

Properties for enabling Databand features are passed as Spark environment variables:

spark.sql.queryExecutionListeners
Used to register a custom listener that is provided by DataBand for Spark SQL query executions. This listener integrates with Spark to collect and report metrics or information about query executions.

DBND__CORE__DATABAND_URL
The URL for accessing your Databand instance (example: `yourcompanyname.databand.ai`).

DBND__CORE__DATABAND_ACCESS_TOKEN
Your Databand Access Token that is used for connecting to the environment.

DBND__TRACKING
Controls whether tracking is enabled or disabled for the project.

DBND__ENABLE__SPARK_CONTEXT_ENV
Activates implicit tracking within the Spark context.

DBND__TRACKING__PROJECT
Specifies the project name for tracking.

DBND__TRACKING__JOB
Specifies the job name for tracking.

DBND__RUN_INFO__NAME
Specifies run names that are displayed in the Databand UI.

dbnd-agent jar
Include the file inside the `appvol`. If you are using an agent, Databand listeners are activated automatically as the agent's FatJar (-all.jar file) includes all necessary Databand code. Download the latest version from Maven Repository.

2. You can add more configuration properties as required as listed in Databand Configuration.
3. Submit the Spark application by using the following curl command:

```
curl -ivk -X POST \
  -d @payload.json \
  -H "Content-Type: application/json" \
  -H "LhInstanceId: <InstanceId>" \
  -H "Authorization: ZenApiKey <Token>" \
  "<watsonx_application_endpoint>"
```

For more information, see Submit engine applications.

After you submit the Spark application, you will receive a confirmation message with the application ID and Spark version. Keep this information for tracking the execution status of your submitted Spark job. You can monitor and track datasets by using Databand's tracking features within the Databand environment.

End-to-End example of using Databand APIs

The following example demonstrates the use of dbnd APIs.

`example.py`

```
import time
import logging
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from dbnd import dbnd_tracking, task, dataset_op_logger, log_metric, log_dataframe

# Initialize Spark session
spark = SparkSession.builder \
    .appName("Data Pipeline with Databand") \
    .getOrCreate()

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

@task
def create_sample_data():
    # Create a DataFrame with sample data including columns to be dropped
    data = [
        ("John", "Camping Equipment", 500, "Regular", "USA"),
        ("Jane", "Golf Equipment", 300, "Premium", "UK"),
        ("Mike", "Camping Equipment", 450, "Regular", "USA"),
        ("Emily", "Golf Equipment", 350, "Premium", "Canada"),
        ("Anna", "Camping Equipment", 600, "Regular", "USA"),
        ("Tom", "Golf Equipment", 200, "Regular", "UK")
    ]
    columns = ["Name", "Product line", "Sales", "Customer Type", "Country"]
    retailData = spark.createDataFrame(data, columns)

    # Log the data creation
    unique_file_name = "sample-data"
    with dataset_op_logger(unique_file_name, "read", with_schema=True, with_preview=True,
with_stats=True) as logger:
        logger.set(data=retailData)

    return retailData

@task
def filter_data(rawData):
    # Define columns to drop
    columns_to_drop = ['Customer Type', 'Country']

    # Drop the specified columns in PySpark DataFrame
    filteredRetailData = rawData.drop(*columns_to_drop)

    # Log the data after dropping columns
    unique_file_name = 'script://Weekly_Sales/Filtered_df'
    with dataset_op_logger(unique_file_name, "read", with_schema=True, with_preview=True) as
logger:
        logger.set(data=filteredRetailData)

    return filteredRetailData

@task
def write_data_by_product_line(filteredData):
    # Filter data for Camping Equipment and write to CSV
    campingEquipment = filteredData.filter(col('Product line') == 'Camping Equipment')
    campingEquipment.write.csv("Camping_Equipment.csv", header=True, mode="overwrite")

    # Log writing the Camping Equipment CSV
    log_dataframe("camping_equipment", campingEquipment, with_schema=True, with_stats=True)
    # Filter data for Golf Equipment and write to CSV
    golfEquipment = filteredData.filter(col('Product line') == 'Golf Equipment')
    golfEquipment.write.csv("Golf_Equipment.csv", header=True, mode="overwrite")

    # Log writing the Golf Equipment CSV
    log_dataframe("golf_equipment", golfEquipment, with_schema=True, with_stats=True)

def prepare_retail_data():
    with dbnd_tracking(
            conf={
                "tracking": {
                    "track_source_code": True
                },
                "log": {
                    "preview_head_bytes": 15360,
                    "preview_tail_bytes": 15360
                }
            }
    ):

        logger.info("Running Databand spark application!")

        start_time_milliseconds = int(round(time.time() * 1000))
        log_metric("metric_check", "OK")

        # Call the step job - create sample data
        rawData = create_sample_data()

        # Filter data
        filteredData = filter_data(rawData)

        # Write data by product line
        write_data_by_product_line(filteredData)

        end_time_milliseconds = int(round(time.time() * 1000))
        elapsed_time = end_time_milliseconds - start_time_milliseconds

        log_metric('elapsed-time', elapsed_time)

        logger.info(f"Total pipeline running time: {elapsed_time:.2f} milliseconds")
        logger.info("Spark execution completed..")

        log_metric("is-success", "OK")

# Invoke the main function
prepare_retail_data()
```

dbnd_tracking

Initializes tracking for your pipeline or application, configuring Databand settings and logging execution details.

Usage:

```
with dbnd_tracking(conf={...}, job_name="job_name", run_name="run_name"):
    # Pipeline code
```

task

Marks a function as a Databand task, enabling tracking and monitoring of individual steps in your pipeline.

Usage:

```
@task
def my_task_function():
```

dataset_op_logger

Logs operations on datasets, including schema and statistics.

Usage:

```
with dataset_op_logger(dataset_name, operation_type) as logger:
    logger.set(data=my_dataframe)
```

log_metric

Records custom metrics to track performance or other quantitative data during execution.

Usage:

```
log_metric("metric_name", metric_value)
```

log_dataframe
Logs details about a DataFrame, such as schema and statistics, for monitoring data transformations.

Usage:

```
log_dataframe("dataframe_name", my_dataframe, with_schema=True, with_stats=True)
```

To submit a Spark job, do the following steps:

1. Create a `payload.json` file that includes the package details as explained here.
2. Submit the Spark job to the watsonx.data application.

```
curl -ivk -X POST \
  -d @payload.json \
  -H "Content-Type: application/json" \
  -H "LhInstanceId: <InstanceId>" \
  -H "Authorization: ZenApiKey <Token>" \
  "<watsonx_application_endpoint>"
```

For more Information and examples, see:

- IBM Data Observability by Databand
- For PySpark applications: Tracking PySpark
- For Spark(Java/Scala): Tracking Spark(Scala/Java)

Enhancing Spark application submission using Spark access control extension

When you submit a Spark application that uses external storage buckets registered in watsonx.data, Spark access control extension allows additional authorization thereby enhancing security. If you enable the extension in the spark configuration, only authorized users are allowed to access and operate watsonx.data catalogs through Spark jobs.

You can create data policies to grant or deny access for catalog, schema, and table even column to a user or user group. Besides data level authorization, storage privilege is also considered. For more information related to the access control checks on catalogs, buckets, schemas and tables, see Access.

Before you begin

1. Create Cloud Object Storage to store data used in the Spark application. To create Cloud Object Storage and a bucket, see Creating a storage bucket. You can provision two buckets, data-bucket to store watsonx.data tables and application bucket to maintain Spark application code.
2. Register Cloud Object Storage bucket in watsonx.data. For more information, see Adding bucket catalog pair.
3. Upload the Spark application to the storage, see Uploading data.
4. You must have `IAM administrator` role or `MetastoreAdmin` role, for creating schema or table inside watsonx.data.

Procedure

Spark access control extension supports native Spark and external Spark.

1. To enable the Spark access control extension, you must update the Spark configuration with the following :

```
add authz.IBMSparkACExtension to spark.sql.extensions.
```

2. Save the following Python application as `iceberg.py`.

```
from pyspark.sql import SparkSession
import os

def init_spark():
    spark = SparkSession.builder \
        .appName("lh-spark-app") \
        .enableHiveSupport() \
        .getOrCreate()
    return spark

def create_database(spark):
    # Create a database in the lakehouse catalog
    spark.sql("create database if not exists lakehouse.demodb LOCATION 's3a://lakehouse-
bucket/'")

def list_databases(spark):
    # list the database under lakehouse catalog
    spark.sql("show databases from lakehouse").show()

def basic_iceberg_table_operations(spark):
    # demonstration: Create a basic Iceberg table, insert some data and then query table
    spark.sql("create table if not exists lakehouse.demodb.testTable(id INTEGER, name
VARCHAR(10), age INTEGER, salary DECIMAL(10, 2)) using iceberg").show()
    spark.sql("insert into lakehouse.demodb.testTable values(1,'Alan',23,3400.00),
(2,'Ben',30,5500.00),(3,'Chen',35,6500.00)")
    spark.sql("select * from lakehouse.demodb.testTable").show()

def create_table_from_parquet_data(spark):
    # load parquet data into dataframe
    df = spark.read.option("header",True).parquet("file:///spark-vol/
yellow_tripdata_2022-01.parquet")
    # write the dataframe into an Iceberg table
    df.writeTo("lakehouse.demodb.yellow_taxi_2022").create()
    # describe the table created
    spark.sql('describe table lakehouse.demodb.yellow_taxi_2022').show(25)
    # query the table
    spark.sql('select * from lakehouse.demodb.yellow_taxi_2022').count()

def ingest_from_csv_temp_table(spark):
    # load csv data into a dataframe
    csvDF = spark.read.option("header",True).csv("file:///spark-vol/zipcodes.csv")
    csvDF.createOrReplaceTempView("tempCSVTable")
    # load temporary table into an Iceberg table
    spark.sql('create or replace table lakehouse.demodb.zipcodes using iceberg as select *
from tempCSVTable')
    # describe the table created
    spark.sql('describe table lakehouse.demodb.zipcodes').show(25)
    # query the table
    spark.sql('select * from lakehouse.demodb.zipcodes').show()

def ingest_monthly_data(spark):
    df_feb = spark.read.option("header",True).parquet("file:///spark-vol/
yellow_tripdata_2022-02.parquet")
    df_march = spark.read.option("header",True).parquet("file:///spark-vol/
yellow_tripdata_2022-03.parquet")
    df_april = spark.read.option("header",True).parquet("file:///spark-vol/
yellow_tripdata_2022-04.parquet")
    df_may = spark.read.option("header",True).parquet("file:///spark-vol/
yellow_tripdata_2022-05.parquet")
    df_june = spark.read.option("header",True).parquet("file:///spark-vol/
yellow_tripdata_2022-06.parquet")
    df_q1_q2 = df_feb.union(df_march).union(df_april).union(df_may).union(df_june)
    df_q1_q2.write.insertInto("lakehouse.demodb.yellow_taxi_2022")

def perform_table_maintenance_operations(spark):
    # Query the metadata files table to list underlying data files
    spark.sql("SELECT file_path, file_size_in_bytes FROM
lakehouse.demodb.yellow_taxi_2022.files").show()
    # There are many smaller files compact them into files of 200MB each using the
    # `rewrite_data_files` Iceberg Spark procedure
    spark.sql(f"CALL lakehouse.system.rewrite_data_files(table => 'demodb.yellow_taxi_2022',
options => map('target-file-size-bytes','209715200'))").show()
    # Again, query the metadata files table to list underlying data files; 6 files are
compacted
    # to 3 files
    spark.sql("SELECT file_path, file_size_in_bytes FROM
lakehouse.demodb.yellow_taxi_2022.files").show()
    # List all the snapshots
    # Expire earlier snapshots. Only latest one with compacted data is required
    # Again, List all the snapshots to see only 1 left
    spark.sql("SELECT committed_at, snapshot_id, operation FROM
lakehouse.demodb.yellow_taxi_2022.snapshots").show()
    #retain only the latest one
    latest_snapshot_committed_at = spark.sql("SELECT committed_at, snapshot_id, operation
FROM lakehouse.demodb.yellow_taxi_2022.snapshots").tail(1)[0].committed_at
    print (latest_snapshot_committed_at)
    spark.sql(f"CALL lakehouse.system.expire_snapshots(table
=> 'demodb.yellow_taxi_2022',older_than => TIMESTAMP
'{latest_snapshot_committed_at}',retain_last => 1)").show()
    spark.sql("SELECT committed_at, snapshot_id, operation FROM
lakehouse.demodb.yellow_taxi_2022.snapshots").show()
    # Removing Orphan data files
    spark.sql(f"CALL lakehouse.system.remove_orphan_files(table =>
'demodb.yellow_taxi_2022')").show(truncate=False)
    # Rewriting Manifest Files
    spark.sql(f"CALL lakehouse.system.rewrite_manifests('demodb.yellow_taxi_2022')").show()

def evolve_schema(spark):
    # demonstration: Schema evolution
    # Add column fare_per_mile to the table
    spark.sql('ALTER TABLE lakehouse.demodb.yellow_taxi_2022 ADD COLUMN(fare_per_mile
double)')
    # describe the table
    spark.sql('describe table lakehouse.demodb.yellow_taxi_2022').show(25)

def clean_database(spark):
    # clean-up the demo database
    spark.sql('drop table if exists lakehouse.demodb.testTable purge')
    spark.sql('drop table if exists lakehouse.demodb.zipcodes purge')
    spark.sql('drop table if exists lakehouse.demodb.yellow_taxi_2022 purge')
    spark.sql('drop database if exists lakehouse.demodb cascade')

def main():
    try:
        spark = init_spark()
        create_database(spark)
        list_databases(spark)
        basic_iceberg_table_operations(spark)
        # demonstration: Ingest parquet and csv data into a watsonx.data Iceberg table
        create_table_from_parquet_data(spark)
        ingest_from_csv_temp_table(spark)
        # load data for the month of February to June into the table yellow_taxi_2022
created above
        ingest_monthly_data(spark)
        # demonstration: Table maintenance
        perform_table_maintenance_operations(spark)
        # demonstration: Schema evolution
        evolve_schema(spark)
    finally:
        # clean-up the demo database
        clean_database(spark)
        spark.stop()

if __name__ == '__main__':
main()
```

3. To submit the Spark application, specify the parameter values and run the following curl command. The following example shows the command to submit `iceberg.py` application.

```
curl --request POST   --url https://<cpd_host_name>/lakehouse/api/v2/spark_engines/
<spark_engine_id>/applications \
   --header 'Authorization: Bearer <token>'   --header 'Content-Type: application/json'   --
header 'Lhinstanceid: <instance_id>'   --data '{
  "application_details": {
  "conf": {
      "spark.hadoop.fs.s3a.bucket.<wxd-data-bucket-name>.endpoint": "<wxd-data-bucket-
endpoint>",
      "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.endpoint": "<COS_ENDPOINT>",
      "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.secret.key": "<COS_SECRET_KEY>",
      "spark.hadoop.fs.cos.<COS_SERVICE_NAME>.access.key": "<COS_ACCESS_KEY>"
      "spark.sql.catalogImplementation": "hive",

"spark.sql.extensions":"org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions,aut
hz.IBMSparkACExtension",
      "spark.sql.iceberg.vectorization.enabled":"false",
        "spark.sql.catalog.<wxd-bucket-catalog-
name>":"org.apache.iceberg.spark.SparkCatalog",
      "spark.sql.catalog.<wxd-bucket-catalog-name>.type":"hive",
      "spark.sql.catalog.<wxd-bucket-catalog-name>.uri":"thrift://<wxd-catalog-metastore-
host>",
      "spark.hive.metastore.client.auth.mode":"PLAIN",
      "spark.hive.metastore.client.plain.username":"<username>",
      "spark.hive.metastore.client.plain.password":"xxx",
      "spark.hive.metastore.use.SSL":"true",
      "spark.hive.metastore.truststore.type":"JKS",
      "spark.hive.metastore.truststore.path":"<truststorepath>",
      "spark.hive.metastore.truststore.password":"changeit",
        "spark.hadoop.fs.s3a.bucket.<wxd-data-bucket-
name>.aws.credentials.provider":"com.ibm.iae.s3.credentialprovider.WatsonxCredentialsProvider
",
        "spark.hadoop.fs.s3a.bucket.<wxd-data-bucket-
name>.custom.signers":"WatsonxAWSV4Signer:com.ibm.iae.s3.credentialprovider.WatsonxAWSV4Signe
r",
        "spark.hadoop.fs.s3a.bucket.<wxd-data-bucket-name>.s3.signing-
algorithm":"WatsonxAWSV4Signer",
        "spark.hadoop.wxd.cas.endpoint":"<cas_endpoint>/cas/v1/signature",
        "spark.hadoop.wxd.instanceId":"<instance_id>",
        "spark.hadoop.wxd.apikey":"ZenApiKey xxx",
        "spark.wxd.api.endpoint":"<wxd-endpoint>",
        "spark.driver.extraClassPath":"opt/ibm/connectors/wxd/spark-authz/cpg-client-1.0-jar-
with-dependencies.jar:/opt/ibm/connectors/wxd/spark-authz/ibmsparkacextension_2.12-1.0.jar",
        "spark.sql.extensions":"<required-storage-support-
extension>,authz.IBMSparkACExtension"

    },
    "application": "cos://<BUCKET_NAME>.<COS_SERVICE_NAME>/<python_file_name>",
  }
}
```

Parameter values:

- *<token>* : To get the access token for your service instance. For more information about generating the token, see Generating a bearer token.
- *<instance_id>* : The instance ID from the watsonx.data cluster instance URL. For example, `1609968977179454`.
- *<wxd-data-bucket-endpoint>*: The host name of the endpoint for accessing the data bucket mentioned above. Example, `s3.us-south.cloud-object-storage.appdomain.cloud` for a Cloud Object storage bucket in us-south region.
- *<wxd-bucket-catalog-name>*: The name of the catalog associated with the data bucket.
- *<wxd-catalog-metastore-host>*: The metastore associated with the registered bucket.
- *<cos_bucket_endpoint>* : Provide the **Metastore host** value. For more information, see storage details.
- *<access_key>* : Provide the access_key_id. For more information, see storage details.
- *<secret_key>* : Provide the secret_access_key. For more information, see storage details.
- *<truststorepath>* : The file path where truststore certificate is available. For native Spark engine, the value is `../../../../../../../../../../opt/ibm/jdk/lib/security/cacerts`. For external Spark engine, the value is the truststore containing WXD HMS certs.
- *<cas_endpoint>* : The Data Access Service (DAS) endpoint. To get the DAS endpoint, see Getting DAS endpoint.
- *<username>* : The username for your watsonx.data instance. Here, ibmlhapikey.
- *<ZenApiKey>* : The base64 encoded `ibmlhapikey_<user_id>:<IAM_APIKEY>. Here, <user_id> is the IBM Cloud id of the user whose apikey is used to access the data bucket. To generate API key, log in into the watsonx.data console and navigate to `Profile > Profile and Settings > API Keys` and generate a new API key.
- *<OBJECT_NAME>*: The IBM Cloud Object Storage name.
- *<BUCKET_NAME>*: The storage bucket where the application file resides.
- *<COS_SERVICE_NAME>*: The Cloud object Storage service name.
- *<python file name>* : The Spark application file name.

Limitations:

a. The user must have full access to create schema and table.

b. To create data policy, you must associate the catalog to Presto engine.
c. If you try to display schema that is not existing, the system throws `nullpointer` issue.

Chapter 10. Working with Milvus

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Adding a Milvus service

Milvus is a vector database that stores, indexes, and manages massive embedding vectors that are developed by deep neural networks and other machine learning (ML) models. It is developed to empower embedding similarity search and AI applications. Milvus makes unstructured data search more accessible and consistent across various environments.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

About this task

You can add Milvus as a service in IBM watsonx.data through web console by using the following steps.

Procedure

1. Log in to watsonx.data console.
2. From the navigation menu, select **Infrastructure Manager**.
3. To define and connect to a service, click **Add component**, select **Milvus**, and click **Next**.
4. In the **Add component - Milvus** window, provide the following details.

Field	Description
Display name	Enter the Milvus service name to be displayed on the screen.
Size	Select the suitable size. • **Small:** Recommended for **10 million vectors**, 64 index parameters, 1024 segment size, and 384 dimensions. • **Medium:** Recommended for **50 million vectors**, 64 index parameters, 1024 segment size, and 384 dimensions. • **Large:** Recommended for **100 million vectors**, 64 index parameters, 1024 segment size, and 384 dimensions.
Add storage bucket	For **Small**, **Medium**, and **Large** sizes, you must associate an external storage. For **Starter** size, you can use an IBM-managed storage or an external storage. **Note:** To associate an external storage, you must have the storage configured.
Path	For external storages, specify the path where you want to store vectorized data files.

Important:

- If the schema of the collection changes (an increase in the number of fields in a collection or increase in the size of the `varchar` field beyond 256 characters, or if multiple vector fields are added into the collection), the number of records might decrease.
- You must provide the endpoint for storages used by Milvus with the region for region-specific storages like S3 and without trailing slashes. For example:

```
https://s3.<REGION>.amazonaws.com
```

For more information about adding external storages, see "Adding a storage-catalog pair" on page 306.

5. Click **Provision**.

Connecting to Milvus service

You can connect to a Milvus service by using a client program or a gRPC URI after provisioning Milvus as a service in IBM watsonx.data through the web console.

watsonx.data on Red Hat OpenShift

Before you begin

To connect to the Milvus service from a client program, make sure that the following items are installed or available:

-
- Python and Pymilvus package. See About PyMilvus.
- Hostname and port for the Milvus server or workstation where the watsonx.data instance is installed.
- Certificates served by the Milvus server to establish the trust.
- Authorized user credentials to access the Milvus server.
- Milvus server name.

About this task

Connect to a Milvus service by using the following steps:

Procedure

1. Provision a Milvus service in watsonx.data through the web console. See "Adding a Milvus service" on page 530.
2. Obtain a self-signed certificate for the route. Run the following command from the terminal of the client machine where PyMilvus SDK is installed.

   You must set the `host` and `port` variables also. To get the GRPC host, from the navigation menu, go to **Infrastructure manager** and click your Milvus service to open the **Details** page. The host is available under **GRPC host**. The port must be set to 443. Alternatively, go to **Configurations** > **Connect information** > **Engine and service connection details** and expand the Milvus service from the list.

   ```
   host=<GRPC host>
   port=<GRPC port>
   echo QUIT | openssl s_client -showcerts -servername $host -connect $host:$port | openssl
   x509 -out milvus-grpc.crt
   ```

   Expected output:

   ```
   depth=0 CN = ibm-lh-tls-secret
   verify error:num=20:unable to get local issuer certificate
   verify return:1
   depth=0 CN = ibm-lh-tls-secret
   verify error:num=21:unable to verify the first certificate
   verify return:1
   DONE
   ```

   Note: It is an expected behavior to have the errors in the output because the `openssl s_client` fails to verify the server's certificate as it is a self-signed certificate.

   The certificate is saved to **`milvus-grpc.crt`** file.

Tip: Alternatively, it is possible to fetch the certificates directly through Milvus. Use the following Python snippet to fetch the certificates.

```
import ssl
from pymilvus import connections

host, port = ...
user, password = ...

with open('milvus-grpc.crt', 'w') as f:
    f.write(ssl.get_server_certificate((host, port)))

print("start connecting to Milvus")
connections.connect("default",
    host=host, port=port,
    secure=True,
    server_name=host,
    server_pem_path="milvus-grpc.crt",
    user=user, password=password,
)
```

3. Run the following commands by using Python SDK (PyMilvus) to connect with Milvus for gRPC route:

```
connections.connect(
    alias='default',
    secure=True, server_pem_path="<path/to/grpc_certificate>", server_name="<grpc_route>",
    host='<grpc_route>',
    port='443',
    user='<CPD_username>',
    password='<CPD_password>')
```

For example:

```
connections.connect(
    alias='default',
    secure=True, server_pem_path="/root/cert/milvus-grpc.crt", server_name="ibm-lh-lakehouse-
milvus475.milvus.apps.keyword.cp.fyre.ibm.com",
    host='ibm-lh-lakehouse-milvus475.milvus.apps.keyword.cp.fyre.ibm.com',
    port='443',
    user='admin',
    password='v4B##..##PBY')
```

Note:

- The parameters **`secure=True`** and **`server_pem_path`** are specific to SSL enabled in Milvus.
- The route that is obtained with `-grpc` is the route for the gRPC server. You can also use `-rest` to obtain the route for the REST server.

  For more information, see Milvus Docs.
- Use the GRPC endpoints for SDKs and `https` endpoints for API calls.

Tip: Milvus can connect to an unactivated storage. The storage is activated only if it has a catalog associated. This is not mandatory.

What to do next

You can perform the following operations after establishing a connection with a Milvus service:

- Manage databases

  Note: A Milvus cluster supports a maximum of 64 databases. For more information, see Manage Databases.
- Manage collections

  Note: A Milvus cluster supports a maximum of 65,536 collections. For more information, see Manage Collections.
- Manage partitions

  Note: A Milvus cluster supports a maximum of 4095 partitions. For more information, see Manage Partitions.

- Manage data

  For more information, see:

  - Insert, Upsert & Delete
  - Data Import
- Manage indexes

  For more information, see Manage Indexes.
- Search and Query

  For more information, see Search, Query & Get.

For more information, see Milvus Docs.

Connecting to a Milvus service in watsonx.data developer edition

You can establish connection and start the Milvus service in IBM watsonx.data developer edition from a client program by following the steps in this procedure.

Note: Milvus is not supported in M1/M2 Mac OS, even if Rosetta emulation is enabled.

watsonx.data Developer edition

Before you begin

Install watsonx.data developer edition. For more information, see Installing the watsonx.data developer version.

Procedure

Complete the following steps to configure Milvus for using it with watsonx.data developer edition.

a. Run the following command to do the one-time setup for Milvus.

```
ibm-lh-dev/bin/setup-milvus
```

   Note: The command `ibm-lh-dev/bin/setup-milvus` is for the one-time setup.

   You need to run this command again only if you are upgrading watsonx.data developer edition and also want to upgrade the Milvus image.

b. Start Milvus in watsonx.data.

```
ibm-lh-dev/bin/start-milvus
```

   You can use Milvus service at `localhost:19530`.

Note: Upgrading watsonx.data development edition from 1.1.3 to 1.1.4 can potentially impact its interaction with existing Milvus collections, especially if Milvus was added and data was loaded in watsonx.data version 1.1.3. Follow the following steps prior to loading data in Milvus in 1.1.3 development version.

a. Edit the `ibm-lh-dev/etc/ibm-lh-etcd.conf` file and add the following line:

```
mnt_dir=/etcd
```

b. Restart the Milvus and etcd services.

What to do next

Run the following command to shut down the Milvus service.

```
ibm-lh-dev/bin/stop-milvus
```

Administering and managing various tasks in Milvus

Milvus is an open source vector database that is designed to efficiently handle large-scale vector similarity searches and storage. It empowers embedding the similarity search for AI applications. Milvus makes unstructured data search more accessible and consistent across various environments.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Creating a collection in Milvus

In Milvus, collections are used to store and manage entities. A **collection** in Milvus is equivalent to a table in a relational database management system (RDBMS).

See Create Collection for creating collection in Milvus.

Inserting data in Milvus

Milvus supports default values for scalar fields except a primary key field. Some fields can be empty during data insert.

For more information, see Insert entities.

Note: It is recommended to insert your data in batches due to the following reasons:

- The number of vectors that can be ingested in a single GRPC API call is limited by the maximum message size that is allowed in Kafka. In IBM Cloud, the maximum limit of message size is limited to 1 MB.
- The maximum number of rows that can be inserted at a time depends on the total size of the data you are trying to ingest. The number is inversely proportional to the dimensions of the vector and the presence of non-vector fields in the row.

Note: Use the bulk insert API to insert the data sets larger than 500,000 vectors. The bulk insert API performs better than the batch insert API when ingesting larger data sets.

If you are using batch insert API, manually flush the collection after every 500,000 rows.

For more information, see Bulk Insert API.

Creating indexes in Milvus

Create an index before you conduct the Approximate Nearest Neighbor (ANN) search in Milvus.

IBM officially supports the following indexes:

Note: Indexes that are not listed in this list might work, but are not validated by IBM.

- `HNSW`
- `SCANN`
- `FLAT`
- `IVF_FLAT`
- `IVF_PQ`

You can create the index by specifying the vector field name and index parameters. For more information, see Build an index on vectors

Conducting a vector similarity search in Milvus

In Milvus, you can conduct a vector similarity search after you prepare the parameters for your search scenario.

- For more information about single-vector and multi-vector search, see Vector similarity search.

- For more information about hybrid (multi-vector) search, see Hybrid search.

Conducting a query based on scalar filtering in Milvus

In Milvus, you can conduct a query based on scalar filtering.

For more information, see Get & scalar query.

You can do the following search types:

- Range search: To find vectors within a specific distance range from the query vector. For more information, see Range search.
- Grouping search: To get results based on a specific field to ensure diversity in the results. For more information, see Grouping search.

Note: When running a search query with scalar filtering on a large data set, it is important to adjust the limit parameter to manage query results effectively. Use the following approach to set the limit parameter:

```
hello_milvus.query(expr=query_condition, output_fields=["random", "embeddings"],
limit=100,offset=0)
```

The sum of the values of **`limit`** and **`offset`** parameters must be in the range of [1, 16384].

Deleting entities from Milvus by using primary key

In Milvus, you can delete the entities by using primary key. Prepare the Boolean expression that filters the entities to delete.

For more information, see Delete entities.

Dropping a collection from Milvus

CAUTION: Dropping a collection from Milvus is irreversible. You cannot recover the deleted data.

Add the following to your `.ipynb` or Python script to drop a collection.

```
from pymilvus import utility
utility.drop_collection(<collection name>)
```

Remember: Following are some best practices:

- If there are long Varchar fields (greater than 256 characters), keep non-vector fields outside of Milvus. You can keep them in a COS bucket or storage bucket and perform a referential search.
- Make sure that the PyMilvus version is 2.4.0 or later. Milvus 2.4.0 or later versions support sparse vector search, hybrid search (`sparse_dense`), and multi vector search.
- While loading collections by using batch insert, follow the pattern: Insert in a batch, release the memory, and then repeat the process.
- Don't ingest in parallel to all of the collections at once. Ingest sequentially and flush between ingests.
- Each collection's maximum size should be corresponding to the maximum number of vectors supported in a T-shirt size.
- If you have multiple collections or partitions loaded, ensure that the sum of vectors in all loaded entities does not exceed the limit of the T-shirt size. You can still store more vectors than the T-shirt size limit as long as the number of vectors that are loaded is within the limit.
- Use API key instead of password for long-running workloads to improve performance. Using password might cause low queries per second (QPS) for long-running workloads.

Chapter 11. Workload Orchestration

Workload orchestration is a process of automating the execution of a sequence of tasks that needs to be run in an order. watsonx.data supports integration of various orchestration tools.

Orchestration by using Apache Airflow

Apache Airflow is an open source platform that allows you to create, schedule, and monitor workflow. Work-flows are defined as Directed Acyclic Graphs (DAGs) which consist of multiple tasks that are written by using Python code. Each task represents a discrete unit of work, such as running a script, querying a database, or calling an API. The Airflow architecture supports scaling and parallel execution, making it suitable for managing complex, data-intensive pipelines.

Apache airflow supports the following use cases:

- **ETL or ELT Pipelines** : Extracting data from various sources, transforming it, and loading it into the data warehouse.
- **Data Warehousing** : Scheduling regular updates and data transformations in a data warehouse.
- **Data Processing**: Orchestrating distributed data processing tasks across different systems.

Running workflow by using Apache airflow

The Extract, Transform, and Load (ETL) operations involves executing multiple Spark jobs either sequentially or concurrently. Using Directed Acyclic Graphs (DAG), you can design a workflow to automate the process. The topic considers a use case to run an Apache Airflow workflow, which executes tasks to ingest data to Presto in watsonx.data and query data from watsonx.data.

watsonx.data on Red Hat OpenShift

Before you Begin

- Install an Apache Airflow stand-alone instance on your computer.
- Install watsonx.data on your computer.
- Install `Pandas` and `Prestodb` libraries.
- watsonx.data console host information, `wxd_host`. For more information, see Getting connection information..
- API Keys for watsonx.data (`username` and `api_key`). For more information to generate API key, see Generating an API authorization token.
- Service Instance ID for watsonx.data (`wxd_instance_id`). Get the instance ID from the watsonx.data information page.
- Spark engine ID of the Spark engine `spark_engine_id`. For more information, see Getting connection information..
- Presto external URL from an active Presto engine `presto_ext_url`. The host URL of the Presto engine. For more information, see Getting connection information..
- SSL Certificate location, which is trusted by the system (if applicable).
- Catalog that is associated with Spark and Presto engine `catalog_name`.
- Storage name that is associated to the catalog `bucket_name`.
- Install the packages, Pandas, and Presto-python-client by using the command: `pip install pandas presto-python-client`.

Procedure

1. The use case considers a task to ingest the data to Presto. To do that, create a Spark application that ingests Iceberg data to the watsonx.data catalog. Here, the sample Python file `ingestion-job.py` is considered.

```
from pyspark.sql import SparkSession
import os, sys

def init_spark():
    spark = SparkSession.builder.appName("ingestion-demo").enableHiveSupport().getOrCreate()
    return spark

def create_database(spark,bucket_name,catalog):
    spark.sql("create database if not exists {}.demodb LOCATION 's3a://{}/
demodb'".format(catalog,bucket_name))

def list_databases(spark,catalog):
    # list the database under lakehouse catalog
    spark.sql("show databases from {}".format(catalog)).show()

def basic_iceberg_table_operations(spark,catalog):
    # demonstration: Create a basic Iceberg table, insert some data and then query table
    print("creating table")
    spark.sql("create table if not exists {}.demodb.testTable(id INTEGER, name VARCHAR(10),
age INTEGER, salary DECIMAL(10, 2)) using iceberg".format(catalog)).show()
    print("table created")
    spark.sql("insert into {}.demodb.testTable values(1,'Alan',23,3400.00),
(2,'Ben',30,5500.00),(3,'Chen',35,6500.00)".format(catalog))
    print("data inserted")
    spark.sql("select * from {}.demodb.testTable".format(catalog)).show()



def clean_database(spark,catalog):
    # clean-up the demo database
    spark.sql("drop table if exists {}.demodb.testTable purge".format(catalog))
    spark.sql("drop database if exists {}.demodb cascade".format(catalog))

def main(wxdDataBucket, wxdDataCatalog):
    try:
        spark = init_spark()

        create_database(spark,wxdDataBucket,wxdDataCatalog)
        list_databases(spark,wxdDataCatalog)
        basic_iceberg_table_operations(spark,wxdDataCatalog)


    finally:
        # clean-up the demo database
        clean_database(spark,wxdDataCatalog)
        spark.stop()

if __name__ == '__main__':
    main(sys.argv[1],sys.argv[2])
```

2. Upload the Python file `ingestion-job.py` into a storage volume. For more information about uploading data to storage volume, see Managing storage volumes. Save the display name of the storage volume (`data_volume_display_name`) for future reference.

3. Design a DAG workflow by using Python language and save the Python file to the Apache Airflow directory location, `$AIRFLOW_HOME/dags/` directory (Default value of `AIRFLOW_HOME` is set to `~/airflow`).

   The following is an example of a workflow, which executes tasks to ingest data to Presto in watsonx.data, and query data from watsonx.data.

   Save the file with the following content, as `wxd_pipeline.py`.

```
from datetime import timedelta, datetime
from time import sleep
import prestodb
import pandas as pd
import base64
import os
# The DAG object
from airflow import DAG
```

```
# Operators
from airflow.operators.python_operator import PythonOperator # type: ignore
import requests

# initializing the default arguments
default_args = {
    'owner': 'IBM watsonx.data',
    'start_date': datetime(2024, 3, 4),
    'retries': 3,
    'retry_delay': timedelta(minutes=5),
    'wxd_endpoint': 'https://<wxd_host>.com', # WXD Host URL
    'wxd_instance_id': '234234....33', # Instance id of wxd service instance
    'wxd_username': 'myuser1', # Your username
    'wxd_api_key': '23sdf....sdff', # API Key from profile
    'spark_engine_id': 'spark23', # Spark engine id
    'data_volume_display_name': 'my_data_vol::zen', # Data Volume display name
    'catalog_name': '<catalog_name>', # Catalog name where data will be ingested
    'bucket_name': '<bucket_name>', # Bucket associated with catalog
    'presto_ext_url': '<presto_external_url>', # Presto external URL of engine without https
    'cert_location': '<presto-certificate>.cert' # Presto SSL certificate
}

# Instantiate a DAG object
wxd_pipeline_dag = DAG('wxd_ingestion_pipeline_software',
        default_args=default_args,
        description='watsonx.data ingestion pipeline',
        schedule_interval=None,
        is_paused_upon_creation=True,
        catchup=False,
        max_active_runs=1,
        tags=['wxd', 'watsonx.data']
)

# Workaround: Enable if you want to disable SSL verification
os.environ['NO_PROXY'] = '*'

def _ingest_via_spark_engine():
    try:
        print('ingest__via_spark_engine')
        url = f"{default_args['wxd_endpoint']}/lakehouse/api/v2/spark_engines/
{default_args['spark_engine_id']}/applications"
        auth_str = base64.b64encode(f'{default_args["wxd_username"]}:
{default_args["wxd_api_key"]}'.encode('ascii')).decode("ascii")

        headers = {'Content-type': 'application/json', 'Authorization': f'ZenApiKey
{auth_str}', 'LhInstanceId': default_args['wxd_instance_id']}

        response = requests.post(url, None, {
            "application_details": {
                "conf": {
                    "spark.executor.cores": "1",
                    "spark.executor.memory": "1G",
                    "spark.driver.cores": "1",
                    "spark.driver.memory": "1G",
                    "spark.hadoop.wxd.apikey": f"ZenApiKey {auth_str}"
                },
                "application": "/app/ingestion-job.py",
                "arguments": [
                    default_args['bucket_name'],
                    default_args['catalog_name']
                ],
            },
            "volumes": [
                {
                    "name": default_args['data_volume_display_name'],
                    "mount_path": "/app"
                }
            ]
        } , headers=headers, verify=False)

        print("Response", response.content)
        return response.json()['application_id']
    except Exception as inst:
        print('Error')
        exit
        print(inst)


def _wait_until_job_is_complete(**context):
    try:
        print('wait_until_job_is_complete')
        application_id =
context['task_instance'].xcom_pull(task_ids='ingest_via_spark_engine')
        print(application_id)

        while True:
            url = f"{default_args['wxd_endpoint']}/lakehouse/api/v2/spark_engines/
{default_args['spark_engine_id']}/applications/{application_id}"

            auth_str = base64.b64encode(f'{default_args["wxd_username"]}:
{default_args["wxd_api_key"]}'.encode('ascii')).decode("ascii")
            headers = {'Content-type': 'application/json', 'Authorization': f'ZenApiKey
{auth_str}', 'LhInstanceId': default_args['wxd_instance_id']}

            response = requests.get(url, headers=headers, verify=False)
            print(response.content)
            data = response.json()

            if data['state'] == 'FINISHED':
                break
            elif data['state'] in ['STOPPED', 'FAILED', 'KILLED']:
                raise ValueError("Job failed: ", data)

            print('Job is not completed, sleeping for 10secs')
            sleep(10)
    except Exception as inst:
        print(inst)


def _query_presto():
    try:
        presto_pwd = base64.b64encode(f'{default_args["wxd_username"]}:
{default_args["wxd_api_key"]}'.encode('ascii'))

        with prestodb.dbapi.connect(
            host=default_args['presto_ext_url'],
            port=443,
            user=default_args['wxd_username'],
            catalog='tpch',
            schema='tiny',
            http_scheme='https',
            auth=prestodb.auth.BasicAuthentication('ibmlhapikey', presto_pwd)
        ) as conn:
            if default_args['cert_location'] != "":
                conn._http_session.verify = default_args['cert_location']
            df = pd.read_sql_query(f"select * from
{default_args['catalog_name']}.demodb.testTable limit 5", conn)

            with pd.option_context('display.max_rows', None, 'display.max_columns', None):
                print("\n", df.head())
    except Exception as inst:
        print(inst)


def start_job():
    print('Validating default arguments')

    if 'wxd_endpoint' not in default_args:
        raise ValueError('wxd_endpoint is mandatory')

    if 'wxd_username' not in default_args:
        raise ValueError('wxd_username is mandatory')

    if 'wxd_instance_id' not in default_args:
        raise ValueError('wxd_instance_id is mandatory')

    if 'wxd_api_key' not in default_args:
        raise ValueError('wxd_api_key is mandatory')

    if 'spark_engine_id' not in default_args:
        raise ValueError('spark_engine_id is mandatory')


start = PythonOperator(task_id='start_task', python_callable=start_job, dag=wxd_pipeline_dag)

ingest_via_spark_engine = PythonOperator(task_id='ingest_via_spark_engine',
python_callable=_ingest_via_spark_engine, dag=wxd_pipeline_dag)
wait_until_ingestion_is_complete =
PythonOperator(task_id='wait_until_ingestion_is_complete',
python_callable=_wait_until_job_is_complete, dag=wxd_pipeline_dag)
query_via_presto = PythonOperator(task_id='query_via_presto', python_callable=_query_presto,
dag=wxd_pipeline_dag)
```

```
start >> ingest_via_spark_engine >> wait_until_ingestion_is_complete >> query_via_presto
```

4. Log in to the Apache Airflow.
5. Search for `wxd_pipeline.py` job, enable the DAG from Apache Airflow console page to run the workflow.

Chapter 12. Semantic automation for data enrichment in watsonx.data

Semantic automation for data enrichment is a new feature introduced in IBM watsonx.data. This feature leverages generative AI with IBM Knowledge Catalog to understand your data on a deeper level and enhance data with automated enrichment to make it valuable for analysis.

Traditional or simple data might lack clear meaning or context. Semantic automation uses AI to analyze the data and add a semantic layer like labels, categories, or relationships to the data. This enriched data is easier to understand and used for tasks like machine learning or data visualization. Semantic automation adds additional instructions to your data, making it more insightful for the users.

Registering and activating semantic layer

To activate semantic enrichment in IBM watsonx.data, you must register IBM Knowledge Catalog through the web console.

Before you begin

To register and enable semantic layer in watsonx.data, make sure that the following are available.

- ZenAPI key (supported from watsonx.data 2.0.2) or API key (supported from watsonx.data 2.0.0) of the corresponding Cloud Pak for Data (CPD) user. For more information, see Using Authorization: ZenApiKey token.
- IBM Knowledge Catalog Standard or IBM Knowledge Catalog Premium within the CPD cluster. For more information, see IBM Knowledge Catalog on Cloud Pak for Data.

Important: The API key will be associated with a functional user in watsonx.data. This user acts as a communication channel between IKC and watsonx.data. The tables IKC can enrich are limited to those this specific watsonx.data user has permission to access. Additionally, any IKC objects like projects, catalogs, business terms, or actions like publishing, reviewing in watsonx.data will be attributed to this functional user.

Procedure

1. Log in to watsonx.data console.
2. From the navigation menu, select **Configurations** and click **Semantic layer integration** tile.
3. Click **Add a registration**.
4. In the **Access Management** window, enter the ZenAPI or API key in the **Zen API key** field.

   Note: You can generate and copy a new API key from the user profile settings of the CPD cluster. See Generate new API key.

5. Select a Presto engine from the engine list and click **Ok**. Semantic automation layer is active.

Enriching data with semantic automation layer

To enrich your data, IBM watsonx.data leverages Semantic Automation Layer (SAL) in IBM Knowledge Catalog.

Follow the procedure in this topic to enrich your data with business terms and descriptions using the semantic enrichment feature in watsonx.data.

Before you begin

- You have a registered semantic automation layer in watsonx.data.

- You have a CSV file in a simplified format with the following fields:
  - Name: The business term you want to define.
  - Artifact Type: Always "glossary_term".
  - Description: The explanation of the business term.

  Sample file format:

  ```
  Name,Artifact Type,Description
  Residence Address,glossary_term,"Identifies an Address at which an Individual dwells, for
  example John Doe Resides At 102 Oak Court."
  Involved Party Markets Product Limit Condition,glossary_term,"Identifies a Limit Condition
  that applies to the Involved Party's marketing of the Product; for example, minimum audience
  or venues."
  Social Security Number,glossary_term,The unique number assigned to an Individual by a
  governmental agency for the purposes of qualifying for Social Security benefits.
  Rating Provider,glossary_term,"Identifies a Rating Issuer that supplies the Rating; for
  example, Credit Agency XYZ Provides Rating For a customer's Credit Risk Rating."
  ```

Procedure

Users with the following roles can perform semantic enrichment in watsonx.data:

- **Admin or Metastore Admin:** These roles can register semantic automation layer and access the **Enrich Data** tab with all its functionality for data enrichment.
- **User or Metastore Viewer:** These roles cannot view the **Configuration** tab for semantic automation layer registration or the **Enrich Data** tab. However, they can see any published enriched information, such as business terms and descriptions, when browsing data in the tables.

1. Log in to watsonx.data console.
2. From the navigation menu, select **Data manager** and click **Enrich data** tab.
3. Click **Enrichment settings** and select **Manage glossary** to upload CSV file containing your business terms and descriptions.

   Note: For the LLM to suggest tags and descriptions for the tables and columns in watsonx.data, upload a CSV file with your business terms (tags) and their corresponding descriptions.
4. In the **Manage glossary** window, click **Upload glossary** and drag the CSV file to the box or click to upload.
5. Select merge option:
   - **Replace all values**: Overwrites all existing tags and descriptions.
   - **Replace only defined values**: Replaces existing values only if the term exists in the uploaded file.
   - **Replace only empty values**: Adds tags and descriptions from the file only to columns without existing ones.
6. Click **Upload glossary**. The glossary displays a list of tags and its descriptions.
7. Click **Enrichment settings** and select **Adjust thresholds** to modify the thresholds for different types of enrichment. Click **Save all the changes**.
8. Select the schemas you want to enrich.
9. Click the overflow menu corresponding to the selected schema and select **Run enrichment**. You can also select single or multiple schemas and click **Enrich**.

   Note: You can select single or multiple schemas and click Enrich. The enriched data will have the business terms and descriptions against each columns in each table inside the schema.
10. Click the overflow menu next to the chosen schema and select **View enrichment**. The page displays the list of tables within the schema.
11. Click on any table to view details.
12. Assign business terms and descriptions manually:
    a. Hover over a business term and select **View more**, select **Governance** and click **Assign business terms**.

    b. Choose the relevant term from the uploaded glossary and click **Assign**.

13. Add display name and descriptions manually:

    a. Hover over a business term and select **View more**, select **Details** and click **Edit** icon.

    b. Modify the **Display name** or **Description** and click **Save**.

14. Review the enriched columns and click the overflow menu next to a column or **More** option and select **Mark as reviewed**.
15. Go to the table view and check if the review status is completed.
16. Click the overflow menu next to the chosen schema and select **Publish**. The page displays the list of enriched tables within the schema.
17. Verify the enriched data in the **Browse data** tab of the **Data Manager** by selecting the schema you enriched on successful completion of publishing.

Chapter 13. Troubleshooting and support

This section describes how to *troubleshoot* and find *support* for IBM watsonx.data.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Unable to delete IBM watsonx.data developer package folder

When trying to delete `ibm-lh-dev` folder by a non-root user using Podman, the folder does not get deleted.

Symptoms

When you try to run `rm -rf <Path_to_ibm-lh-dev_folder>` command to delete `ibm-lh-dev` folder, you get a `Permission denied` error.

Resolving the problem

If you want to delete the `ibm-lh-dev` folder as a non-root user using Podman, run the following command:

```
podman unshare rm -rf <Path_to_ibm-lh-dev_folder>
```

Resolving Presto (Java) login issues

Issue: You cannot log in to Presto (Java) UI as a normal user due to authentication failure.

IBM watsonx.data developer edition

Symptoms

If you used Sudo to install the watsonx.data developer edition and later switched to a normal user, you might fail to log into Presto (Java) UI.

Resolving the problem

Modify the group ownership by running the following command:

```
chgrp -R username ibm-lh-dev
```

Case-sensitive search configuration with Presto (Java)

When you deal with character data, case sensitivity is important when you search for specific matches or patterns. However, not all databases and query engines behave in the same way. Some are case-insensitive by default, while others are not. The case-sensitive search can lead to unexpected results if the case of the data is not considered.

watsonx.data on Red Hat OpenShift

Symptoms

Following is an example of why you might need to take steps for case-sensitive search configuration.

You are accessing a MySQL database directly:

```
mysql> select * from state where name='iowa';
+------+----+--------------+
| name | id | abbreviation |
+------+----+--------------+
| Iowa | 19 | IA           |
+------+----+--------------+
1 row in set (0.00 sec)
```

MySQL is case-insensitive by default. Even though MySQL column contains the capitalized string 'Iowa', it still matched the query's restriction of 'Iowa' that is acceptable. In some use cases, it might lead to unexpected results.

When you use Presto (Java) to access the same MySQL data source, the behavior of search result is different, and arguably, in a more expected way:

```
presto:demo> select * from state where name='iowa';
 name | id | abbreviation
------+----+--------------
(0 rows)

Query 20201120_151345_00001_wjx6r, FINISHED, 1 node
Splits: 17 total, 17 done (100.00%)
0:07 [1 rows, 0B] [0 rows/s, 0B/s]

presto:demo> select * from state where name='Iowa';
 name | id | abbreviation
------+----+--------------
 Iowa | 19 | IA
(1 row)
```

You get a match with 'Iowa', and not with 'iowa'. Presto (Java) made this data source (MySQL) case-sensitive, even though it is the same database in both examples, with default configurations used.

Resolving the problem

Reconfigure case-sensitivity:

With an RDBMS like MySQL, you can configure the collation setting to control if you want case sensitivity or not. You can set the collation at the database creation or table creation level as a part of the CREATE statement. Or you can use ALTER to change the collation of a database, table or individual column.

Presto (Java) is not a database and does not store data to support collation and has no configuration parameter that controls the case-sensitivity. To manage case-sensitivity in Presto (Java), and mimic collation, you can rewrite the query to force case insensitivity explicitly by using:

- Simple `lower()` or `upper()` functions

Example:

```
select * from state where lower(name)='california';
    name
------------
 california
 California
 CALIFORNIA
(3 rows)
```

This query matched any uppercase or lowercase combination in the table, mimicking case insensitivity.

Or regular expressions.

Example:

```
select * from state where regexp_like(name, '(?i)california');
    name
------------
 california
 California
 CALIFORNIA
(3 rows)
```

The regular expression syntax **(?i)** means that matches are case-insensitive.

Ingestion job failure analysis

When an ingestion job is done through the IBM watsonx.data web console, the UI does not show any message if ingestion fails.

watsonx.data on Red Hat OpenShift

Symptoms

Attention: This topic is applicable only to watsonx.data v1.0.0 and v1.0.1. In watsonx.data v1.0.2, the issue is fixed and you can find the failure or error message by clicking the ingestion job ID from the web console UI.

When you try to run an ingestion job through the IBM watsonx.data web console and if it fails, the UI does not show any message or notification. You cannot find the cause of failure.

Resolving the problem

If an ingestion job fails, complete the following steps to find the cause of failure:

1. Navigate to the **Data manager** tab, click **Ingestion jobs** tab and select the failed job.
2. Record the creation time of failed job and the first 6 digits of the generated **Job ID**.

   For example, if the **Job ID** is **`ingestion-1688651654362`**, first 6 digits are **168865**.
3. Log in to `ocp` by using one of the following options:
   a. Run the following command to log in to the cluster by providing a username and password:

   ```
   ibm-lakehouse-manage login-to-ocp \
   --user=${OCP_USERNAME} \
   --password=${OCP_PASSWORD} \
   --server=${OCP_URL}
   ```

   b. Run the following command to log in to the cluster by providing a token:

   ```
   ibm-lakehouse-manage login-to-ocp \
   --server=${OCP_URL} \
   --token=${OCP_TOKEN}
   ```
4. Set the project by using the following command:

   ```
   oc project <PROJECT_CPD_INST_OPERANDS>
   ```
5. Use the following command with the first 6 digits of the generated **Job ID** to list down the ingestion jobs.

   ```
   oc get pods | grep ingest-remote-file | grep <first 6 digits of job id>
   ```
6. Identify the name of ingestion job pod from the list by using the recorded creation time of failed job.
7. Use the job pod name in the following command to see the output of the ingestion job along with the error message.

   ```
   oc logs <failed-job-pod-name>
   ```

Note: Contact IBM support for further assistance if you cannot resolve the issue.

Longer wait period on importing data

Longer wait period on restoring watsonx.data metadata, and the default buckets associated with it.

watsonx.data on Red Hat OpenShift

Symptoms

Longer wait period for completing the import process. It takes about 15 minutes to complete the import procedure and because of which data outage occurs during this period.

Resolving the problem

Wait about 15 minutes to complete the import procedure and clear the cache after every import process.

Scaling down does not work as expected

Because of a Kubernetes limitation of StatefulSets, you cannot scale down a StatefulSet when any of the stateful pods it manages is unhealthy.

Symptoms

When you attempt to increase the `presto_worker_replicas` to a number more than the system can handle, additional worker pods enter in the pending state. However, if you revert the value back the original number, the previously pending worker pods remain in **Pending** state indefinitely.

Resolving the problem

Delete the unhealthy StatefulSet and patch the engine with your new scale setting. The operator re-creates the StatefulSet on next reconcile with the new size.

watsonx.data deployment fails on OSV installed cluster

IBM watsonx.data deployment fails due to the presence of the OpenShift Virtualisation (OSV) operator in the Red Hat OpenShift Container Platform (OCP) cluster.

Symptoms

When you deploy watsonx.data service on OCP cluster that has the OpenShift Virtualisation (OSV) operator, the installation fails with the following log message.

```
ValueError: too many values to unpack (expected 2)\n", "module_stdout": "", "msg": "MODULE
FAILURE\nSee stdout/stderr for the exact error
```

Restriction: If you are installing Red Hat OpenShift Container Platform Version 4.14, do not install the KubeVirt HyperConverged Cluster Operator on the cluster. It can cause problems when installing some Cloud Pak for Data software.

Resolving the problem

1. Delete the **OSV hyperconverged** customer resource to make the OSV operator to stop running. To do that:

   ```
   oc get HyperConverged -A
   NAMESPACE        NAME                        AGE
   openshift-cnv    kubevirt-hyperconverged     11d
   ```

   ```
   oc delete HyperConverged {HyperConverged_instance_name} -n {HyperConverged_namespace}
   ```

2. Run the following command to verify that **OSV hyperconverged** resource is not present.

```
oc get crd | grep virtualmachineinstances.kubevirt.io
```

Reducing the duration of omrgc_spinlock_acquire calls for IBM watsonx.data developer version

Longer `omrgc_spinlock_acquire` calls slow down the performance of IBM watsonx.data developer version.

IBM watsonx.data developer edition

Symptoms

When you run large workloads by using IBM watsonx.data developer version, the `omrgc_spinlock_acquire` call takes longer to complete, making IBM watsonx.data slower.

Resolving the problem

Customize the Java virtual machine (JVM) parameters such that `omrgc_spinlock_acquire` calls do not take a long time.

1. Run the following command to create a directory inside the host location where IBM watsonx.data developer version is installed.

```
mkdir -p /root/ibm-lh-dev/localstorage/volumes/infra/ibm-lh-presto-config
```

2. Get the `jvm.config` file from the `/opt/presto/etc` directory inside the `ibm-lh-presto` container. Run the following command to copy the files from the container to the host location.

```
docker cp ibm-lh-presto:/opt/presto/etc/jvm.config /root/ibm-lh-dev/localstorage/volumes/
infra/ibm-lh-presto-config/jvm.config
```

3. Update the `jvm.config` file to include the following JVM parameters:
   - `-Xgc:tlhInitialSize=8096`
   - `-Xgc:tlhIncrementSize=16384`
   - `-Xgc:tlhMaximumSize=1048576`
4. Restart the `ibm-lh-presto` container by running the following command:

```
/root/ibm-lh-dev/bin/stop_service ibm-lh-presto
/root/ibm-lh-dev/bin/start_service ibm-lh-presto
```

5. Verify the `jvm.config` settings by running the following command:

```
-server

-Xmx16G

-XX:+UseG1GC

-XX:G1HeapRegionSize=32M

-XX:+UseGCOverheadLimit

-XX:+ExplicitGCInvokesConcurrent

-XX:+HeapDumpOnOutOfMemoryError

-XX:+ExitOnOutOfMemoryError

-XX:-UseBiasedLocking

-XX:ReservedCodeCacheSize=256M

-Djdk.nio.maxCachedBufferSize=2000000
```

```
-Djdk.attach.allowAttachSelf=true

-Duser.timezone=Etc/GMT

-javaagent:/opt/presto/lib/jmx_prometheus_javaagent-0.18.0.jar=9090:/opt/presto/etc/jmx-
exporter-config.yaml

-javaagent:/opt/presto/etc/setvendor.zip

-Djavax.net.ssl.trustStore=/mnt/infra/tls/truststore.jks

-Djavax.net.ssl.trustStorePassword=changeit

-Djavax.net.ssl.trustStoreType=jks

-Xgc:tlhInitialSize=8096

-Xgc:tlhIncrementSize=16384

-Xgc:tlhMaximumSize=1048576
```

Installing watsonx.data fails during storage outage time

The `ibm-lh-postgres-edb-1-initdb` pods show error status when installing watsonx.data during storage outage time.

Symptoms

When you deploy watsonx.data service during storage outage time, the installation fails with the following pod errors.

```
ibm-lh-postgres-edb-1-initdb-8nvzk            0/1     Error       0             26h
ibm-lh-postgres-edb-1-initdb-f7hsd            0/1     Error       0             26h
ibm-lh-postgres-edb-1-initdb-hpkns            0/1     Error       0             25h
ibm-lh-postgres-edb-1-initdb-pdmxd            0/1     Error       0             2d7h
ibm-lh-postgres-edb-1-initdb-sf299            0/1     Error       0             26h
```

The pod log displays the following message:

```
Error: stat /var/lib/postgresql/data/pgdata: permission denied
```

Resolving the problem

1. Run the following command to remove the old PostgreSQL cluster.

   ```
   oc delete clusters.postgresql.k8s.enterprisedb.io ibm-lh-postgres-edb -n ${instance-
   namespace}
   ```

2. You can do one of the following:

   a. Waiting for the watsonx.data operator pod to reconcile by itself.

   b. Run the following command to delete watsonx.data operator pod to trigger operator reconcilement immediately.

      ```
      oc delete $(oc get pod -l app.kubernetes.io/name=ibm-lakehouse -o name -n ${operator-
      namespace}) -n ${operator-namespace}
      ```

MinIO URLs allow listing for air-gapped CPD installations

This topic provides information to allow list MinIO URLs to ensure seamless connection with watsonx.data in air-gapped environments. If the URLs are not added to the allow list in your air-gapped cluster, you cannot establish the connections.

Symptoms

Air-gapped deployments require allow listing specific URLs before installing Cloud Pak for Data (CPD) and, subsequently, watsonx.data on CPD. While these URLs can be allow listed beforehand, there might be situations where they are added during or after the installation process.

Resolving the problem

You can add MinIO URLs to the allow list depending on where you are in the installation process:

1. Before starting CPD or watsonx.data installation.
   a. Identify the namespace where you plan to install watsonx.data.
   b. Construct the MinIO URL by adding the namespace in the following format:

      ```
      http://ibm-lh-lakehouse-minio-svc.{namespace}.svc.cluster.local:9000
      ```

      - Replace `{namespace}` with the actual namespace you obtained.
      - The port number is always 9000.

   c. Add the constructed URL to your allow list URLs for your specific proxy settings.
2. After starting CPD or watsonx.data installation.
   a. Access your watsonx.data instance.
   b. Navigate to the **Infrastructure Manager** tab.
   c. Click on any MinIO buckets. The MinIO URL will be displayed in the details tab against endpoint details in the format:

      ```
      http://ibm-lh-lakehouse-minio-svc.cpd-instance.svc.cluster.local:9000
      ```

   d. Copy the URL and add to your allow list URLs for your specific proxy settings.

Updating the configuration settings for Query Optimizer

Issue: The configuration settings for Query Optimizer might not be updated resulting in a poor performance of Query Optimizer.

Symptoms

The query results are not optimized since the configuration settings were not updated resulting in a unoptimized output.

Resolving the problem

You have to verify and update the configuration settings for Query Optimizer to improve the performance of Query Optimizer.

1. Run the following commands to login to the container and apply configuration settings:

   ```
   oc exec -it c-lakehouse-oaas-db2u-0 -n ${PROJECT_CPD_INST_OPERANDS} -- sh
   ```

2. Verify that the following registry variable settings and *OAAS db* configuration settings are updated. If the variables are not updated, do the following additional settings:
   a. Registry variable settings
      i) Run the following command to get the existing registry variable settings:

      ```
      su - ${DB2INSTANCE}
      db2 connect to ${DBNAME}
      db2set
      ```

      ii) Update the values as follows if they are not updated:

```
db2set DB2_ENABLE_PRESTO_MODE=true

db2set DB2_ENABLE_PRESTO_3PART_NAMES=true

db2set DB2_WORKLOAD=ANALYTICS

db2set DB2_REDUCED_OPTIMIZATION="EGAD 0,STARJN_CARD ON 1"

db2set DB2_EXTENDED_OPTIMIZATION="REDSCANELIM_UCP ON,PCD ON,EXP_GRPSETS_BYREPL
EAGGENF,MRG_DISJ_SCALARSQ_CASE ON,JFR ON,COLJOIN 1,PSQTGU ON,DISTINCT_ON_NULLARM_OF_OJ
ON,EXP_GRPSETS_SIMPL_GBY_EXPR ON,EXP_GRPSETS_WITH_OLAP ON,ENFD2GBCSEIN ON"

db2set DB2_SELECTIVITY=ALL

db2set DB2_CORRELATED_PREDICATES="UNIQUE_KEY_JOIN_ADJUST ON"

db2set DB2_OPTIMIZER_MODIFIERS="RSTFEXSTS ON,OJSJTE ON,CSE2OLAP ON,FJFR ON,DERSJF
ON,D2GB4ED ON,GBSJFD ON"

db2set DB2_UNION_OPTIMIZATION="OJ_UA_JPPD=ON"
```

b. *OAAS db* configuration settings

i) Run the following command to update configuration as follows if it is not updated:

```
db2 get db cfg
db2 update db cfg for OAAS using LARGE_AGGREGATION NO
```

ii)

```
bigsql stop
bigsql start
bigsql status
```

Re-enrichment not reflecting glossary updates in semantic automation

Issue: Updating a watsonx.data glossary in semantic automation does not impact previously enriched watsonx.data tables.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Symptoms

Metadata enrichment run determines that the data has already been enriched and skips re-evaluation. Therefore, subsequent metadata enrichment runs immediately gets terminated due to the existing enriched state.

Resolving the problem

Try the following to resolve the issue:

1. Locate the corresponding Semantic automation project in CPD for the affected watsonx.data table.
2. Delete the metadata enrichment asset associated with the Semantic automation project. This will also delete the metadata enrichment job.
3. Return to watsonx.data and re-enrich the table. The updated glossary terms will now be used for the new enrichment process.

Spark

This section list the troubleshooting topics for Spark.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Certificate error while connecting to Spark labs

When you try to establish a connection with Spark labs, "Certificate error" is displayed.

Your watsonx.data endpoint does not have a secure SSL certificate that is installed in the machine.

Symptoms

When you try to establish a connection with Spark labs, "Certificate error" is displayed.

Resolving the problem

To resolve this, you can configure the settings to disable SSL verification.

1. Open **Settings** from **Visual Studio Code**.
2. Search for **Verify SSL** and disable the feature.
3. Refresh Spark labs extension by clicking the **Refresh** icon from the left navigation pane and try reconnecting.

Timeout error while connecting to Spark labs

You fail to connect to Spark labs because of a timed-out connection to the host or the connection automatically drops after a few seconds.

Symptoms

`Timed-out connection to the host` error appears or the connection automatically drops after a few seconds.

Resolving the problem

Update the following settings from Visual Studio Code.

1. Open **Settings** from **Visual Studio Code**.
2. Search for **Use Exec Server** and select the check box to enable the feature.
3. Refresh Spark labs extension by clicking the **Refresh** icon from the left navigation pane and try reconnecting.

Permission denied error while connecting to Spark labs

When you try to establish a connection with Spark labs, a "Permission denied" error is displayed.

Symptoms

You receive permission denied error while connecting to Spark labs.

Causes

Your SSH connection is not using the right private key to authenticate with the Spark labs.

Resolving the problem

Ensure the following to resolve the issue.

1. The extension settings use the correct SSH private and public key pairs (**PrivateKeyPath** and **PublicKeyPath**).
2. Use the same public key in the **Settings** page and in the **Create Spark Lab** window.

3. Add an entry to your SSH configuration under ~/.ssh/config. Specify `<port_number>` and `~/.ssh/id_rsa` in the following configuration.

```
Host cpdenv
    HostName localhost
    Port <port_number> # Same port as spark-labs.localPort
    IdentityFile ~/.ssh/id_rsa # Path to your private key
```

Milvus

This section list the troubleshooting topics for Milvus.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Unable to access Milvus from a non-VM macOS system

Issue: If you try to access Milvus through VPN from a non-VM macOS system that uses pymilvus SDK, the connection might fail.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Symptoms

The connection might fail with the following error:

```
"MilvusException: (code=2, message=Fail connecting to server on your-host-name:443, illegal
connection params or server unavailable)>".
```

Resolving the problem

Try the following to resolve the issue:

1. Check whether the provisioned Milvus instance was deleted and a new one was created. In that case, the GRPC server and host URL might have changed.
2. Check whether the saved certificate in the system is a GRPC certificate and not an `https` certificate.
3. If step 1 and step 2 are verified as correct, add the following lines to your `.pynb` notebook or Python script and try again:

   a. `import os`

   b. `os.environ["GRPC_DNS_RESOLVER"] = "native"`

   For example:

```
import os
os.environ["GRPC_DNS_RESOLVER"] = "native"
connections.connect(
            alias='<db name>',
            host="<host>",
            port=443,
            secure=True,
            server_pem_path="<cert_path>",
            server_name="<host>",
            user="<username>",
            password="<password>")
```

Milvus becomes unresponsive if the loaded collections exceeds the capacity of the cluster

Milvus might become unresponsive if the consumed memory of the loaded collections exceeds the capacity of the cluster.

watsonx.data on Red Hat OpenShift

watsonx.data Developer edition

Symptoms

The query nodes crash and any further queries that are made are not completed.

Resolving the problem

If there are multiple collections that are loaded, release some of them to free up memory. In PyMilvus, you can do it using:

```
client.release_collection("collection_name")
```

Note: The released collections must be loaded again if you want to search them.

If a single collection doesn't fit in the memory, you cannot query the collection to retrieve data. To recover the collection:

- Increase the query node memory by doing the following:
  - Upgrade the size.
  - Increase the query node replicas.
- Reduce the collection size by doing the following:
  - Delete rows.
  - After upgrading the size, extract the required data from the collection, drop the collection, and then reset the size. Alternatively, continue to use the upgraded size without modifying the collection.

Engine failure status check

You can check the engine failure status by using the **`oc describe wxdengine`** command and by checking the `ibm-lakehouse-controller-manager` logs.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Procedure

1. Run the following command to display the `Status` and `Events` information of an engine.

   ```
   oc describe wxdengine <enginename> -n <operand namespace>
   ```

2. For more information, check the `ibm-lakehouse-controller-manager` pod logs in the operator namespace. This pod manages the `wxdengine` and other `wxd` resources. If an Ansible task fails, it is displayed in red color. Alternatively, use the following grep command:

   ```
   oc logs ibm-lakehouse-controller-manager-<auto-generated id key> -n <operator namespace> |
   grep <fail term> -C 10
   ```

   Note: You can search for `error`, `failure`, `failed`, or other similar terms. Replace *`<fail term>`* in the command with a term of your choice.

Chapter 14. References

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Getting connection information

You can now find the connectivity information for IBM watsonx.data in the **Connect information** tile in the **Configurations** page and in the **Instance details** page. You can copy JSON snippet and export JSON. This information is useful to create connections between IBM Cloud Pak for Data and IBM watsonx.data.

1. The **Connect information** tile in the **Configurations** page provides the following information:

- **Instance details:**
  - Host IP address
  - Port
  - Instance CRN
  - SSL certificate
- **Engine and service connection details:**
  - Select the checkbox for **Connecting from IBM Cloud Pak for Data** if you are connecting to IBM watsonx.data from IBM Cloud Pak for Data.
  - Select the checkbox for **Use internal hostname** for network efficiency.
  - You can select one engine for the JSON snippet.

2. The **Instance details** page provides the following details. To open the **Instance details**, click the `i` icon on the home page.

- – Instance ID
  - Data Access Service (DAS) endpoint
  - Common Policy Gateway (CPG) endpoint

SQL statements, data types and mixed-case behavior supported by Presto

This section provides details on SQL statements, data types and mixed-case feature flag behavior for databases in IBM watsonx.data.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Note: For more information on mixed-case feature flag behavior, supported SQL statements and supported data types matrices, see Support content.

Presto SQL statements

This section contains the list of SQL statements supported by Presto.

watsonx.data Developer edition

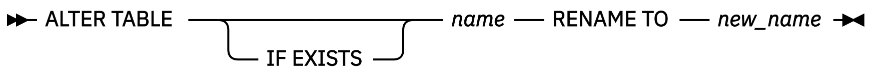
watsonx.data on Red Hat OpenShift

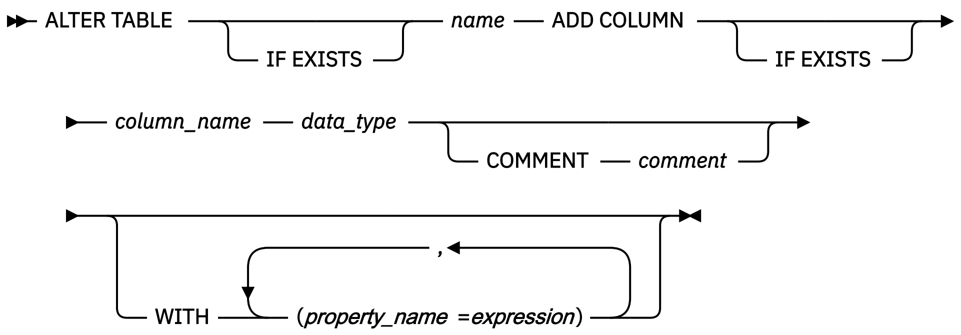
ALTER TABLE statement

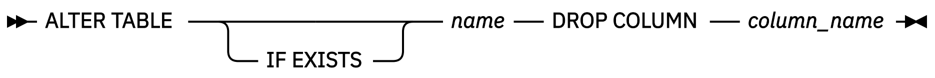
Synopsis

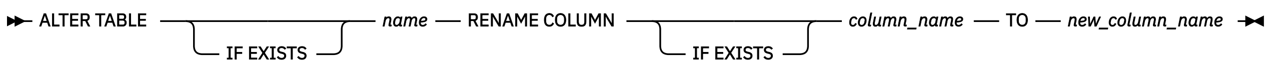
```
ALTER TABLE [ IF EXISTS ] name RENAME TO new_name
ALTER TABLE [ IF EXISTS ] name ADD COLUMN [ IF NOT EXISTS ] column_name data_type [ COMMENT
comment ] [ WITH ( property_name = expression [, ...] ) ]
ALTER TABLE [ IF EXISTS ] name DROP COLUMN column_name
ALTER TABLE [ IF EXISTS ] name RENAME COLUMN [ IF EXISTS ] column_name TO new_column_name
ALTER TABLE [ IF EXISTS ] name ALTER COLUMN column_name SET DATA TYPE new_type
```

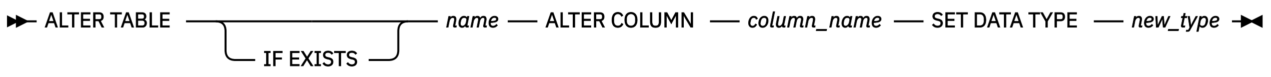
Syntax Diagram









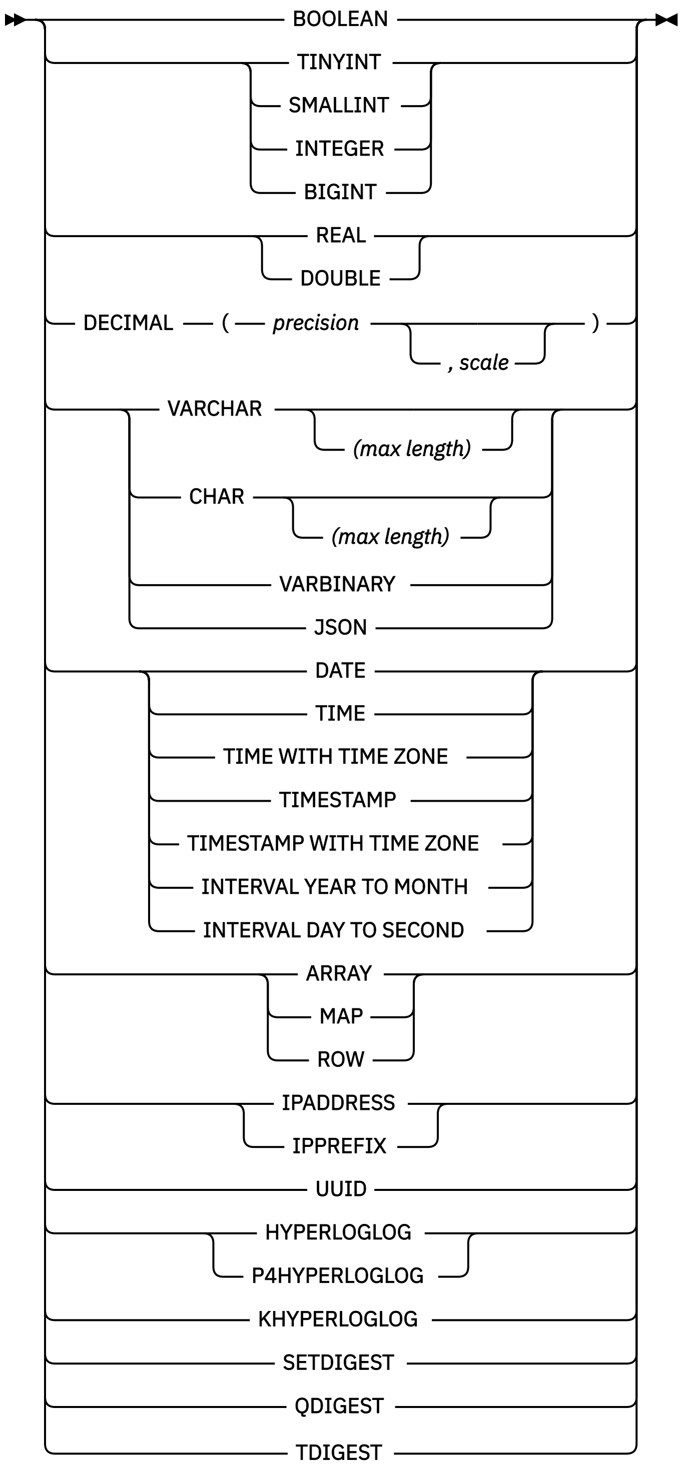


For information on reading syntax diagrams, see How to read syntax diagrams

Description

- Change the definition of an existing table.
- The optional `IF EXISTS` (when used before the table name) clause causes the error to be suppressed if the table does not exists.
- The optional `IF EXISTS` (when used before the column name) clause causes the error to be suppressed if the column does not exists.
- The optional `IF NOT EXISTS` clause causes the error to be suppressed if the column already exists.

DATA TYPE



Examples

Rename table `users` to `people`:

```
ALTER TABLE users RENAME TO people;
```

Rename table `users` to `people` if table `users` exists:

```
ALTER TABLE IF EXISTS users RENAME TO people;
```

Add column zip to the users table:

```
ALTER TABLE users ADD COLUMN zip varchar;
```

Add column zip to the users table if table users exists and column zip not already exists:

```
ALTER TABLE IF EXISTS users ADD COLUMN IF NOT EXISTS zip varchar;
```

Drop column zip from the users table:

```
ALTER TABLE users DROP COLUMN zip;
```

Drop column zip from the users table if table users and column zip exists:

```
ALTER TABLE IF EXISTS users DROP COLUMN IF EXISTS zip;
```

Rename column id to user_id in the users table:

```
ALTER TABLE users RENAME COLUMN id TO user_id;
```

Rename column id to user_id in the users table if table users and column id exists:

```
ALTER TABLE IF EXISTS users RENAME column IF EXISTS id to user_id;
```

Change type of column id to bigint in the users table:

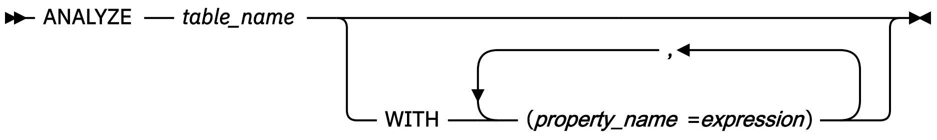
```
ALTER TABLE users ALTER COLUMN id SET DATA TYPE bigint;
```

ANALYZE statement

Synopsis

```
ANALYZE table_name [ WITH ( property_name = expression [, ...] ) ]
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

- Collects table and column statistics for a given table. Currently column statistics are only collected for primitive types.
- The optional WITH clause can be used to provide connector-specific properties. To list all available properties, run the following query:

  ```
  SELECT * FROM system.metadata.analyze_properties
  ```

- Currently, this statement is only supported by the Hive connector.

Examples

Analyze table web to collect table and column statistics:

```
ANALYZE web;
```

Analyze table `stores` in catalog `hive` and schema `default`:

```
ANALYZE hive.default.stores;
```

Analyze partitions `'1992-01-01', '1992-01-02'` from a Hive partitioned table `sales`:

```
ANALYZE hive.default.sales WITH (partitions = ARRAY[ARRAY['1992-01-01'], ARRAY['1992-01-02']]);
```

Analyze partitions with complex partition key (`state` and `city` columns) from a Hive partitioned table `customers`:

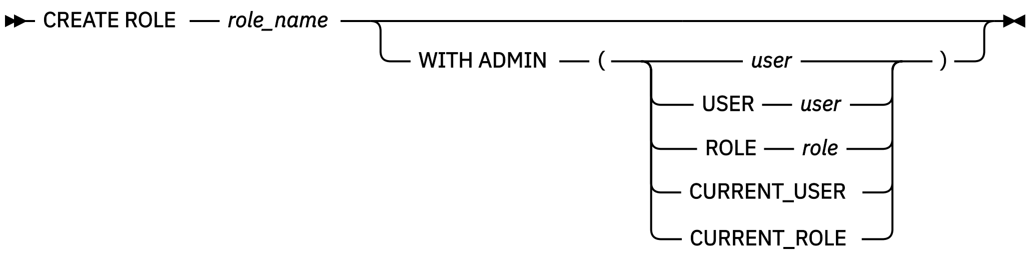
```
ANALYZE hive.default.customers WITH (partitions = ARRAY[ARRAY['CA', 'San Francisco'],
ARRAY['NY', 'NY']]);
```

CREATE ROLE statement

Synopsis

```
CREATE ROLE role_name
[ WITH ADMIN ( user | USER user | ROLE role | CURRENT_USER | CURRENT_ROLE ) ]
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

`CREATE ROLE` creates the specified role in the current catalog.

The optional `WITH ADMIN` clause causes the role to be created with the specified user as a role admin. A role admin has permission to drop or grant a role. If the optional `WITH ADMIN` clause is not specified, the role is created with current user as admin.

Examples

Create role `admin`

```
CREATE ROLE admin;
```

Create role `moderator` with admin bob:

```
CREATE ROLE moderator WITH ADMIN USER bob;
```

CREATE SCHEMA statement

Synopsis

```
CREATE SCHEMA [ IF NOT EXISTS ] schema_name
[ WITH ( property_name = expression [, ...] ) ]
```

Note: `schema_name` with following characters are not allowed:

- Schema names wrapped with quotation marks (example: use test instead of “test”)
- Schema names with special characters like semicolon(;), colon(:), single quotation mark('), or double quotation mark(")
- Schema names with leading space
- Schema names with leading special character exclamation mark(!)

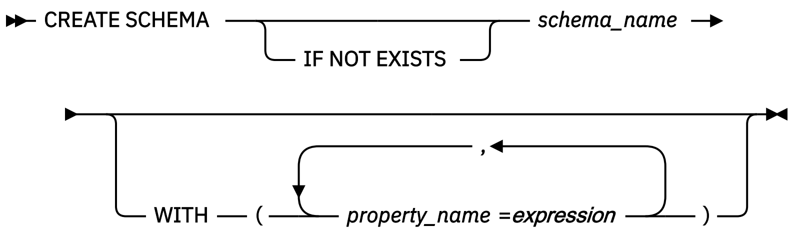
Note: The `schema_name` must be unique. Use unique names when schemas are created in different catalogs.

Note: It is mandatory to specify the location of the bucket with a subfolder to store tables when a schema is created. If location is not specified, then the behavior is unpredictable when a schema is created. The location must be enclosed in single quotation marks.

For example:

```
CREATE SCHEMA [ IF NOT EXISTS ] schema_name [ WITH (location: '<bucket_name>/subpath/)' ]
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Create an empty schema. A schema is a container that holds tables, views, and other database objects.

The clause `IF NOT EXISTS` causes the error to be suppressed if the schema exists.

The clause `WITH` can be used to set properties on the newly created schema. To list all available schema properties, run the following query:

```
SELECT * FROM system.metadata.schema_properties
```

Examples

Create a schema named `web` in the current catalog:

```
CREATE SCHEMA web
```

Create a schema named `sales` in the `hive` catalog:

```
CREATE SCHEMA hive.sales
```

Create a schema named `traffic` if it does not already exist:

```
CREATE SCHEMA IF NOT EXISTS traffic
```

CREATE TABLE statement

The CREATE TABLE statement defines a table. The definition must include its name and the names and attributes of its columns.

The Hive catalog supports two different types of tables:

- **Managed** : Indicates that the data in the table is owned and managed by the database manager. If you drop the table, the table definition and the data are removed from both the database manager and Hive catalogs.
- **External** : Indicates that the data in the table is not managed by the database manager. If you drop the table, the table definition is removed from both the database manager and Hive catalogs, but the data remains unaffected.

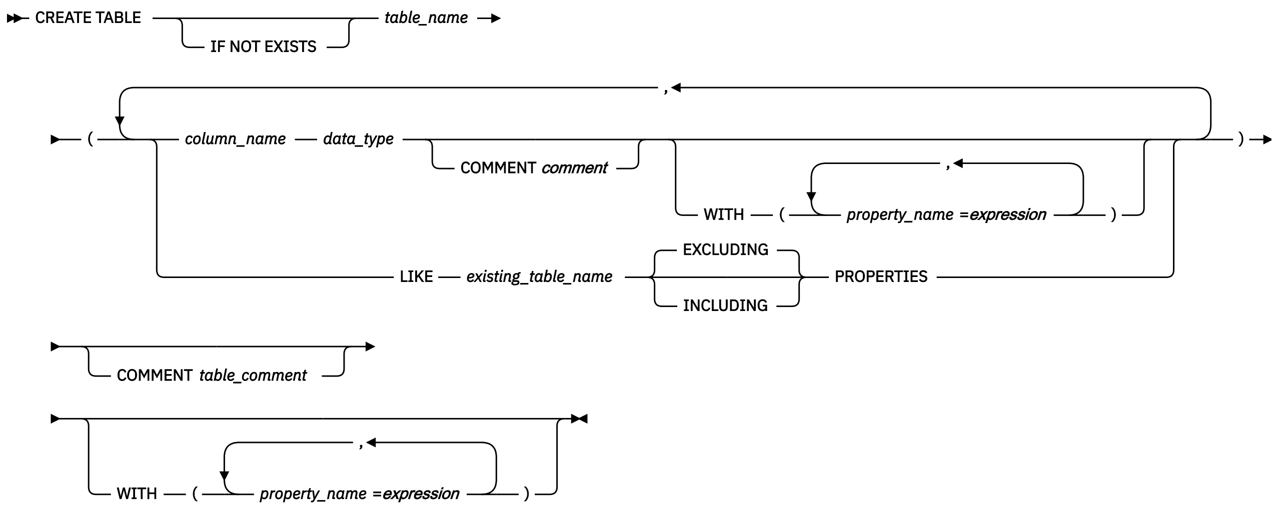
Invocation

This statement can be embedded in an application program or issued by using dynamic SQL statements.

Syntax

```
CREATE TABLE [ IF NOT EXISTS ]
table_name (
  { column_name data_type [ COMMENT comment ] [ WITH ( property_name = expression [, ...] ) ]
  | LIKE existing_table_name [ { INCLUDING | EXCLUDING } PROPERTIES ] }
  [, ...]
)
[ COMMENT table_comment ]
[ WITH ( property_name = expression [, ...] ) ]
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

IF NOT EXISTS
Specifies that an error message is suppressed when the table cannot be created because a table with the specified name exists in the current database and schema.

table_name
Names the table. The name, including the implicit or explicit qualifier, must not identify a table, view, nickname, or alias described in the catalog.

Note: The following conditions apply for *table_name*:

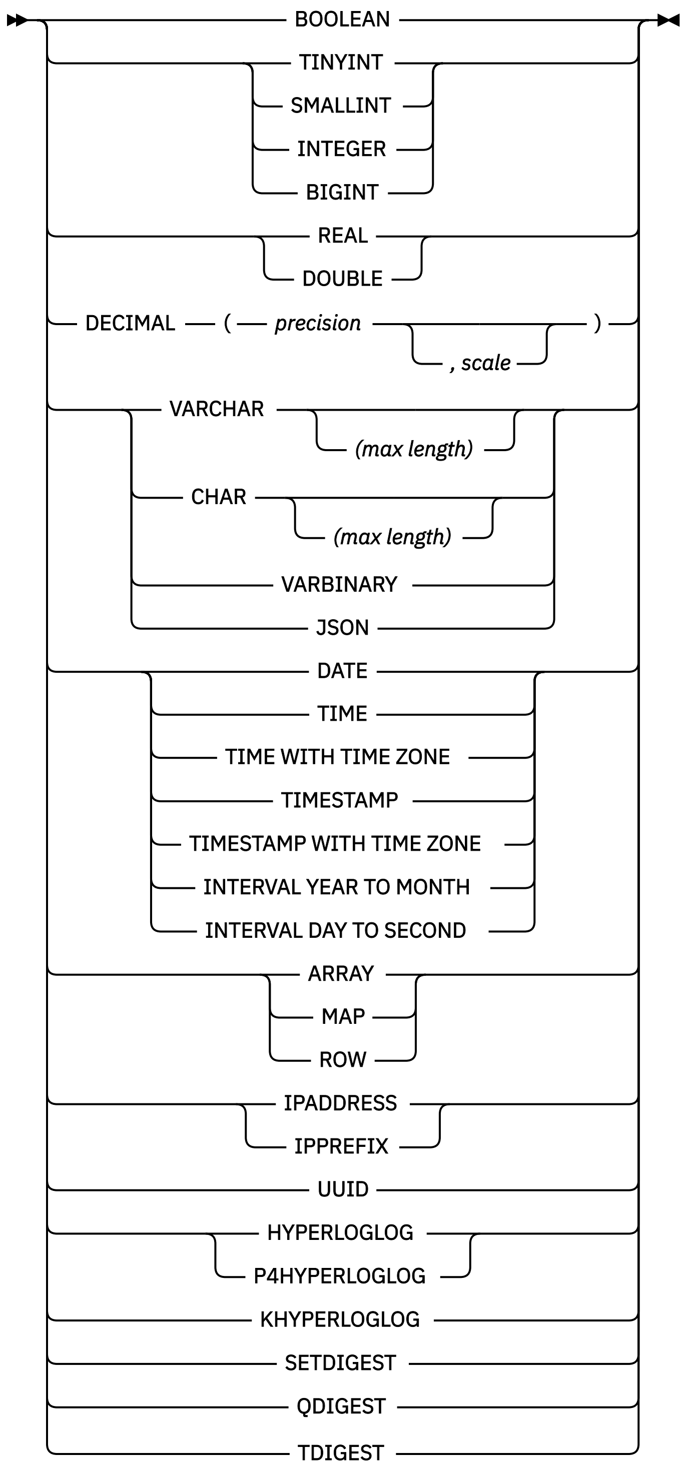
- *table_name* with square brackets '[]' must contain some character inside it. For example: 'p![ab]&'.
- *table_name* with open square bracket '[' must also contain a closed square bracket ']'.
- *table_name* with only closed square bracket ']' is valid.

column-definition
Defines the attributes of a column.

column_name
Names a column of the table. The name cannot be qualified, and the same name cannot be used for more than one column of the table.

data_type
Specifies the data type of the column.

DATA TYPE



LIKE *existing_table_name [{ INCLUDING | EXCLUDING } PROPERTIES]*

Specifies that the columns of the table have the same name and description as the columns of the specified table (*existing_table_name*). The specified table must either exist in the catalog or must be a declared temporary table.

Multiple LIKE clauses can be specified, which allows copying the columns from multiple tables.

If INCLUDING PROPERTIES is specified, all the table properties are copied to the new table. The default behavior is EXCLUDING PROPERTIES. The INCLUDING PROPERTIES option must be specified for at most one table.

WITH *property_name = expression [, ...]*

Sets properties on the newly created table or on single columns.

To list all the available table properties, run the following query:

```
SELECT * FROM system.metadata.table_properties
```

To list all available column properties, run the following query:

```
SELECT * FROM system.metadata.column_properties
```

Note: Currently, no column properties are supported.

Note: If the WITH clause specifies the same property name as one of the copied properties, the value from the WITH clause is used.

Examples

Create the table ORDERS.

```
CREATE TABLE ORDERS (
  ORDERKEY bigint,
  STATUS varchar,
  PRICE double,
  DATE date
)
WITH (format = 'ORC')
```

Create the table ORDERS if it does not exist, adding a table comment and a column comment.

```
CREATE TABLE IF NOT EXISTS ORDERS (
  ORDERKEY bigint,
  STATUS varchar,
  PRICE double COMMENT 'Price in cents.',
  DATE date
)
COMMENT 'A table to keep track of orders.'
```

Create the table EXPAND_ORDERS by using the columns from ORDERS with more columns at the start and end.

```
CREATE TABLE EXPAND_ORDERS (
  NEW_ORDERKEY bigint,
  LIKE orders,
  NEW_DATE date
)
```

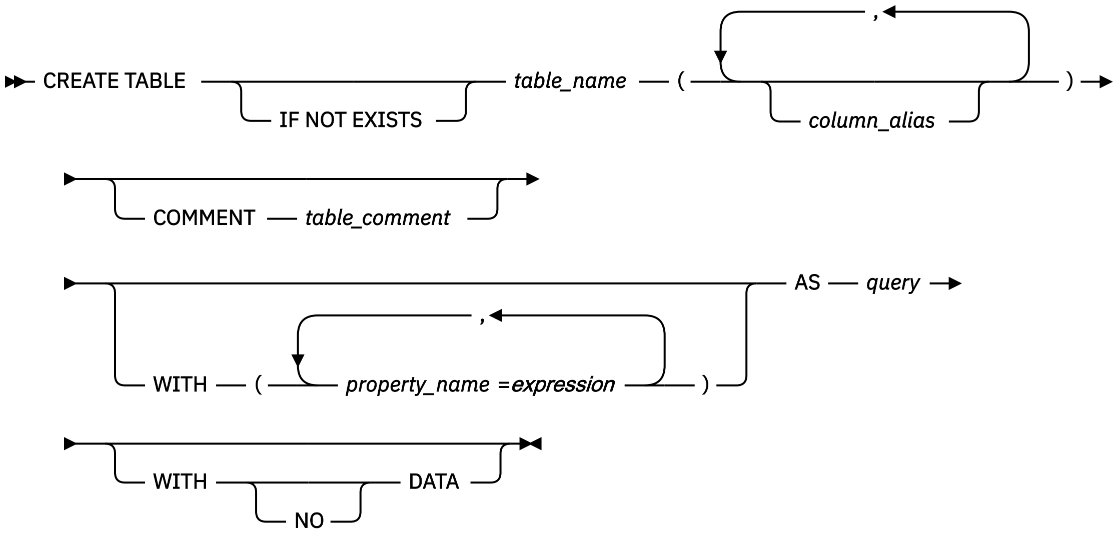
CREATE TABLE AS statement

Synopsis

```
CREATE TABLE [ IF NOT EXISTS ] table_name [ ( column_alias, ... ) ]
[ COMMENT table_comment ]
[ WITH ( property_name = expression [, ...] ) ]
```

```
AS query
[ WITH [ NO ] DATA ]
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Create a new table containing the result of a SELECT query. Use CREATE TABLE statement to create an empty table.

The optional `IF NOT EXISTS` clause causes the error to be suppressed if the table already exists.

The optional `WITH` clause can be used to set properties on the newly created table. To list all available table properties, run the following query:

```
SELECT * FROM system.metadata.table_properties
```

Examples

Create a new table `orders_column_aliased` with the results of a query and the given column names:

```
CREATE TABLE orders_column_aliased (order_date, total_price)
AS
SELECT orderdate, totalprice
FROM orders
```

Create a new table `orders_by_date` that summarizes `orders`:

```
CREATE TABLE orders_by_date
COMMENT 'Summary of orders by date'
WITH (format = 'ORC')
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
```

Create the table `orders_by_date` if it does not already exist:

```
CREATE TABLE IF NOT EXISTS orders_by_date AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
```

Create a new `empty_nation` table with the same schema as `nation` and no data:

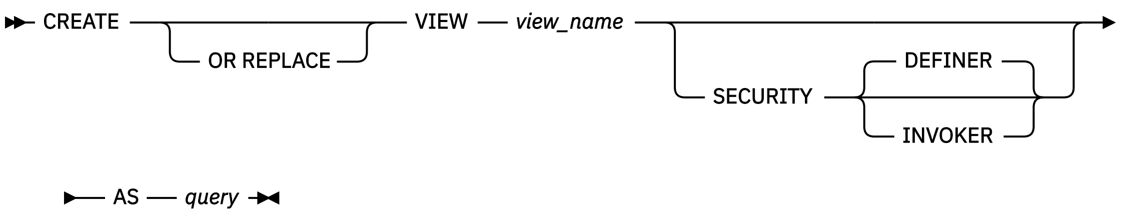
```
CREATE TABLE empty_nation AS
SELECT *
FROM nation
WITH NO DATA
```

CREATE VIEW statement

Synopsis

```
CREATE [ OR REPLACE ] VIEW view_name
[ SECURITY { DEFINER | INVOKER } ]
AS query
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Create a new view of a SELECT query. The view is a logical table that can be referenced by future queries. Views do not contain any data. Instead, the query stored by the view is executed every time the view is referenced by another query.

The optional `OR REPLACE` clause causes the view to be replaced if it already exists rather than raising an error.

Security

In the default `DEFINER` security mode, tables referenced in the view are accessed using the permissions of the view owner (the *creator* or *definer* of the view) rather than the user executing the query. This allows providing restricted access to the underlying tables, for which the query user may not be allowed to access directly. Note that the `current_user`function will return the query user, not the view owner, and thus may be used to filter out rows or otherwise restrict access based on the user currently accessing the view.

In the `INVOKER` security mode, tables referenced in the view are accessed using the permissions of the query user (the *invoker* of the view). A view created in this mode is simply a stored query.

Examples

Create a simple view `test` over the `orders` table:

```
CREATE VIEW test AS
SELECT orderkey, orderstatus, totalprice / 2 AS half
FROM orders
```

Create a view `orders_by_date` that summarizes `orders`:

```
CREATE VIEW orders_by_date AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
```

Create a view that replaces an existing view:

```
CREATE OR REPLACE VIEW test AS
SELECT orderkey, orderstatus, totalprice / 4 AS quarter
FROM orders
```

DEALLOCATE PREPARE statement

Synopsis

```
DEALLOCATE PREPARE statement_name
```

Syntax Diagram

▶▶─ DEALLOCATE ── PREPARE ── *statement_name* ─▶◀

For information on reading syntax diagrams, see How to read syntax diagrams

Description

- Removes a statement with the name `statement_name` from the list of prepared statements in a session.

Examples

Deallocate a statement with the name `my_query`:

```
DEALLOCATE PREPARE my_query;
```

DESCRIBE statement

Synopsis

```
DESCRIBE table_name
```

Syntax Diagram

▶▶─ DESCRIBE ── *table_name* ─▶◀

For information on reading syntax diagrams, see How to read syntax diagrams

Description

DESCRIBE is an alias for SHOW COLUMNS statement.

DESCRIBE INPUT statement

Synopsis

```
DESCRIBE INPUT statement_name
```

Syntax Diagram

▶▶─ DESCRIBE INPUT ── *statement_name* ─▶◀

For information on reading syntax diagrams, see How to read syntax diagrams

Description

Lists the input parameters of a prepared statement along with the position and type of each parameter. Parameter types that cannot be determined will appear as unknown.

Examples

Prepare and describe a query with three parameters:

```
PREPARE my_select1 FROM
SELECT ? FROM nation WHERE regionkey = ? AND name < ?;
```

```
DESCRIBE INPUT my_select1;
```

```
 Position | Type
--------------------
        0 | unknown
        1 | bigint
        2 | varchar
(3 rows)
```

Prepare and describe a query with no parameters:

```
PREPARE my_select2 FROM
SELECT * FROM nation;
```

```
DESCRIBE INPUT my_select2;
```

```
 Position | Type
-----------------
(0 rows)
```

DESCRIBE OUTPUT statement

Synopsis

```
DESCRIBE OUTPUT statement_name
```

Syntax Diagram

▶▶— DESCRIBE OUTPUT — *statement_name* —▶◀

For information on reading syntax diagrams, see How to read syntax diagrams

Description

List the output columns of a prepared statement, including the column name (or alias), catalog, schema, table, type, type size in bytes, and a boolean indicating if the column is aliased.

Examples

Prepare and describe a query with four output columns:

```
PREPARE my_select1 FROM
SELECT * FROM nation
```

```
DESCRIBE OUTPUT my_select1;
```

```
  Column Name | Catalog | Schema | Table  |  Type   | Type Size | Aliased
-------------+---------+--------+--------+---------+-----------+---------
 nationkey   | tpch    | sf1    | nation | bigint  |         8 | false
 name        | tpch    | sf1    | nation | varchar |         0 | false
 regionkey   | tpch    | sf1    | nation | bigint  |         8 | false
 comment     | tpch    | sf1    | nation | varchar |         0 | false
(4 rows)
```

Prepare and describe a query whose output columns are expressions:

```
PREPARE my_select2 FROM
SELECT count(*) as my_count, 1+2 FROM nation
```

```
DESCRIBE OUTPUT my_select2;
```

```
  Column Name | Catalog | Schema | Table |  Type  | Type Size | Aliased
-------------+---------+--------+-------+--------+-----------+---------
 my_count    |         |        |       | bigint |         8 | true
 _col1       |         |        |       | bigint |         8 | false
(2 rows)
```

Prepare and describe a row count query:

```
PREPARE my_create FROM
CREATE TABLE foo AS SELECT * FROM nation
```

```
DESCRIBE OUTPUT my_create;
```

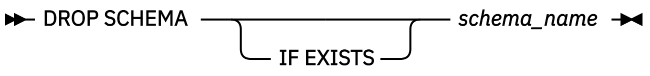
```
   Column Name | Catalog | Schema | Table |  Type  | Type Size | Aliased
-------------+---------+--------+-------+--------+-----------+---------
 rows        |         |        |       | bigint |         8 | false
(1 row)
```

DROP SCHEMA statement

Synopsis

```
DROP SCHEMA [ IF EXISTS ] schema_name
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Drop an existing schema. The schema must be empty.

The optional IF EXISTS clause causes the error to be suppressed if the schema does not exist.

Examples

Drop the schema web:

```
DROP SCHEMA web
```

Drop the schema `sales` if it exists:

```
DROP SCHEMA IF EXISTS sales
```

Expected behavior

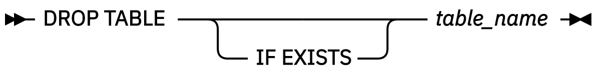
If you run a `DROP SCHEMA` query, it deletes the metadata only and not the actual directory. For all external schemas, you must explicitly delete the directory or data in the directory.

DROP TABLE statement

Synopsis

```
DROP TABLE  [ IF EXISTS ] table_name
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Drops an existing table.

The optional `IF EXISTS` clause causes the error to be suppressed if the table does not exist.

Examples

Drop the table `orders_by_date`:

```
DROP TABLE orders_by_date
```

Drop the table `orders_by_date` if it exists:

```
DROP TABLE IF EXISTS orders_by_date
```

Expected behavior

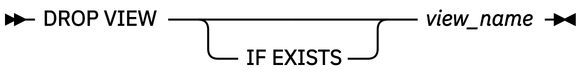
- If you run a `DROP TABLE` query on external tables, only the metadata is deleted and the underlying data remains as is. For all external tables, you must explicitly delete the directory and the data.
- If you run a `DROP TABLE` query on managed tables, the metadata and the underlying data are deleted.

DROP VIEW statement

Synopsis

```
DROP VIEW [ IF EXISTS ] view_name
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Drop an existing view.

The optional `IF EXISTS` clause causes the error to be suppressed if the view does not exist.

Examples

Drop the view `orders_by_date`:

```
DROP VIEW orders_by_date
```

Drop the view `orders_by_date` if it exists:

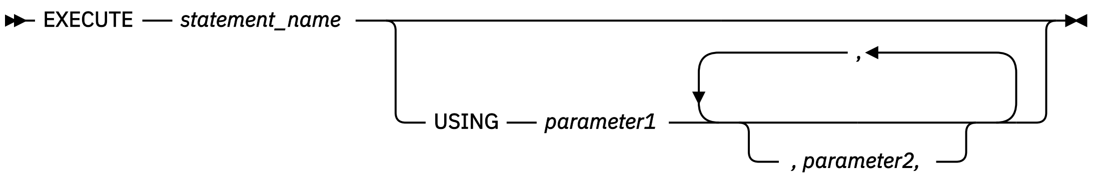
```
DROP VIEW IF EXISTS orders_by_date
```

EXECUTE statement

Synopsis

```
EXECUTE statement_name [ USING parameter1 [ , parameter2, ... ] ]
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Executes a prepared statement with the name `statement_name`. Parameter values are defined in the `USING` clause.

Examples

Prepare and execute a query with no parameters:

```
PREPARE my_select1 FROM
SELECT name FROM nation;
```

```
EXECUTE my_select1;
```

Prepare and execute a query with two parameters:

```
PREPARE my_select2 FROM
SELECT name FROM nation WHERE regionkey = ? and nationkey < ?;
```

```
EXECUTE my_select2 USING 1, 3;
```

This is equivalent to:

```
SELECT name FROM nation WHERE regionkey = 1 AND nationkey < 3;
```

EXPLAIN statement

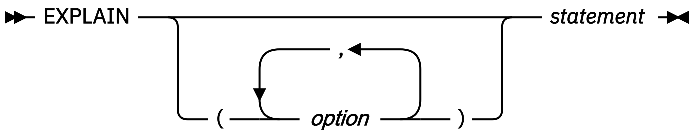
Synopsis

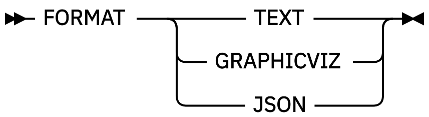
```
EXPLAIN [ ( option [, ...] ) ] statement
```

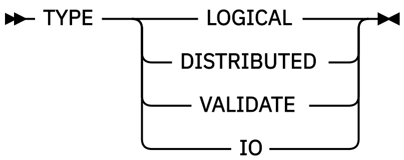
where `option` can be one of:

```
FORMAT { TEXT | GRAPHVIZ | JSON }
TYPE { LOGICAL | DISTRIBUTED | VALIDATE | IO }
```

Syntax Diagram







For information on reading syntax diagrams, see How to read syntax diagrams

Description

Show the logical or distributed execution plan of a statement, or validate the statement. Use TYPE `DISTRIBUTED` option to display fragmented plan. Each plan fragment is executed by a single or multiple Presto nodes. Fragments separation represent the data exchange between Presto nodes. Fragment type specifies how the fragment is executed by Presto nodes and how the data is distributed between fragments:

SINGLE

Fragment is executed on a single node.

HASH

Fragment is executed on a fixed number of nodes with the input data distributed using a hash function.

ROUND_ROBIN

Fragment is executed on a fixed number of nodes with the input data distributed in a round-robin fashion.

BROADCAST

Fragment is executed on a fixed number of nodes with the input data broadcasted to all nodes.

SOURCE

Fragment is executed on nodes where input splits are accessed.

Examples

Logical plan:

```
presto:tiny> EXPLAIN SELECT regionkey, count(*) FROM nation GROUP BY 1;
                                                Query Plan
-----------------------------------------------------------------------------------------------
----------
 - Output[regionkey, _col1] => [regionkey:bigint, count:bigint]
         _col1 := count
     - RemoteExchange[GATHER] => regionkey:bigint, count:bigint
```

```
         - Aggregate(FINAL)[regionkey] => [regionkey:bigint, count:bigint]
                count := "count"("count_8")
             - LocalExchange[HASH][$hashvalue] ("regionkey") => regionkey:bigint,
count_8:bigint, $hashvalue:bigint
                 - RemoteExchange[REPARTITION][$hashvalue_9] => regionkey:bigint,
count_8:bigint, $hashvalue_9:bigint
                     - Project[] => [regionkey:bigint, count_8:bigint, $hashvalue_10:bigint]
                             $hashvalue_10 := "combine_hash"(BIGINT '0',
COALESCE("$operator$hash_code"("regionkey"), 0))
                         - Aggregate(PARTIAL)[regionkey] => [regionkey:bigint, count_8:bigint]
                                 count_8 := "count"(*)
                             - TableScan[tpch:tpch:nation:sf0.1, originalConstraint = true] =>
[regionkey:bigint]
                                     regionkey := tpch:regionkey
```

Distributed plan:

```
presto:tiny> EXPLAIN (TYPE DISTRIBUTED) SELECT regionkey, count(*) FROM nation GROUP BY 1;
                                          Query Plan
----------------------------------------------------------------------------------------------
 Fragment 0 [SINGLE]
     Output layout: [regionkey, count]
     Output partitioning: SINGLE []
     - Output[regionkey, _col1] => [regionkey:bigint, count:bigint]
             _col1 := count
         - RemoteSource[1] => [regionkey:bigint, count:bigint]

 Fragment 1 [HASH]
     Output layout: [regionkey, count]
     Output partitioning: SINGLE []
     - Aggregate(FINAL)[regionkey] => [regionkey:bigint, count:bigint]
             count := "count"("count_8")
         - LocalExchange[HASH][$hashvalue] ("regionkey") => regionkey:bigint, count_8:bigint,
$hashvalue:bigint
             - RemoteSource[2] => [regionkey:bigint, count_8:bigint, $hashvalue_9:bigint]

 Fragment 2 [SOURCE]
     Output layout: [regionkey, count_8, $hashvalue_10]
     Output partitioning: HASH [regionkey][$hashvalue_10]
     - Project[] => [regionkey:bigint, count_8:bigint, $hashvalue_10:bigint]
             $hashvalue_10 := "combine_hash"(BIGINT '0',
COALESCE("$operator$hash_code"("regionkey"), 0))
         - Aggregate(PARTIAL)[regionkey] => [regionkey:bigint, count_8:bigint]
                 count_8 := "count"(*)
             - TableScan[tpch:tpch:nation:sf0.1, originalConstraint = true] =>
[regionkey:bigint]
                     regionkey := tpch:regionkey
```

Validate:

```
presto:tiny> EXPLAIN (TYPE VALIDATE) SELECT regionkey, count(*) FROM nation GROUP BY 1;
 Valid
-------
 true
```

IO:

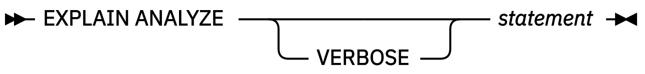
```
presto:hive> EXPLAIN (TYPE IO, FORMAT JSON) INSERT INTO test_nation SELECT * FROM nation WHERE
regionkey = 2;
            Query Plan
-----------------------------------
 {
   "inputTableColumnInfos" : [ {
     "table" : {
       "catalog" : "hive",
       "schemaTable" : {
         "schema" : "tpch",
         "table" : "nation"
       }
     },
     "columns" : [ {
       "columnName" : "regionkey",
       "type" : "bigint",
       "domain" : {
         "nullsAllowed" : false,
         "ranges" : [ {
           "low" : {
             "value" : "2",
             "bound" : "EXACTLY"
           },
           "high" : {
             "value" : "2",
             "bound" : "EXACTLY"
           }
         } ]
       }
     } ]
   } ],
   "outputTable" : {
     "catalog" : "hive",
     "schemaTable" : {
       "schema" : "tpch",
       "table" : "test_nation"
     }
   }
 }
```

EXPLAIN ANALYZE statement

Synopsis

```
EXPLAIN ANALYZE [VERBOSE] statement
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Execute the statement and show the distributed execution plan of the statement along with the cost of each operation.

The VERBOSE option will give more detailed information and low-level statistics; understanding these may require knowledge of Presto (Java) and Presto (C++) internals and implementation details.

Note: The stats may not be entirely accurate, especially for queries that complete quickly.

Examples

In the example below, you can see the CPU time spent in each stage, as well as the relative cost of each plan node in the stage. Note that the relative cost of the plan nodes is based on wall time, which may or may not be correlated to CPU time. For each plan node you can see some additional statistics (e.g: average input per node instance, average number of hash collisions for relevant plan nodes). Such statistics are useful when one wants to detect data anomalies for a query (skewness, abnormal hash collisions).

```
presto:sf1> EXPLAIN ANALYZE SELECT count(*), clerk FROM orders WHERE orderdate > date
'1995-01-01' GROUP BY clerk;

                                          Query Plan
-----------------------------------------------------------------------------------------------
Fragment 1 [HASH]
    Cost: CPU 88.57ms, Input: 4000 rows (148.44kB), Output: 1000 rows (28.32kB)
    Output layout: [count, clerk]
    Output partitioning: SINGLE []
    - Project[] => [count:bigint, clerk:varchar(15)]
            Cost: 26.24%, Input: 1000 rows (37.11kB), Output: 1000 rows (28.32kB), Filtered:
0.00%
            Input avg.: 62.50 lines, Input std.dev.: 14.77%
        - Aggregate(FINAL)[clerk][$hashvalue] => [clerk:varchar(15), $hashvalue:bigint,
count:bigint]
                Cost: 16.83%, Output: 1000 rows (37.11kB)
                Input avg.: 250.00 lines, Input std.dev.: 14.77%
                count := "count"("count_8")
            - LocalExchange[HASH][$hashvalue] ("clerk") => clerk:varchar(15), count_8:bigint,
$hashvalue:bigint
                    Cost: 47.28%, Output: 4000 rows (148.44kB)
                    Input avg.: 4000.00 lines, Input std.dev.: 0.00%
                - RemoteSource[2] => [clerk:varchar(15), count_8:bigint, $hashvalue_9:bigint]
                        Cost: 9.65%, Output: 4000 rows (148.44kB)
                        Input avg.: 4000.00 lines, Input std.dev.: 0.00%

Fragment 2 [tpch:orders:1500000]
    Cost: CPU 14.00s, Input: 818058 rows (22.62MB), Output: 4000 rows (148.44kB)
    Output layout: [clerk, count_8, $hashvalue_10]
    Output partitioning: HASH [clerk][$hashvalue_10]
    - Aggregate(PARTIAL)[clerk][$hashvalue_10] => [clerk:varchar(15), $hashvalue_10:bigint,
count_8:bigint]
            Cost: 4.47%, Output: 4000 rows (148.44kB)
            Input avg.: 204514.50 lines, Input std.dev.: 0.05%
            Collisions avg.: 5701.28 (17569.93% est.), Collisions std.dev.: 1.12%
            count_8 := "count"(*)
        - ScanFilterProject[table = tpch:tpch:orders:sf1.0, originalConstraint = ("orderdate"
> "$literal$date"(BIGINT '9131')), filterPredicate = ("orderdate" > "$literal$date"(BIGINT
'9131'))] => [cler
                Cost: 95.53%, Input: 1500000 rows (0B), Output: 818058 rows (22.62MB),
Filtered: 45.46%
                Input avg.: 375000.00 lines, Input std.dev.: 0.00%
                $hashvalue_10 := "combine_hash"(BIGINT '0',
COALESCE("$operator$hash_code"("clerk"), 0))
                orderdate := tpch:orderdate
                clerk := tpch:clerk
```

When the VERBOSE option is used, some operators may report additional information. For example, the window function operator will output the following:

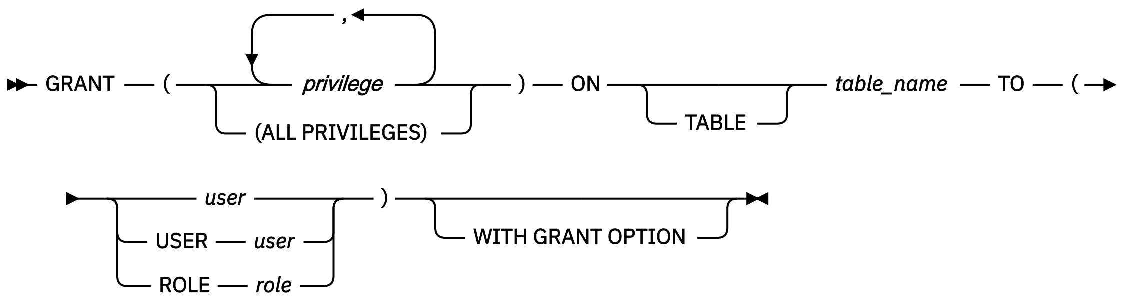
```
EXPLAIN ANALYZE VERBOSE SELECT count(clerk) OVER() FROM orders WHERE orderdate > date
'1995-01-01';

                                          Query Plan
-----------------------------------------------------------------------------------------------
  ...
       - Window[] => [clerk:varchar(15), count:bigint]
               Cost: {rows: ?, bytes: ?}
               CPU fraction: 75.93%, Output: 8130 rows (230.24kB)
               Input avg.: 8130.00 lines, Input std.dev.: 0.00%
               Active Drivers: [ 1 / 1 ]
               Index size: std.dev.: 0.00 bytes , 0.00 rows
               Index count per driver: std.dev.: 0.00
               Rows per driver: std.dev.: 0.00
               Size of partition: std.dev.: 0.00
               count := count("clerk")
 ...
```

GRANT statement

Synopsis

```
GRANT ( privilege [, ...] | ( ALL PRIVILEGES ) )
ON [ TABLE ] table_name TO ( user | USER user | ROLE role )
[ WITH GRANT OPTION ]
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Grants the specified privileges to the specified grantee.

Specifying ALL PRIVILEGES grants DELETE, INSERT, and SELECT privileges.

Specifying ROLE PUBLIC grants privileges to the PUBLIC role and hence to all users.

The optional WITH GRANT OPTION clause allows the grantee to grant these same privileges to others.

For GRANT statement to succeed, the user executing it should possess the specified privileges as well as the GRANT OPTION for those privileges.

Examples

Grant INSERT and SELECT privileges on the table orders to user alice:

```
GRANT INSERT, SELECT ON orders TO alice;
```

Grant SELECT privilege on the table nation to user alice, additionally allowing alice to grant SELECT privilege to others:

```
GRANT SELECT ON nation TO alice WITH GRANT OPTION;
```

Grant SELECT privilege on the table orders to everyone:

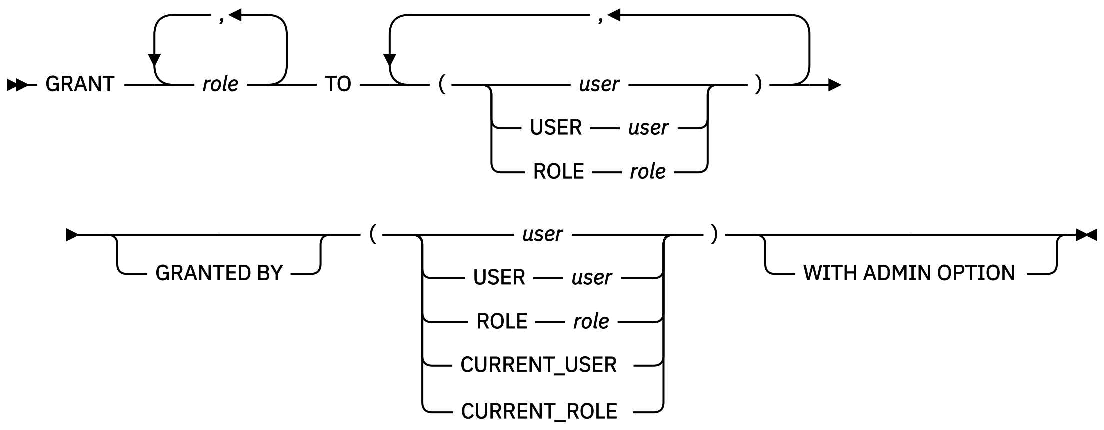
```
GRANT SELECT ON orders TO ROLE PUBLIC;
```

GRANT ROLES statement

Synopsis

```
GRANT role [, ...]
TO ( user | USER user | ROLE role) [, ...]
[ GRANTED BY ( user | USER user | ROLE role | CURRENT_USER | CURRENT_ROLE ) ]
[ WITH ADMIN OPTION ]
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Grants the specified role(s) to the specified principal(s) in the current catalog.

If the WITH ADMIN OPTION clause is specified, the role(s) are granted to the users with GRANToption.

For the GRANT statement for roles to succeed, the user executing it either should be the role admin or should possess the GRANT option for the given role.

The optional GRANTED BY clause causes the role(s) to be granted with the specified principal as a grantor. If the GRANTED BY clause is not specified, the roles are granted with the current user as a grantor.

Examples

Grant role bar to user foo

```
GRANT bar TO USER foo;
```

Grant roles bar and foo to user baz and role qux with admin option

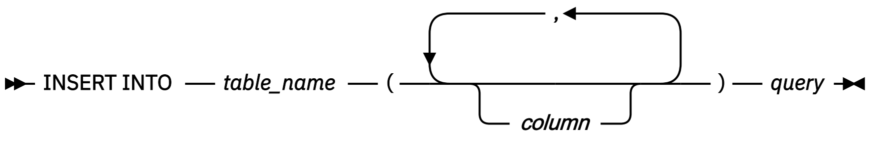
```
GRANT bar, foo TO USER baz, ROLE qux WITH ADMIN OPTION;
```

INSERT statement

Synopsis

```
INSERT INTO table_name [ ( column [, ... ] ) ] query
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Insert new rows into a table.

If the list of column names is specified, they must exactly match the list of columns produced by the query. Each column in the table not present in the column list will be filled with a nullvalue. Otherwise, if the list of columns is not specified, the columns produced by the query must exactly match the columns in the table being inserted into.

Examples

Load additional rows into the orders table from the new_orders table:

```
INSERT INTO orders
SELECT * FROM new_orders;
```

Insert a single row into the cities table:

```
INSERT INTO cities VALUES (1, 'San Francisco');
```

Insert multiple rows into the cities table:

```
INSERT INTO cities VALUES (2, 'San Jose'), (3, 'Oakland');
```

Insert a single row into the `nation` table with the specified column list:

```
INSERT INTO nation (nationkey, name, regionkey, comment)
VALUES (26, 'POLAND', 3, 'no comment');
```

Insert a row without specifying the `comment` column. That column will be `null`:

```
INSERT INTO nation (nationkey, name, regionkey)
VALUES (26, 'POLAND', 3);
```

PREPARE statement

Synopsis

```
PREPARE statement_name FROM statement
```

Syntax Diagram

▶▶─ PREPARE ── *statement_name* ── FROM ── *statement* ─▶◀

For information on reading syntax diagrams, see How to read syntax diagrams

Description

Prepares a statement for execution at a later time. Prepared statements are queries that are saved in a session with a given name. The statement can include parameters in place of literals to be replaced at execution time. Parameters are represented by question marks.

Examples

Prepare a select query:

```
PREPARE my_select1 FROM
SELECT * FROM nation;
```

Prepare a select query that includes parameters. The values to compare with `regionkey` and `nationkey` will be filled in with the EXECUTE statement:

```
PREPARE my_select2 FROM
SELECT name FROM nation WHERE regionkey = ? AND nationkey < ?;
```

Prepare an insert query:

```
PREPARE my_insert FROM
INSERT INTO cities VALUES (1, 'San Francisco');
```

RESET SESSION statement

Synopsis

```
RESET SESSION name
RESET SESSION catalog.name
```

Syntax Diagram

▶▶─ RESET SESSION ── *name* ─▶◀

▶▶─ RESET SESSION ── *catalog.name* ─▶◀

For information on reading syntax diagrams, see How to read syntax diagrams

Description

Reset a session property value to the default value.

Examples

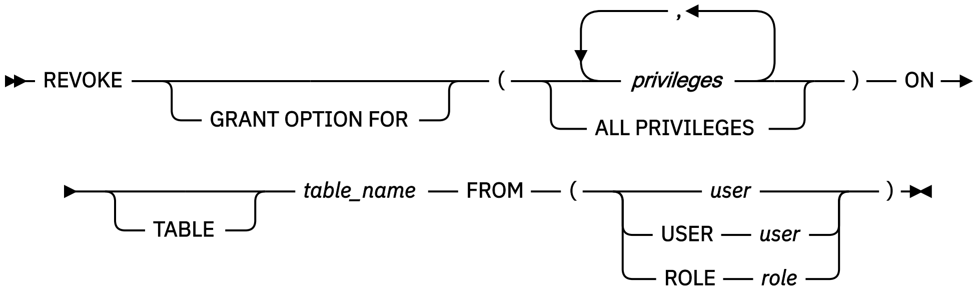
```
RESET SESSION optimize_hash_generation;
RESET SESSION hive.optimized_reader_enabled;
```

REVOKE statement

Synopsis

```
REVOKE [ GRANT OPTION FOR ]
( privilege [, ...] | ALL PRIVILEGES )
ON [ TABLE ] table_name FROM ( user | USER user | ROLE role )
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Revokes the specified privileges from the specified grantee.

Specifying `ALL PRIVILEGES` revokes DELETE, INSERT, and SELECT privileges.

Specifying `ROLE PUBLIC` revokes privileges from the `PUBLIC` role. Users will retain privileges assigned to them directly or via other roles.

The optional `GRANT OPTION FOR` clause also revokes the privileges to grant the specified privileges.

For `REVOKE` statement to succeed, the user executing it should possess the specified privileges as well as the `GRANT OPTION` for those privileges.

Examples

Revoke `INSERT` and `SELECT` privileges on the table `orders` from user `alice`:

```
REVOKE INSERT, SELECT ON orders FROM alice;
```

Revoke `SELECT` privilege on the table `nation` from everyone, additionally revoking the privilege to grant `SELECT` privilege:

```
REVOKE GRANT OPTION FOR SELECT ON nation FROM ROLE PUBLIC;
```

Revoke all privileges on the table `test` from user `alice`:

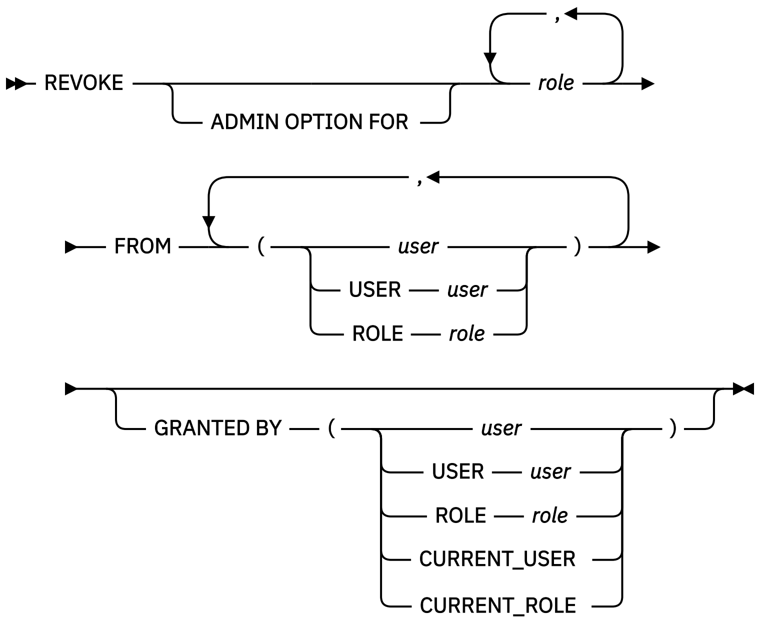
```
REVOKE ALL PRIVILEGES ON test FROM alice;
```

REVOKE ROLES statement

Synopsis

```
REVOKE
[ ADMIN OPTION FOR ]
role [, ...]
FROM ( user | USER user | ROLE role) [, ...]
[ GRANTED BY ( user | USER user | ROLE role | CURRENT_USER | CURRENT_ROLE ) ]
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Revokes the specified role(s) from the specified principal(s) in the current catalog.

If the `ADMIN OPTION FOR` clause is specified, the `GRANT` permission is revoked instead of the role.

For the `REVOKE` statement for roles to succeed, the user executing it either should be the role admin or should possess the `GRANT` option for the given role.

The optional `GRANTED BY` clause causes the role(s) to be revoked with the specified principal as a revoker. If the `GRANTED BY` clause is not specified, the roles are revoked by the current user as a revoker.

Examples

Revoke role `bar` from user `foo`

```
REVOKE bar FROM USER foo;
```

Revoke admin option for roles `bar` and `foo` from user `baz` and role `qux`

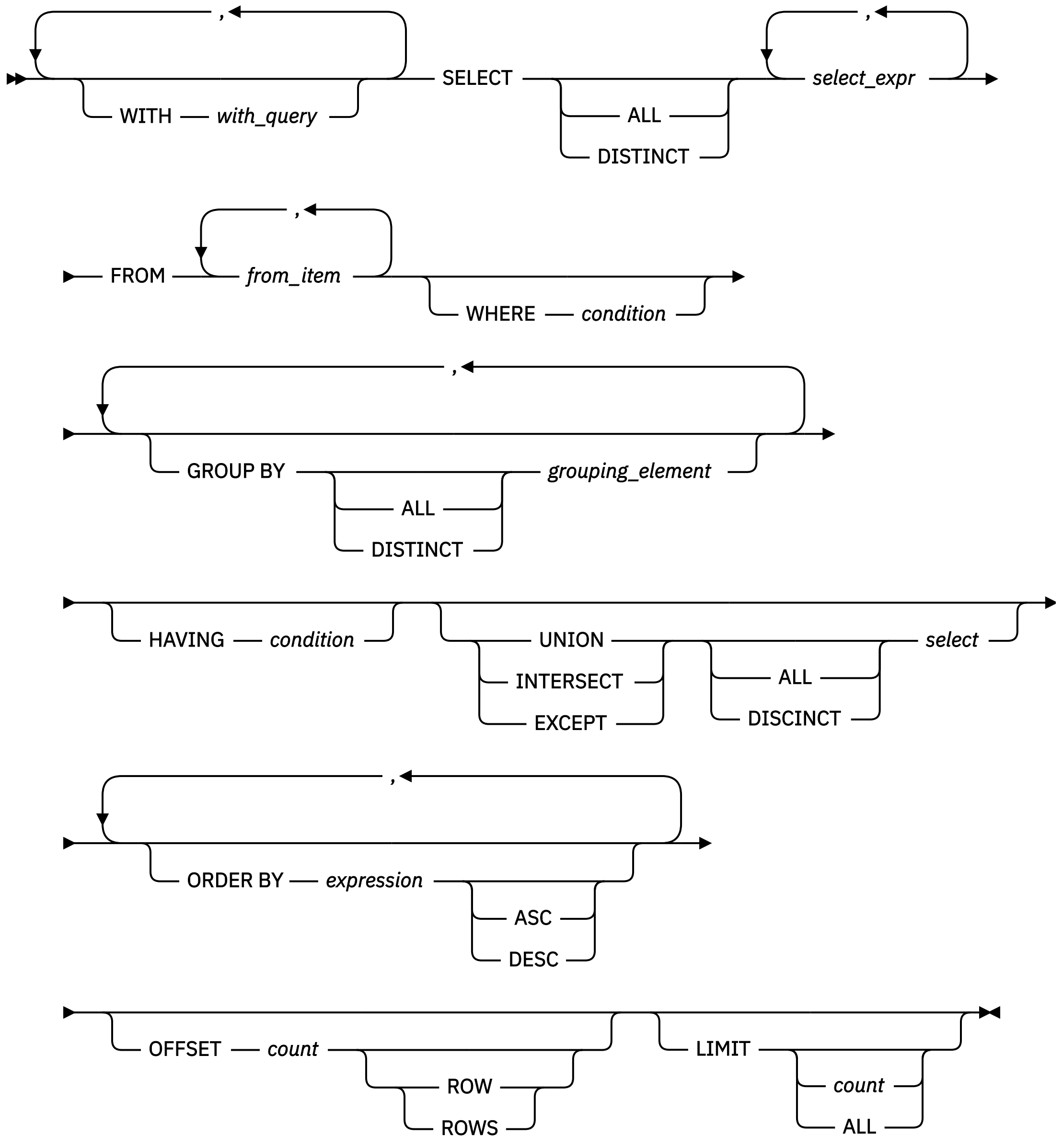
```
REVOKE ADMIN OPTION FOR bar, foo FROM USER baz, ROLE qux;
```

SELECT statement

Synopsis

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] select_expr [, ...]
FROM from_item [, ...]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] [, ...] ]
[ OFFSET count [ { ROW | ROWS } ] ]
[ { LIMIT [ count | ALL ] } ]
```

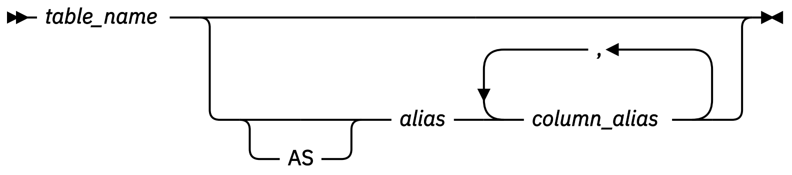
Syntax Diagram



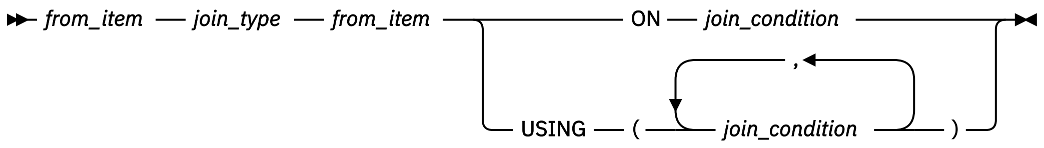
For information on reading syntax diagrams, see How to read syntax diagrams

where `from_item` is one of

```
table_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
```

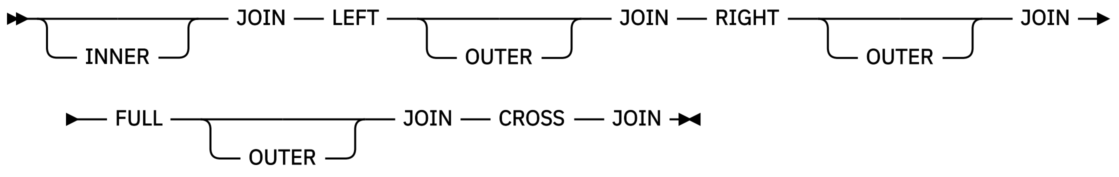


```
from_item join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]
```



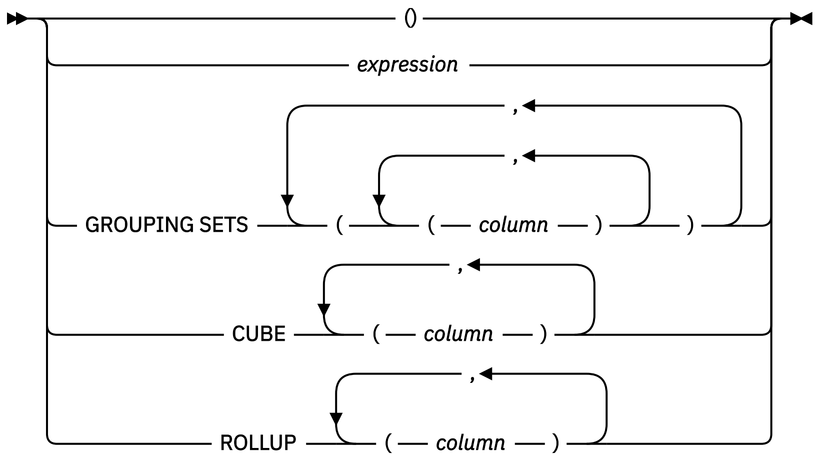
and `join_type` is one of

```
[ INNER ] JOIN
LEFT [ OUTER ] JOIN
RIGHT [ OUTER ] JOIN
FULL [ OUTER ] JOIN
CROSS JOIN
```



and `grouping_element` is one of

```
()
expression
GROUPING SETS ( ( column [, ...] ) [, ...] )
CUBE ( column [, ...] )
ROLLUP ( column [, ...] )
```



Description

Retrieve rows from zero or more tables.

WITH Clause

The WITH clause defines named relations for use within a query. It allows flattening nested queries or simplifying subqueries. For example, the following queries are equivalent:

```
SELECT a, b
FROM (
  SELECT a, MAX(b) AS b FROM t GROUP BY a
) AS x;

WITH x AS (SELECT a, MAX(b) AS b FROM t GROUP BY a)
SELECT a, b FROM x;
```

This example also works with multiple subqueries:

```
WITH
  t1 AS (SELECT a, MAX(b) AS b FROM x GROUP BY a),
  t2 AS (SELECT a, AVG(d) AS d FROM y GROUP BY a)
SELECT t1.*, t2.*
FROM t1
JOIN t2 ON t1.a = t2.a;
```

Additionally, the relations within a WITH clause can chain:

```
WITH
  x AS (SELECT a FROM t),
  y AS (SELECT a AS b FROM x),
  z AS (SELECT b AS c FROM y)
SELECT c FROM z;
```

Warning: Currently, the SQL for the WITH clause is inlined anywhere the named relation is used. This means that if the relation is used more than once and the query is nondeterministic, the results may be different each time.

GROUP BY Clause

The GROUP BY clause divides the output of a SELECT statement into groups of rows that contain matching values. A simple GROUP BY clause may contain any expression that is composed of input columns or it may be an ordinal number that selects an output column by position (starting at one).

The following queries are equivalent. They both group the output by the nationkey input column with the first query by using the ordinal position of the output column and the second query by using the input column name:

```
SELECT count(*), nationkey FROM customer GROUP BY 2;

SELECT count(*), nationkey FROM customer GROUP BY nationkey;
```

GROUP BY clauses can group output by input column names not appearing in the output of a select statement. For example, the following query generates row counts for the customer table by using the input column mktsegment:

```
SELECT count(*) FROM customer GROUP BY mktsegment;
```

```
 _col0
-------
 29968
 30142
 30189
 29949
 29752
(5 rows)
```

When a GROUP BY clause is used in a SELECT statement all output expressions must be either aggregate functions or columns present in the GROUP BY clause.

Complex Grouping Operations

Presto (Java) and Presto (C++) also support complex aggregations by using the GROUPING SETS, CUBE, and ROLLUP syntax. This syntax allows users to perform analysis that requires aggregation on multiple sets of columns in a single query. Complex grouping operations do not support grouping on expressions that are composed of input columns. Only column names or ordinals are allowed.

Complex grouping operations are often equivalent to a UNION ALL of simple GROUP BY expressions, as shown in the following examples. However, this equivalence does not apply when the source of data for the aggregation is nondeterministic.

GROUPING SETS

Grouping sets allow users to specify multiple lists of columns to group on. The columns not part of a given sublist of grouping columns are set to NULL.

```
SELECT * FROM shipping;
```

```
 origin_state | origin_zip | destination_state | destination_zip | package_weight
--------------+------------+-------------------+-----------------+----------------
 California   |      94131 | New Jersey        |            8648 |             13
 California   |      94131 | New Jersey        |            8540 |             42
 New Jersey   |       7081 | Connecticut       |            6708 |            225
 California   |      90210 | Connecticut       |            6927 |           1337
 California   |      94131 | Colorado          |           80302 |              5
 New York     |      10002 | New Jersey        |            8540 |              3
(6 rows)
```

GROUPING SETS semantics are demonstrated by this example query:

```
SELECT origin_state, origin_zip, destination_state, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state),
    (origin_state, origin_zip),
    (destination_state));
```

```
 origin_state | origin_zip | destination_state | _col0
--------------+------------+-------------------+-------
 New Jersey   | NULL       | NULL              |   225
 California   | NULL       | NULL              |  1397
 New York     | NULL       | NULL              |     3
 California   |      90210 | NULL              |  1337
 California   |      94131 | NULL              |    60
 New Jersey   |       7081 | NULL              |   225
 New York     |      10002 | NULL              |     3
 NULL         | NULL       | Colorado          |     5
 NULL         | NULL       | New Jersey        |    58
 NULL         | NULL       | Connecticut       |  1562
(10 rows)
```

The preceding query may be considered logically equivalent to a UNION ALL of multiple GROUP BY queries:

```
SELECT origin_state, NULL, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state

UNION ALL

SELECT origin_state, origin_zip, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state, origin_zip

UNION ALL

SELECT NULL, NULL, destination_state, sum(package_weight)
FROM shipping GROUP BY destination_state;
```

However, the query with the complex grouping syntax (GROUPING SETS, CUBE or ROLLUP) reads from the underlying data source only one time, while the query with the UNION ALL reads the underlying data three times. This is why queries with a UNION ALL may produce inconsistent results when the data source is not deterministic.

CUBE

The CUBE operator generates all possible grouping sets (that is, a power set) for a given set of columns. For example, the query:

```
SELECT origin_state, destination_state, sum(package_weight)
FROM shipping
GROUP BY CUBE (origin_state, destination_state);
```

is equivalent to:

```
SELECT origin_state, destination_state, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state),
    (origin_state),
    (destination_state),
    ());
```

```
 origin_state | destination_state | _col0
--------------+-------------------+-------
 California   | New Jersey        |    55
 California   | Colorado          |     5
 New York     | New Jersey        |     3
 New Jersey   | Connecticut       |   225
 California   | Connecticut       |  1337
 California   | NULL              |  1397
 New York     | NULL              |     3
 New Jersey   | NULL              |   225
 NULL         | New Jersey        |    58
 NULL         | Connecticut       |  1562
 NULL         | Colorado          |     5
 NULL         | NULL              |  1625
(12 rows)
```

ROLLUP

The ROLLUP operator generates all possible subtotals for a given set of columns. For example, the query:

```
SELECT origin_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY ROLLUP (origin_state, origin_zip);
```

```
 origin_state | origin_zip | _col2
--------------+------------+-------
 California   |      94131 |    60
 California   |      90210 |  1337
 New Jersey   |       7081 |   225
 New York     |      10002 |     3
 California   | NULL       |  1397
 New York     | NULL       |     3
 New Jersey   | NULL       |   225
 NULL         | NULL       |  1625
(8 rows)
```

is equivalent to:

```
SELECT origin_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS ((origin_state, origin_zip), (origin_state), ());
```

Combining multiple grouping expressions

Multiple grouping expressions in the same query are interpreted as having cross-product semantics. For example, the following query:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY
    GROUPING SETS ((origin_state, destination_state)),
    ROLLUP (origin_zip);
```

which can be rewritten as:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY
    GROUPING SETS ((origin_state, destination_state)),
    GROUPING SETS ((origin_zip), ());
```

is logically equivalent to:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state, origin_zip),
    (origin_state, destination_state));
```

```
 origin_state | destination_state | origin_zip | _col3
--------------+-------------------+------------+-------
 New York     | New Jersey        |      10002 |     3
 California   | New Jersey        |      94131 |    55
 New Jersey   | Connecticut       |       7081 |   225
 California   | Connecticut       |      90210 |  1337
 California   | Colorado          |      94131 |     5
 New York     | New Jersey        | NULL       |     3
 New Jersey   | Connecticut       | NULL       |   225
 California   | Colorado          | NULL       |     5
 California   | Connecticut       | NULL       |  1337
 California   | New Jersey        | NULL       |    55
(10 rows)
```

The ALL and DISTINCT quantifiers determine whether duplicate grouping sets each produce distinct output rows. This is particularly useful when multiple complex grouping sets are combined in the same query. For example, the following query:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY ALL
    CUBE (origin_state, destination_state),
    ROLLUP (origin_state, origin_zip);
```

is equivalent to:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state, origin_zip),
    (origin_state, origin_zip),
    (origin_state, destination_state, origin_zip),
    (origin_state, origin_zip),
    (origin_state, destination_state),
    (origin_state),
    (origin_state, destination_state),
    (origin_state),
    (origin_state, destination_state),
    (origin_state),
    (destination_state),
    ());
```

However, if the query uses the DISTINCT quantifier for the GROUP BY:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY DISTINCT
    CUBE (origin_state, destination_state),
    ROLLUP (origin_state, origin_zip);
```

only unique grouping sets are generated:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state, origin_zip),
    (origin_state, origin_zip),
    (origin_state, destination_state),
    (origin_state),
    (destination_state),
    ());
```

The default set quantifier is ALL.

GROUPING Operation

`grouping(col1, ..., colN) -> bigint`

The grouping operation returns a bit set converted to decimal, indicating which columns are present in a grouping. It must be used in conjunction with GROUPING SETS, ROLLUP, CUBE or GROUP BY and its arguments must match exactly the columns referenced in the corresponding GROUPING SETS, ROLLUP, CUBE or GROUP BY clause.

To compute the resulting bit set for a particular row, bits are assigned to the argument columns with the rightmost column being the least significant bit. For a given grouping, a bit is set to 0 if the corresponding column is included in the grouping and to 1 otherwise. For example, consider the query below:

```
SELECT origin_state, origin_zip, destination_state, sum(package_weight),
       grouping(origin_state, origin_zip, destination_state)
FROM shipping
GROUP BY GROUPING SETS (
        (origin_state),
        (origin_state, origin_zip),
        (destination_state));
```

```
origin_state | origin_zip | destination_state | _col3 | _col4
--------------+------------+-------------------+-------+-------
California   | NULL       | NULL              |  1397 |     3
New Jersey   | NULL       | NULL              |   225 |     3
New York     | NULL       | NULL              |     3 |     3
California   |      94131 | NULL              |    60 |     1
New Jersey   |       7081 | NULL              |   225 |     1
California   |      90210 | NULL              |  1337 |     1
New York     |      10002 | NULL              |     3 |     1
NULL         | NULL       | New Jersey        |    58 |     6
NULL         | NULL       | Connecticut       |  1562 |     6
NULL         | NULL       | Colorado          |     5 |     6
(10 rows)
```

The first grouping in the above result only includes the `origin_state` column and excludes the `origin_zip` and `destination_state` columns. The bit set constructed for that grouping is 011 where the most significant bit represents `origin_state`.

HAVING Clause

The HAVING clause is used in conjunction with aggregate functions and the GROUP BY clause to control which groups are selected. A HAVING clause eliminates groups that do not satisfy the given conditions. HAVING filters groups after groups and aggregates are computed.

The following example queries the `customer` table and selects groups with an account balance greater than the specified value:

```
SELECT count(*), mktsegment, nationkey,
       CAST(sum(acctbal) AS bigint) AS totalbal
FROM customer
GROUP BY mktsegment, nationkey
HAVING sum(acctbal) > 5700000
ORDER BY totalbal DESC;
```

```
 _col0 | mktsegment | nationkey | totalbal
-------+------------+-----------+----------
  1272 | AUTOMOBILE |        19 |  5856939
  1253 | FURNITURE  |        14 |  5794887
  1248 | FURNITURE  |         9 |  5784628
  1243 | FURNITURE  |        12 |  5757371
  1231 | HOUSEHOLD  |         3 |  5753216
  1251 | MACHINERY  |         2 |  5719140
  1247 | FURNITURE  |         8 |  5701952
(7 rows)
```

UNION | INTERSECT | EXCEPT Clause

UNION INTERSECT and EXCEPT are all set operations. These clauses are used to combine the results of more than one select statement into a single result set:

```
query UNION [ALL | DISTINCT] query
```

```
query INTERSECT [DISTINCT] query
```

```
query EXCEPT [DISTINCT] query
```

The argument ALL or DISTINCT controls which rows are included in the final result set. If the argument ALL is specified all rows are included even if the rows are identical. If the argument DISTINCT is specified only unique rows are included in the combined result set. If neither is specified, the behavior defaults to DISTINCT. The ALL argument is not supported for INTERSECTor EXCEPT.

Multiple set operations are processed left to right, unless the order is explicitly specified via parentheses. Additionally, INTERSECT binds more tightly than EXCEPT and UNION. That means `A UNION B INTERSECT C EXCEPT D`is the same as `A UNION (B INTERSECT C) EXCEPT D`.

UNION

UNION combines all the rows that are in the result set from the first query with those that are in the result set for the second query. The following is an example of one of the simplest possible UNION clauses. It selects the value 13 and combines this result set with a second query that selects the value 42:

```
SELECT 13
UNION
SELECT 42;
```

```
 _col0
-------
    13
    42
(2 rows)
```

The following query demonstrates the difference between UNION and `UNION ALL`. It selects the value 13 and combines this result set with a second query that selects the values 42 and 13:

```
SELECT 13
UNION
SELECT * FROM (VALUES 42, 13);
```

```
 _col0
-------
    13
    42
(2 rows)
```

```
SELECT 13
UNION ALL
SELECT * FROM (VALUES 42, 13);
```

```
 _col0
-------
    13
    42
    13
(2 rows)
```

INTERSECT

INTERSECT returns only the rows that are in the result sets of both the first and the second queries. The following is an example of one of the simplest possible INTERSECT clauses. It selects the values 13 and

42 and combines this result set with a second query that selects the value 13. Since 42 is only in the result set of the first query, it is not included in the final results.:

```
SELECT * FROM (VALUES 13, 42)
INTERSECT
SELECT 13;
```

```
 _col0
-------
    13
(2 rows)
```

EXCEPT

EXCEPT returns the rows that are in the result set of the first query, but not the second. The following is an example of one of the simplest possible EXCEPT clauses. It selects the values 13and 42 and combines this result set with a second query that selects the value 13. Since 13 is also in the result set of the second query, it is not included in the final result.:

```
SELECT * FROM (VALUES 13, 42)
EXCEPT
SELECT 13;
```

```
 _col0
-------
   42
(2 rows)
```

ORDER BY Clause

The `ORDER BY` clause is used to sort a result set by one or more output expressions:

```
ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...]
```

Each expression may be composed of output columns or it may be an ordinal number selecting an output column by position (starting at one). The `ORDER BY` clause is evaluated after any `GROUP BY` or `HAVING` clause and before any

`OFFSET, LIMIT or FETCH FIRST clause. The default null ordering is NULLS LAST, regardless of the ordering direction.`

OFFSET Clause

The `OFFSET` clause is used to discard a number of leading rows from the result set:

```
OFFSET count [ ROW | ROWS ]
```

If the `ORDER BY` clause is present, the `OFFSET` clause is evaluated over a sorted result set, and the set remains sorted after the leading rows are discarded:

```
SELECT name FROM nation ORDER BY name OFFSET 22;
```

```
      name
----------------
 UNITED KINGDOM
 UNITED STATES
 VIETNAM
(3 rows)
```

Otherwise, it is arbitrary which rows are discarded. If the count specified in the `OFFSET` clause equals or exceeds the size of the result set, the final result is empty.

LIMIT Clause

The `LIMIT` clause restricts the number of rows in the result set. `LIMIT ALL` is the same as omitting the `LIMIT` clause.

```
LIMIT { count | ALL }
```

The following example queries a large table, but the limit clause restricts the output to only have five rows (because the query lacks an `ORDER BY`, exactly which rows are returned is arbitrary):

```
SELECT orderdate FROM orders LIMIT 5;
```

```
 o_orderdate
-------------
 1996-04-14
 1992-01-15
 1995-02-01
 1995-11-12
 1992-04-26
(5 rows)
```

`LIMIT ALL` is the same as omitting the `LIMIT` clause.

If the `OFFSET` clause is present, the `LIMIT` clause is evaluated after the `OFFSET` clause:

```
SELECT * FROM (VALUES 5, 2, 4, 1, 3) t(x) ORDER BY x OFFSET 2 LIMIT 2;
```

```
 x
---
 3
 4
(2 rows)
```

TABLE SAMPLE

There are multiple sample methods:

BERNOULLI

Each row is selected to be in the table sample with a probability of the sample percentage. When a table is sampled using the Bernoulli method, all physical blocks of the table are scanned and certain rows are skipped (based on a comparison between the sample percentage and a random value calculated at runtime).

The probability of a row being included in the result is independent from any other row. This does not reduce the time required to read the sampled table from disk. It may have an impact on the total query time if the sampled output is processed further.

SYSTEM

This sampling method divides the table into logical segments of data and samples the table at this granularity. This sampling method either selects all the rows from a particular segment of data or skips it (based on a comparison between the sample percentage and a random value calculated at runtime).

The rows selected in a system sampling will be dependent on which connector is used. For example, when used with Hive, it is dependent on how the data is laid out on HDFS. This method does not guarantee independent sampling probabilities.

Note: Neither of the two methods allow deterministic bounds on the number of rows returned.

Examples:

```
SELECT *
FROM users TABLESAMPLE BERNOULLI (50);
```

```
SELECT *
FROM users TABLESAMPLE SYSTEM (75);
```

Using sampling with joins:

```
SELECT o.*, i.*
FROM orders o TABLESAMPLE SYSTEM (10)
JOIN lineitem i TABLESAMPLE BERNOULLI (40)
  ON o.orderkey = i.orderkey;
```

UNNEST

UNNEST can be used to expand an ARRAY or MAP into a relation. Arrays are expanded into a single column, and maps are expanded into two columns (key, value). UNNEST can also be used with multiple arguments, in which case they are expanded into multiple columns, with as many rows as the highest cardinality argument (the other columns are padded with nulls).UNNEST can optionally have a `WITH ORDINALITY` clause, in which case an additional ordinality column is added to the end. UNNEST is normally used with a `JOIN` and can reference columns from relations on the left side of the join.

Using a single array column:

```
SELECT student, score
FROM tests
CROSS JOIN UNNEST(scores) AS t (score);
```

Using multiple array columns:

```
SELECT numbers, animals, n, a
FROM (
  VALUES
    (ARRAY[2, 5], ARRAY['dog', 'cat', 'bird']),
    (ARRAY[7, 8, 9], ARRAY['cow', 'pig'])
) AS x (numbers, animals)
CROSS JOIN UNNEST(numbers, animals) AS t (n, a);
```

```
  numbers  |     animals      |  n   |  a
-----------+------------------+------+------
 [2, 5]    | [dog, cat, bird] |    2 | dog
 [2, 5]    | [dog, cat, bird] |    5 | cat
 [2, 5]    | [dog, cat, bird] | NULL | bird
 [7, 8, 9] | [cow, pig]       |    7 | cow
 [7, 8, 9] | [cow, pig]       |    8 | pig
 [7, 8, 9] | [cow, pig]       |    9 | NULL
(6 rows)
```

`WITH ORDINALITY` clause:

```
SELECT numbers, n, a
FROM (
  VALUES
    (ARRAY[2, 5]),
    (ARRAY[7, 8, 9])
) AS x (numbers)
CROSS JOIN UNNEST(numbers) WITH ORDINALITY AS t (n, a);
```

```
  numbers  | n | a
-----------+---+---
 [2, 5]    | 2 | 1
 [2, 5]    | 5 | 2
 [7, 8, 9] | 7 | 1
 [7, 8, 9] | 8 | 2
 [7, 8, 9] | 9 | 3
(5 rows)
```

Using a single map column:

```
SELECT
    animals, a, n
FROM (
    VALUES
```

```
        (MAP(ARRAY['dog', 'cat', 'bird'], ARRAY[1, 2, 0])),
        (MAP(ARRAY['dog', 'cat'], ARRAY[4, 5]))
) AS x (animals)
CROSS JOIN UNNEST(animals) AS t (a, n);
```

```
           animals          |  a   | n
----------------------------+------+---
 {"cat":2,"bird":0,"dog":1} | dog  | 1
 {"cat":2,"bird":0,"dog":1} | cat  | 2
 {"cat":2,"bird":0,"dog":1} | bird | 0
 {"cat":5,"dog":4}          | dog  | 4
 {"cat":5,"dog":4}          | cat  | 5
(5 rows)
```

Joins

Joins allow you to combine data from multiple relations.

CROSS JOIN

A cross join returns the Cartesian product (all combinations) of two relations. Cross joins can either be specified using the explicit `CROSS JOIN` syntax or by specifying multiple relations in the `FROM` clause.

Both of the following queries are equivalent:

```
SELECT *
FROM nation
CROSS JOIN region;

SELECT *
FROM nation, region;
```

The `nation` table contains 25 rows and the `region` table contains 5 rows, so a cross join between the two tables produces 125 rows:

```
SELECT n.name AS nation, r.name AS region
FROM nation AS n
CROSS JOIN region AS r
ORDER BY 1, 2;
```

```
     nation     |   region
----------------+-------------
 ALGERIA        | AFRICA
 ALGERIA        | AMERICA
 ALGERIA        | ASIA
 ALGERIA        | EUROPE
 ALGERIA        | MIDDLE EAST
 ARGENTINA      | AFRICA
 ARGENTINA      | AMERICA
...
(125 rows)
```

Qualifying Column Names

When two relations in a join have columns with the same name, the column references must be qualified using the relation alias (if the relation has an alias), or with the relation name:

```
SELECT nation.name, region.name
FROM nation
CROSS JOIN region;

SELECT n.name, r.name
FROM nation AS n
CROSS JOIN region AS r;

SELECT n.name, r.name
FROM nation n
CROSS JOIN region r;
```

The following query will fail with the error `Column 'name' is ambiguous`:

```
SELECT name
FROM nation
CROSS JOIN region;
```

USING

The `USING` clause allows you to write shorter queries when both tables you are joining have the same name for the join key.

For example:

```
SELECT *
FROM table_1
JOIN table_2
ON table_1.key_A = table_2.key_A AND table_1.key_B = table_2.key_B
```

can be rewritten to:

```
SELECT *
FROM table_1
JOIN table_2
USING (key_A, key_B)
```

The output of doing `JOIN` with `USING` will be one copy of the join key columns (`key_A` and `key_B`in the example above) followed by the remaining columns in `table_1` and then the remaining columns in `table_2`. Note that the join keys are not included in the list of columns from the origin tables for the purpose of referencing them in the query. You cannot access them with a table prefix and if you run `SELECT table_1.*, table_2.*`, the join columns are not included in the output.

The following two queries are equivalent:

```
SELECT *
FROM (
    VALUES
        (1, 3, 10),
        (2, 4, 20)
) AS table_1 (key_A, key_B, y1)
LEFT JOIN (
    VALUES
        (1, 3, 100),
        (2, 4, 200)
) AS table_2 (key_A, key_B, y2)
USING (key_A, key_B)

-----------------------------

SELECT key_A, key_B, table_1.*, table_2.*
FROM (
    VALUES
        (1, 3, 10),
        (2, 4, 20)
) AS table_1 (key_A, key_B, y1)
LEFT JOIN (
    VALUES
        (1, 3, 100),
        (2, 4, 200)
) AS table_2 (key_A, key_B, y2)
USING (key_A, key_B)
```

And produce the output:

```
 key_A | key_B | y1 | y2
-------+-------+----+-----
     1 |     3 | 10 | 100
     2 |     4 | 20 | 200
(2 rows)
```

Subqueries

A subquery is an expression which is composed of a query. The subquery is correlated when it refers to columns outside of the subquery. Logically, the subquery will be evaluated for each row in the surrounding query. The referenced columns will thus be constant during any single evaluation of the subquery.

Note: Support for correlated subqueries is limited. Not every standard form is supported.

EXISTS

The EXISTS predicate determines if a subquery returns any rows:

```
SELECT name
FROM nation
WHERE EXISTS (SELECT * FROM region WHERE region.regionkey = nation.regionkey)
```

IN

The `IN` predicate determines if any values produced by the subquery are equal to the provided expression. The result of `IN` follows the standard rules for nulls. The subquery must produce exactly one column:

```
SELECT name
FROM nation
WHERE regionkey IN (SELECT regionkey FROM region)
```

Scalar Subquery

A scalar subquery is a non-correlated subquery that returns zero or one row. It is an error for the subquery to produce more than one row. The returned value is `NULL` if the subquery produces no rows:

```
SELECT name
FROM nation
WHERE regionkey = (SELECT max(regionkey) FROM region)
```

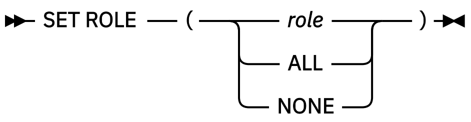
Note: Currently only single column can be returned from the scalar subquery.

SET ROLE statement

Synopsis

```
SET ROLE ( role | ALL | NONE )
```

Syntax Diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

`SET ROLE` sets the enabled role for the current session in the current catalog.

`SET ROLE role` enables a single specified role for the current session. For the `SET ROLE role` statement to succeed, the user who is running it must have a grant for the given role.

`SET ROLE ALL` enables all roles that the current user is granted for the current session.

`SET ROLE NONE` disables all the roles that are granted to the current user for the current session.

Limitations

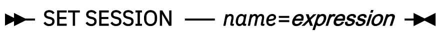
Some connectors do not support role management. See connector documentation for more details.

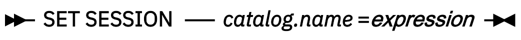
SET SESSION statement

Synopsis

```
SET SESSION name = expression
SET SESSION catalog.name = expression
```

Syntax Diagram





For information on reading syntax diagrams, see How to read syntax diagrams

Description

Set a session property value.

Examples

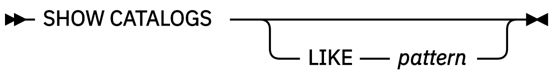
```
SET SESSION optimize_hash_generation = true;
SET SESSION hive.optimized_reader_enabled = true;
```

SHOW CATALOGS statement

Synopsis

```
SHOW CATALOGS [ LIKE pattern ]
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

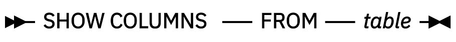
List the available catalogs. The LIKE clause can be used to restrict the list of catalog names.

SHOW COLUMNS statement

Synopsis

```
SHOW COLUMNS FROM table
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

List the columns in `table` along with their data type and other attributes.

SHOW CREATE TABLE statement

Synopsis

```
SHOW CREATE TABLE table_name
```

Syntax diagram

▶▶─ SHOW CREATE TABLE ── *table_name* ─▶◀

For information on reading syntax diagrams, see How to read syntax diagrams

Description

Show the SQL statement that creates the specified table.

Examples

Show the SQL that can be run to create the `orders` table:

```
SHOW CREATE TABLE sf1.orders;
```

```
               Create Table
-----------------------------------------
 CREATE TABLE tpch.sf1.orders (
    orderkey bigint,
    orderstatus varchar,
    totalprice double,
    orderdate varchar
 )
 WITH (
    format = 'ORC',
    partitioned_by = ARRAY['orderdate']
 )
(1 row)
```

SHOW CREATE VIEW statement

Synopsis

```
SHOW CREATE VIEW view_name
```

Syntax diagram

▶▶─ SHOW CREATE VIEW ── *view_name* ─▶◀

For information on reading syntax diagrams, see How to read syntax diagrams

Description

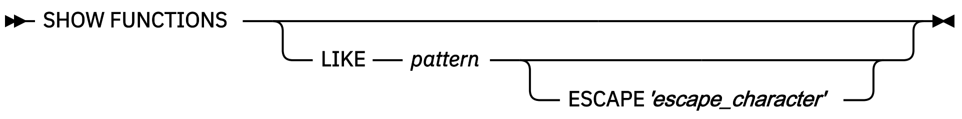
Show the SQL statement that creates the specified view.

SHOW FUNCTIONS statement

Synopsis

```
SHOW FUNCTIONS [ LIKE pattern [ ESCAPE 'escape_character' ] ]
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

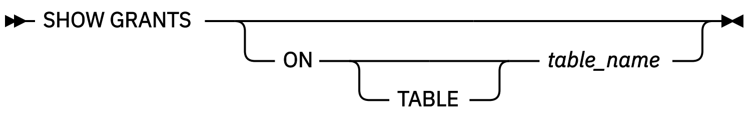
List all functions available for use in queries. The `LIKE` clause can be used to restrict the list of function names.

SHOW GRANTS statement

Synopsis

```
SHOW GRANTS [ ON [ TABLE ] table_name ]
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

List the grants for the current user on the specified table in the current catalog.

If no table name is specified, the command lists the grants for the current user on all the tables in all schemas of the current catalog.

The command requires the current catalog to be set.

Note: Ensure that authentication has been enabled before running any of the authorization commands.

Examples

List the grants for the current user on table `orders`:

```
SHOW GRANTS ON TABLE orders;
```

List the grants for the current user on all the tables in all schemas of the current catalog:

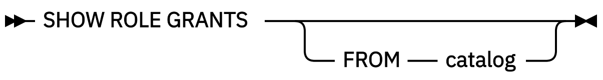
```
SHOW GRANTS;
```

SHOW ROLE GRANTS statement

Synopsis

```
SHOW ROLE GRANTS [ FROM catalog ]
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

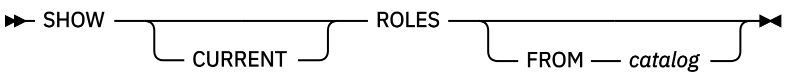
List non-recursively the `ROLE``s that have been granted to the session user in ``catalog`, or the current catalog if `catalog` is not specified.

SHOW ROLE statement

Synopsis

```
SHOW [CURRENT] ROLES [ FROM catalog ]
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

`SHOW ROLES` lists all the roles in `catalog` or in the current catalog if `catalog` is not specified.

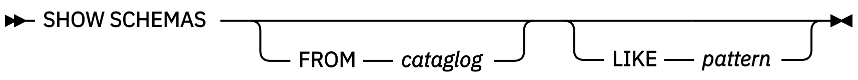
`SHOW CURRENT ROLES` lists the enabled roles for the session in `catalog`, or in the current `catalog` if `catalog`is not specified.

SHOW SCHEMAS statement

Synopsis

```
SHOW SCHEMAS [ FROM catalog ] [ LIKE pattern ]
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

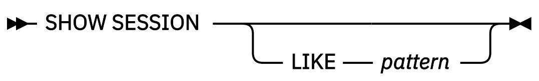
List the schemas in `catalog` or in the current catalog. The `LIKE` clause can be used to restrict the list of schema names.

SHOW SESSION statement

Synopsis

```
SHOW SESSION [ LIKE pattern ]
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

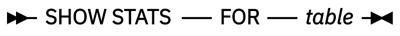
List the current session properties. The `LIKE` clause can be used to restrict the list of session properties.

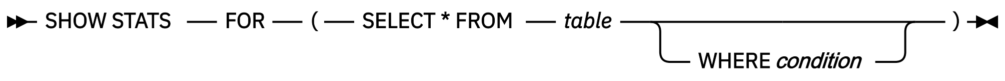
SHOW STATS statement

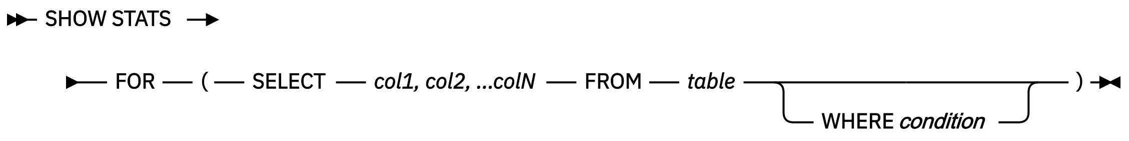
Synopsis

```
SHOW STATS FOR table
SHOW STATS FOR ( SELECT * FROM table [ WHERE condition ] )
SHOW STATS FOR ( SELECT col1, col2,... colN FROM table [ WHERE condition ] )
```

Syntax diagram







For information on reading syntax diagrams, see How to read syntax diagrams

Description

Returns approximated statistics for the named table or for the results of a (limited) query.

Statistics are returned for the specified columns, along with a summary row. (column_name will be `NULL` for the summary row).

Column	Description
column_name	The name of the column
data_size	The total size in bytes of all of the values in the column
distinct_values_count	The number of distinct values in the column
nulls_fractions	The portion of the values in the column that are `NULL`

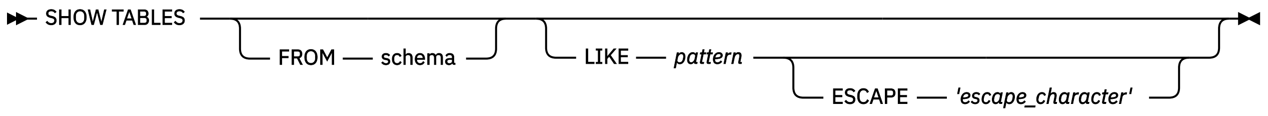
row_count	The number of rows (only returned for the summary row)
low_value	The lowest value found in this column (only for some types)
high_value	The highest value found in this column (only for some types)

SHOW TABLES statement

Synopsis

```
SHOW TABLES [ FROM schema ] [ LIKE pattern [ ESCAPE 'escape_character' ] ]
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

List the tables in `schema` or in the current schema. The `LIKE` clause can be used to restrict the list of table names.

TRUNCATE statement

Synopsis

```
TRUNCATE TABLE table_name
```

Syntax diagram

For information on reading syntax diagrams, see How to read syntax diagrams

Description

Deletes all rows from a table but not the table itself. It retains the table structure like columns, constraints, and indexes.

Examples

Truncate the table `orders`:

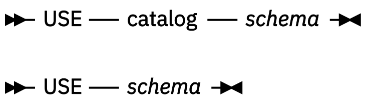
```
TRUNCATE TABLE orders;
```

USE statement

Synopsis

```
USE catalog.schema
USE schema
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Update the session to use the specified catalog and schema. If a catalog is not specified, the schema is resolved relative to the current catalog.

Examples

```
USE hive.finance;
USE information_schema;
```

VALUES statement

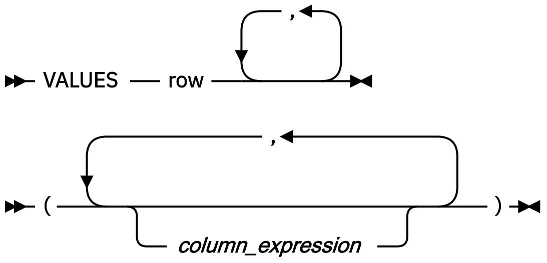
Synopsis

```
VALUES row [, ...]
```

where `row` is a single expression or

```
( column_expression [, ...] )
```

Syntax diagram



For information on reading syntax diagrams, see How to read syntax diagrams

Description

Defines a literal inline table.

VALUES can be used anywhere a query can be used (e.g., the FROM clause of a SELECT statement, an INSERT statement, or even at the top level). VALUES creates an anonymous table without column names, but the table and columns can be named using an AS clause with column aliases.

Examples

Return a table with one column and three rows:

```
VALUES 1, 2, 3
```

Return a table with two columns and three rows:

```
VALUES
    (1, 'a'),
    (2, 'b'),
    (3, 'c')
```

Return table with column `id` and `name`:

```
SELECT * FROM (
    VALUES
        (1, 'a'),
        (2, 'b'),
        (3, 'c')
) AS t (id, name)
```

Create a new table with column `id` and `name`:

```
CREATE TABLE example AS
SELECT * FROM (
    VALUES
        (1, 'a'),
        (2, 'b'),
        (3, 'c')
) AS t (id, name)
```

Presto data types

This section contains the list of data types supported by Presto.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Presto has a set of built-in data types that are described in the following sections. Additional types can be provided by plug-ins.

- Boolean
- Integer
- Floating-point
- Fixed-Precision
- String
- Date and time
- Structural
- Network Address
- UUID
- HyperLogLog
- KHyperLogLog (applicable only for Presto (Java))
- Quantile digest (applicable only for Presto (Java))
- T-Digest (applicable only for Presto (Java))

Boolean

BOOLEAN
This type captures Boolean values `true` and `false`.

Integer

TINYINT
An 8-bit signed two's complement integer with a minimum value of `-2^7` and a maximum value of `2^7 - 1`.

SMALLINT
A 16-bit signed two's complement integer with a minimum value of `-2^15` and a maximum value of `2^15 - 1`.

INTEGER
A 16-bit signed two's complement integer with a minimum value of `-2^31` and a maximum value of `2^31 - 1`. The name `INT` is also available for this type.

BIGINT
A 64-bit signed two's complement integer with a minimum value of `-2^63` and a maximum value of `2^63 - 1`.

Floating-point

REAL
A real is a 32-bit inexact, variable-precision implementing the IEEE Standard 754 for Binary Floating-Point Arithmetic.

DOUBLE
A double is a 64-bit inexact, variable-precision implementing the IEEE Standard 754 for Binary Floating-Point Arithmetic.

Fixed-precision

DECIMAL
A fixed precision decimal number. Precision up to 38 digits is supported but performance is best up to 18 digits.

The decimal type takes two literal parameters:

- **precision** - total number of digits
- **scale** - number of digits in fractional part. Scale is optional and defaults to 0.

Example type definitions: `DECIMAL(10,3)`, `DECIMAL(20)`

Example literals: `DECIMAL '10.3'`, `DECIMAL '1234567890'`, `1.1`

Note:

For compatibility reasons decimal literals without explicit type specifier (for example `1.2`) are treated as values of the `DOUBLE` type by default up to version 0.198. After 0.198 they are parsed as DECIMAL.

- System-wide property: `parse-decimal-literals-as-double`
- Session-wide property: `parse_decimal_literals_as_double`

String

VARCHAR
Variable length character data with an optional maximum length. Example type definitions: `varchar`, `varchar(20)`.

SQL supports simple and Unicode string literals:

- Literal string: `'Hello winter !'`
- Unicode string with default escape character: `U&'Hello winter \2603 !'`
- Unicode string with custom escape character: `U&'Hello winter #2603 !' UESCAPE '#'`

A Unicode string is prefixed with U& and requires an escape character before any Unicode character usage with 4 digits. In these examples `\2603` and `#2603` represent a snowman character. Long Unicode codes with 6 digits require a plus symbol + before the code. For example, use `\+01F600` for a grinning face emoji.

Single quotes in string literals can be escaped by using another single quote: `'It''s a beautiful day!'`

CHAR (applicable only for Presto (Java))

Fixed-length character data. A CHAR type without length that is specified has a default length of `1`. A `CHAR(x)` value always has x characters. For instance, casting dog to `CHAR(7)` adds 4 implicit trailing spaces. Leading and trailing spaces are included in comparisons of CHAR values. As a result, two character values with different lengths (`CHAR(x)` and `CHAR(y)` where `x != y`) will never be equal. But comparison of such values implicitly converts the types to the same length and pads with spaces so that the following query returns true:

```
sql SELECT cast('example' AS char(20)) = cast('example ' AS char(25));
```

A single quote in a CHAR literal can be escaped with another single quote:

```
sql SELECT CHAR 'All right, Mr. DeMille, I''m ready for my close-up.'
```

VARBINARY

Variable length binary data.

Note: Binary strings with length are not yet supported: `varbinary(n)`

JSON

JSON value type, which can be a JSON object, a JSON array, a JSON number, a JSON string, `true`, `false` or `null`.

Date and time

DATE

Calendar date (year, month, day).

Example: `DATE '2001-08-22'`

TIME (applicable only for Presto (Java))

Time of day (hour, minute, second, millisecond) without a time zone. Values of this type are parsed and rendered in the session time zone.

Example: `TIME '01:02:03.456'`

TIME WITH TIME ZONE (applicable only for Presto (Java))

Time of day (hour, minute, second, millisecond) with a time zone. Values of this type are rendered by using the time zone from the value.

Example: `TIME '01:02:03.456 America/Los_Angeles'`

TIMESTAMP

Instant in time that includes the date and time of day without a time zone. Values of this type are parsed and rendered in the session time zone.

Example: `TIMESTAMP '2001-08-22 03:04:05.321'`.

TIMESTAMP WITH TIME ZONE

Instant in time that includes the date and time of day with a time zone. Values of this type are rendered by using the time zone from the value.

Example: `TIMESTAMP '2001-08-22 03:04:05.321 America/Los_Angeles'`

INTERVAL YEAR TO MONTH

Span of years and months.

Example: `INTERVAL '3' MONTH`

INTERVAL DAY TO SECOND

Span of days, hours, minutes, seconds, and milliseconds.

Example: `INTERVAL '2' DAY`

Structural

ARRAY

An array of the given component type.

Example: `ARRAY[1, 2, 3]`

MAP

A map between the given component types.

Example: `MAP(ARRAY['foo', 'bar'], ARRAY[1, 2])`

ROW

Structure made up of named fields. The fields may be of any SQL type, and are accessed with field reference operator.

Example: `CAST(ROW(1, 2.0) AS ROW(x BIGINT, y DOUBLE))`

Network Address

IPADDRESS (applicable only for Presto (Java))

An IP address that can represent either an IPv4 or IPv6 address.

Internally, the type is a pure IPv6 address. Support for IPv4 is handled by using the *IPv4-mapped IPv6 address* range (**RFC 4291#section-2.5.5.2**). When creating an `IPADDRESS`, IPv4 addresses are mapped into that range.

When formatting an `IPADDRESS`, any address within the mapped range will be formatted as an IPv4 address. Other addresses are formatted as IPv6 using the canonical format defined in **RFC 5952**.

Examples: `IPADDRESS '10.0.0.1'`, `IPADDRESS '2001:db8::1'`

IPPREFIX (applicable only for Presto (Java))

An IP routing prefix that can represent either an IPv4 or IPv6 address.

Internally, an address is a pure IPv6 address. Support for IPv4 is handled by using the *IPv4-mapped IPv6 address* range (**RFC 4291#section-2.5.5.2**). When creating an `IPPREFIX`, IPv4 addresses are mapped into that range. Also, addresses are reduced to the first address of a network.

`IPPREFIX` values are formatted in CIDR notation, which is written as an IP address, a slash ('/') character, and the bit-length of the prefix. Any address within the IPv4-mapped IPv6 address range will be formatted as an IPv4 address. Other addresses are formatted as IPv6 using the canonical format defined in **RFC 5952**.

Examples: `IPPREFIX '10.0.1.0/24'`, `IPPREFIX '2001:db8::/48'`

UUID

UUID

This type represents a UUID (Universally Unique IDentifier), also known as a GUID (Globally Unique IDentifier), using the format defined in **RFC 4122**.

Example: `UUID '12151fd2-7586-11e9-8f9e-2a86e4085a59'`

HyperLogLog

HyperLogLog

A HyperLogLog sketch allows efficient computation of `approx_distinct()`. It starts as a sparse representation, switching to a dense representation when it becomes more efficient.

P4HyperLogLog (applicable only for Presto (Java))

A P4HyperLogLog sketch is similar to HyperLogLog, but it starts (and remains) in the dense representation.

KHyperLogLog (applicable only for Presto (Java))

KHyperLogLog

A KHyperLogLog is a data sketch that can be used to compactly represent the association of two columns. See KHyperLogLog Functions.

Quantile Digest (applicable only for Presto (Java))

QDigest

A quantile digest (qdigest) is a summary structure, which captures the approximate distribution of data for a given input set, and can be queried to retrieve approximate quantile values from the distribution. The level of accuracy for a qdigest is tunable, allowing for more precise results at the expense of space.

A qdigest can be used to give an approximate answer to queries asking for what value belongs at a certain quantile. A useful property of qdigests is that they are additive, meaning they can be merged without losing precision.

A qdigest may be helpful whenever the partial results of `approx_percentile` can be reused. For example, one may be interested in a daily reading of the 99th percentile values that are read over the course of a week. Instead of calculating the past week of data with `approx_percentile`, `qdigest` s could be stored daily, and quickly merged to retrieve the 99th percentile value.

See Quantile Digest Functions.

T-Digest (applicable only for Presto (Java))

TDigest

A t-digest is similar to qdigest, but it uses a different algorithm to represent the approximate distribution of a set of numbers. T-digest has better performance than quantile digests but only supports the

`DOUBLE` type. See T-Digest Functions.

How to read syntax diagrams

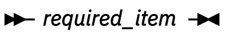
This section describes conventions that apply to the syntax diagrams that are used in IBM® documentation.

watsonx.data Developer edition

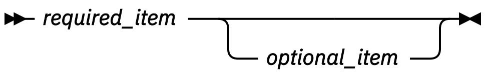
watsonx.data on Red Hat OpenShift

Apply the following rules when reading the syntax diagrams that are used in IBM watsonx.data documentation:

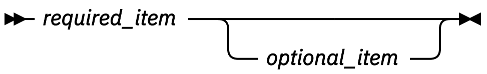
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
- Required items appear on the horizontal line (the main path).



- Optional items appear below the main path.

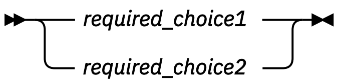


- If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

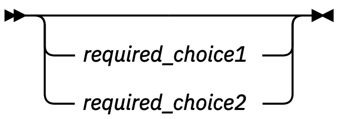


- If you can choose from two or more items, they appear vertically, in a stack.

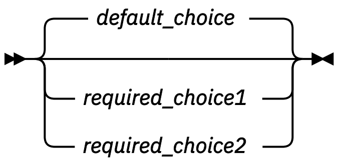
  If you *must* choose one of the items, one item of the stack appears on the main path.



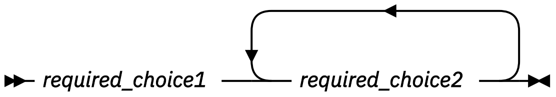
- If choosing one of the items is optional, the entire stack appears below the main path.



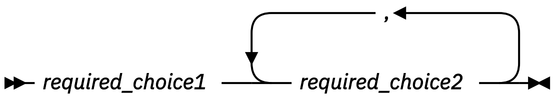
- If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.

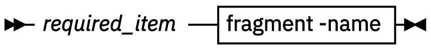


- If the repeat arrow contains a comma, you must separate repeated items with a comma.

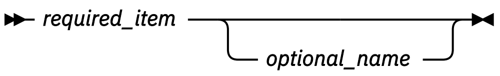


  A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



  fragment-name



- For some references in syntax diagrams, you must follow any rules described in the description for that diagram, and also rules that are described in other syntax diagrams. For example:
  - For *expression*, you must also follow the rules described in Expressions.
  - For references to *fullselect*, you must also follow the rules described in fullselect.
  - For references to *search-condition*, you must also follow the rules described in Search conditions.
- With the exception of XPath keywords, keywords appear in uppercase (for example, `FROM`). Keywords must be spelled exactly as shown.
- XPath keywords are defined as lowercase names, and must be spelled exactly as shown.
- Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Mixed-case behavior based on connectors

From IBM watsonx.data version 2.0.0, a new feature is available to switch between case-sensitive and case-insensitive behavior in Presto (Java) by using a mixed-case feature flag. This feature is not applicable for Presto (C++) engine.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

The mixed-case feature flag is set to OFF in Presto (Java) by default. The flag can be set to ON or OFF as required during deployment of the Presto (Java) engine. It is advised not to toggle between ON and OFF configurations after the deployment, as it may result in inconsistent system behavior.

Mixed-case feature flag: ON

The following section lists the behaviors of connectors if mixed-case feature flag is set to ON:

General behavior for all connectors:

- Column names in the `SELECT` query are case-insensitive.
- While referencing the data within the `WITH` clause, it is essential to use the exact alias name (case-sensitive) assigned during its definition. Using an incorrect alias triggers the following error message:

  ```
  Schema must be specified when session schema is not set.
  ```

- For tables that are specified using `WITH` and USE catalog.schema clauses, the alias name is case-sensitive. Using an incorrect alias triggers the following error message:

  ```
  Table does not exist.
  ```

Hive:

- The alias name is case-sensitive for `WITH` clause.

Iceberg:

- The alias name is case-sensitive for `WITH` clause.
- Column names in the `ALTER` table `RENAME` and `ALTER` table `DROP` are case-sensitive and accept only lowercase.

MongoDB:

- Column names for `DELETE` statement are case-insensitive.

Teradata:

- Tables with duplicate names cannot be created, regardless of the case.

Netezza:

- Tables with the same name and case cannot exist in multiple schemas that have the same name but different cases.

Delta Lake:

- Schema name in SELECT statement is case-sensitive.

Amazon Redshift:

- In the Amazon Redshift server, all the identifiers are case-sensitive if the configuration is set as:

```
enable_case_sensitive_identifier=true
```

- In the Amazon Redshift server, only lowercase identifiers are fetched if the configuration is set as:

```
enable_case_sensitive_identifier=false
```

SQL Server:

- Enable case-sensitive collation in the SQL Server to create duplicate tables in mixed-case.

Apache Pinot:

- Schema name is case insensitive regardless of the case sensitivity of Presto (Java) engine.

Mixed-case feature flag: OFF

The following section lists the behaviors of connectors if mixed-case feature flag is set to OFF:

MariaDB:

- Table and schema names should be in lowercase.

Greenplum:

- Table and schema names should be in lowercase.

Netezza:

- Schema name for USE statement is case-sensitive if you use it before `CREATE VIEW` statement.

Db2:

- Schema name for USE statement is case-sensitive if you use it before `CREATE VIEW` statement.

Note: Access control policies that are created in the mixed-case feature flag OFF mode will be applicable in OFF mode only and vice versa.

For more information on mixed-case feature flag behavior, supported SQL statements and supported data types matrices, see Support content.

FAQs

This topic is a collection of frequently asked questions (FAQs) about the IBM watsonx.data service.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

General

What is IBM watsonx.data?

IBM watsonx.data is an open, hybrid, and governed fit-for-purpose data store optimized to scale all data, analytics, and AI workloads to get greater value from your analytics ecosystem.

What can I do with IBM watsonx.data?

You can use IBM watsonx.data to collect, store, query, and analyze all your enterprise data with a single unified data platform. You can connect to data in multiple locations and get started in minutes with built-in governance, security, and automation. You can use multiple query engines to run analytics and AI workloads, reducing your data warehouse costs by up to 50%.

What are the key features of IBM watsonx.data?

The key features of IBM watsonx.data are:

1. An architecture that fully separates compute, metadata, and storage to offer ultimate flexibility.
2. Multiple engines such as Presto and Spark that provide fast, reliable, and efficient processing of big data at scale.
3. Open formats for analytic data sets, allowing different engines to access and share the data at the same time.
4. Data sharing between IBM watsonx.data, Db2® Warehouse, and Netezza Performance Server or any other data management solution through common Iceberg table format support, connectors, and a shareable metadata store.
5. Built-in governance that is compatible with existing solutions, including IBM Knowledge Catalog.
6. Cost-effective, simple object storage is available across hybrid-cloud and multicloud environments.
7. Integration with a robust ecosystem of IBM's best-in-class solutions and third-party services to enable easy development and deployment of key use cases.

Which data formats are supported in IBM watsonx.data?

The following data formats are supported in IBM watsonx.data:

1. Ingestion: Data ingestion in IBM watsonx.data supports .CSV and .parquet data file formats.
2. Create Table: Create table in IBM watsonx.data supports .CSV, .Parquet, .JSON, and .TXT data file formats.

What is the maximum size of the default IBM managed storage?

The IBM-managed storage is a default 10 GB storage.

Presto

What is Presto?

Presto is a distributed SQL query engine, with the capability to query vast data sets located in different data sources, thus solving data problems at scale.

What are the Presto server types?

A Presto installation includes three server types: Coordinator, Worker, and Resource manager.

What SQL statements are supported in IBM watsonx.data?

For information on supported SQL statements, see Supported SQL statements.

Metastore

What is HMS (Hive Metastore)?

Hive Metastore (HMS) is a service that stores metadata that is related to Presto and other services in a backend Relational Database Management System (RDBMS) or Hadoop Distributed File System (HDFS).

Installation and Set up

What version of Cloud Pak for Data do I need to use the latest version of IBM watsonx.data?

For version updates, see What's new in watsonx.data.

How can I create an IBM watsonx.data service instance?

To create an IBM watsonx.data service instance, see Installing watsonx.data.

How can I delete my IBM watsonx.data instance?

To uninstall the IBM watsonx.data service, see Uninstalling watsonx.data.

How can I configure an engine?

From the IBM watsonx.data web console, go to Infrastructure manager to configure an engine. For more information, see Provisioning a Presto engine.

How can I configure catalog or metastore?

To configure a catalog with an engine, see Associating a catalog with an engine.

How can I configure a storage?

From the IBM watsonx.data web console, go to Infrastructure manager to configure a storage. For more information, see Adding a storage-catalog pair.

Access

How can I manage IAM access for IBM watsonx.data?

Controlling access to the engines and other components is a critical requirement for many enterprises. To ensure that the resource usage is under control, IBM watsonx.data provides the ability to manage access controls on these resources. A user with admin privileges on the resources can grant access to other users. For more information about infrastructure access management, see Infrastructure access.

How can I add and remove the users?

To add or remove users, see Managing user access.

How is the access control for users provided?

To provide access control for users to restrict unauthorized access, see Data policy.

What is the process to assign access to a user?

To assign access to a user, see Managing access to the platform.

What is the process to assign access to a group?

To assign access to a group, see Managing user groups.

Presto Engine

How can I create an engine?

To create an engine, see Provisioning a Presto engine.

How can I pause and resume an engine?

To pause an engine, use one of the following methods:

1. Pausing an engine in list view:
   a. Click the overflow menu icon at the end of the row and click **Pause** icon. A pause confirmation dialog appears.
   b. Click **Pause**.
2. Pausing an engine in topology view:

    a. Hover over the engine that you want to pause and click the **Pause** icon. A pause confirmation dialog appears.
    b. Click **Pause**.

To resume a paused engine, use one of the following methods:

1. Resuming an engine in list view:
    a. Click the overflow menu icon at the end of the row.
    b. Click **Resume** icon.
2. Resuming an engine in topology view:
    a. Hover over the engine that you want to resume.
    b. Click **Resume** icon.

How can I delete an engine?

To delete an engine, see Deleting an engine.

How can I run SQL queries?

You can use the Query workspace interface in IBM watsonx.data to run SQL queries and scripts against your data. For more information, see Running SQL queries.

Databases and Connectors

How can I add a database?

To add a database, see Adding a database-catalog pair.

How can I remove a database?

To remove a database, see Deleting a database-catalog pair.

What data sources does IBM watsonx.data currently support?

IBM watsonx.data currently supports the following data sources:

1. IBM Db2
2. IBM Netezza
3. Apache Kafka
4. MongoDB
5. MySQL
6. PostgreSQL
7. SQL Server
8. Custom
9. Teradata
10. SAP HANA
11. Elasticsearch
12. SingleStore
13. Snowflake
14. IBM Data Virtualization Manager for z/OS

How can I load the data into IBM watsonx.data?

You can load the data into IBM watsonx.data by the following ways:

1. Web console: You can use the Ingestion jobs tab from the Data manager page to securely and easily load data into IBM watsonx.dataconsole. For more information, see Ingesting data by using web console.

2. Command Line Interface: You can load data into IBM watsonx.data through CLI. For more information, see Ingesting data by using command line interface (CLI) .
3. Creating tables: You can load or ingest local data files to create tables by using the `CREATE TABLE` option. For more information, see Creating tables.

How can I create tables?

You can create tables by the following methods:

1. Through the Data manager page by using the web console. For more information, see Creating tables.
2. Through the Command Line Interface. For more information, see Creating tables through CLI.

How can I create schema?

You can create schema by the following methods:

1. Through the Data manager page by using the web console. For more information, see Creating schemas.
2. Through the Command Line Interface. For more information, see Creating schema through CLI.

How can I query the loaded data?

You can use the Query workspace interface in IBM watsonx.data to run SQL queries and scripts against your data. For more information, see Running SQL queries.

Ingestion

What are the storage bucket options available?

The storage bucket options available are IBM Storage Ceph, IBM Cloud Object Storage (COS), AWS S3, and MinIO object storage.

What type of data files can be ingested?

Only .Parquet and .CSV data files can be ingested.

Can a folder of multiple files be ingested together?

Yes a folder of multiple data files be ingested. An S3 folder must be created with data files in it for ingesting. The source folder must contain either all Parquet files or all CSV files. For detailed information on S3 folder creation, see Preparing for ingesting data.

What commands are supported in the command line interface during ingestion?

For commands supported in the command line interface during ingestion, see Options and parameters supported in ibm-lh tool.

API documentation

API documentation provides instructions on how to use and integrate with an API. It includes information about the API endpoints, operations, resources, and services, as well as how to use them.

For detailed information on API docs, see https://cloud.ibm.com/apidocs/watsonxdata-software.

watsonx.data Developer edition

watsonx.data on Red Hat OpenShift

Index