dremio

Dremio Cloud

Operating Dremio Cloud Runbook

Introduction

This guide provides details on the tasks that should be periodically completed to maintain an operationally healthy Dremio Sonar and Arctic Catalog on Dremio Cloud. The document is broken into three sections; sonar project operations, use case/semantic layer operations, and catalog operations (Arctic catalog or AWS Glue data catalog). Each section contains a set of tasks. For each task, there is information about the task, why it is important, and what could happen if you do not complete it. The document is intentionally not focused on *how* to complete each task; for that there are appropriate links in the text to further reading material and guides.

Operations Tasks for Dremio Sonar Projects

This section presents all the Sonar project-related activities that a Dremio operative needs to perform regularly. The table below summarizes Dremio's recommended cadence for performing each activity that should be performed for every Sonar project. Please refer to the relevant subsections for further details on each activity.

Activity	Frequency	Method
Manage Sonar projects	Bi-Monthly	Dremio Console
Evaluate Engines (size and replicas), review Engine routing rules, adjust WLM (concurrency and Limits)	Monthly as standard or twice daily after notable events	Analyze query history for each project using sys.project.history.jobs
Capacity Planning for data growth and new use cases.	Semi-annually	Perform load tests and then analyze the results for each project using sys.project.history.jobs
Get content out of a Dremio Sonar catalog	During deployment cycles	Dremio REST API calls
Put content into a Dremio Sonar catalog	When Required	Dremio REST API calls

Manage Sonar Projects

Sonar projects will be created as private by default. To allow access to the project you need to grant access to a role or a user. The projects deactivate automatically if not in use for 15 days and a project is automatically activated when a user accesses it via Dremio Console, ODBC, or JDBC connection. Dremio advises archiving or deleting a project that is no longer in use. Once a project is archived only the user who has ownership privilege on the project will be able to reactivate the project. For more information see Managing Projects.

Evaluate Engines (size and replicas), Engine routing, WLM

Dremio has guidelines for setting up some sensible initial guardrails for engine routing and engines (size, concurrency, replicas) when the project is first created.

It is important to set up engines of appropriate size and set sensible concurrency and replica limits to prevent rogue queries from bringing down Executors unnecessarily. Information in sys.project.history.jobs can be used to identify the heavy memory hitters and route these

queries to larger engines with bigger max concurrency limits rather than risk OOM failure on a smaller engine that will impact other queries running concurrently on that engine.

Every month as standard, but as frequently as twice per day after notable events such as a new workload being added or removed in production or a significant increase\decrease in the number of users being given access to Dremio has occurred, you should analyze the query history to determine if a change in WLM rules (max concurrency per replica, min/max replicas), or a change in the number of executors in any of the engines is necessary. The query history is stored in the sys.project.history.jobs table.

When the volume of queries being simultaneously executed by the current set of executor nodes in an engine starts to reach a saturation point there are several key symptoms that Dremio exhibits. One of the biggest symptoms is increased sleep time during query execution, sleep time is incurred when a running query needs to wait for available CPU cycles due to all available CPUs being in operation.

Another symptom is an increased number of Out Of Memory exceptions occurring, even on queries that are not particularly heavy memory consumers; if a query uses a small amount of memory but needs a tiny bit more, if the request for that tiny bit more memory pushes Dremio over its limit, then that small query will be marked as Out Of Memory since it was the one that requested memory and it couldn't be allocated. Seeing these types of Out Of Memory exceptions indicates that the engine cannot handle the concurrency allowed on a single replica by the engine settings.

Additionally, watch for any error exceptions related to engine scaling, these indicate that the incoming concurrency of queries exceeds the max concurrency per replica and max replicas allowed by the engine settings.

These symptoms can be identified by analyzing the query history which gives reasons for query failure.

Failure to address these symptoms can result in increasing query times and an increasing number of query execution failures, which leads to a bad end user experience and poor satisfaction.

In the above circumstances, you can alleviate the issue by reviewing and adjusting the engine routing rules and engine parameters. This may include adding new engines and engine rules that send appropriate queries to new engines, adjusting max concurrency per replica and max replicas, or changing the size of the existing engines.

A good reason for creating a new engine is if a new workload gets introduced to Dremio, perhaps by a new department within an organization, and their queries are causing the existing engine setup to decrease performance. Creating a new engine to isolate the new workload, most likely by creating rules to route queries from users in that organization to the new engine is a useful way of segregating workloads.

In Dremio Cloud deployments you can configure new engines on-demand from the Dremio console, as described here, or adjust concurrency and replicas on existing engines to manage increased workloads.

Capacity Planning

Twice yearly, Dremio recommends performing an engine sizing exercise to gauge whether the engines are sized correctly for their current workload and how much concurrency expansion each engine can handle if the current workload increases. An engine sizing exercise cannot be used to gauge how much capacity an engine might need for workloads that aren't already present in Dremio, it is based purely on existing workloads.

A typical engine sizing exercise involves gathering a representative set of 50-100 queries per engine and using a load-testing framework to submit these queries at increasing concurrency rates. The tests should be repeated with different engine sizes, scaling-in or scaling-out number of nodes, and adjusting the minimum and maximum replicas for the engine. What to test and how to test it are covered in the Dremio Load Testing white paper.

The results from the test are available in the sys.project.history.jobs table, this can be reviewed and aggregated by individual queries per test to deduce at which concurrency you start to see enqueued times go up. From this information, you can infer what percentage-wise concurrency improvements you can expect by scaling up the number of replicas for the specific engine.

Ideally, this load test should run against the production project where the queries were obtained, but in the worst case it can be run against a Test/QA/UAT project if it is configured to be identical to the production project (or very close to it); the load test is meaningless if it's performed on a project that in no way resembles the production project.

Though this is not an exact indicator of what size your engine needs to be, it allows you to retrieve real metrics that you can use as predictors so that you can plan your scale-up of the number of nodes or scale-up of the number of replicas appropriately for what you anticipate future concurrency levels to be. Without performing any such sizing exercise it is difficult to say how many extra nodes or replicas you might need if query concurrency doubled or tripled for example.

Get Content out of a Dremio Sonar Catalog

During deployment cycles, several activities are performed repeatedly, for example moving a small number of resources from one Dremio environment to another. Dremio exposes REST APIs for performing programmatic catalog operations such as retrieving metadata for specific objects. By combining sequences of REST API calls you can retrieve sets of metadata from the Project.

These sequences of REST API calls can be used to export entire semantic layers from the Dremio Sonar Catalog to disk. Furthermore, you can develop scripts to check in the semantic objects into a version control repository such as GIT, if desired. The Dremio Sonar Rest APIs are listed here.

Put Content into a Dremio Sonar Catalog

During deployment cycles, several activities are performed repeatedly, for example moving a small number of resources from one Dremio environment to another. Dremio exposes REST APIs for performing programmatic catalog operations such as importing metadata from a local disk into Dremio. By combining sequences of REST API calls you can import metadata into Dremio in a controlled manner.

Assuming you have already exported the resources for your chosen folder onto a local disk, you could import the semantic objects to a different sonar catalog, or recreate them in the same sonar catalog. The Dremio Sonar REST APIs are listed here.

Use Case / Semantic Layer Operations Tasks

This section details all the use case-related activities a Dremio operative needs to perform regularly. The table below summarizes Dremio's recommended cadence for performing each activity. Please refer to the relevant subsections for further details on each activity.

Activity	Frequency	Method
Evaluate reflection usage	Monthly	Analyze sys.project.history.jobs data
Evaluate current worst-performing queries	Monthly	Analyze sys.project.history.jobs data
Evaluate query errors	Weekly	Analyze sys.project.history.jobs data
Self-serve project usage information	Weekly	Analyze sys.project.history.jobs data

Evaluate Reflection Usage

Every month Dremio recommends evaluating the reflection usage strategies being employed on the project. With great power comes great responsibility and you need to ensure that your users are not abusing their power to create reflections to the detriment of the overall operational efficiency of the project. Reflections only need adding when the circumstances are right for doing so and you also need to diligently evaluate and remove reflections that are not providing the value they should.

Evaluating Adding Reflections

When developing use cases in Dremio's semantic layer, it's often best to build out the use case iteratively without any reflections initially. Then, as you complete iterations you can run the queries and analyze the data in the query history to deduce which ones are taking the longest to execute and whether there are any common factors between a set of slow queries contributing to the slowness.

For example, perhaps there are a set of 5 slow queries which are each derived from a VDS that contains a join between two relatively large tables, in this situation, you might find that putting a raw reflection on the VDS that is performing the join helps to speed up all 5 queries because an Apache Iceberg representation of the join results will be created and can be automatically

used to accelerate views derived from the join. This allows you to get the query planning and performance benefits of Apache Iceberg, and you can even partition the reflection to accelerate queries the underlying data wasn't initially optimized for. This is an important pattern since it means you can leverage a small number of reflections to speed up potentially many workloads.

Raw reflections can also be useful in cases where you have large volumes of JSON or CSV data. Whenever this type of data is queried the entire data set needs to be processed, which can be inefficient. Adding a raw reflection over the JSON or CSV data again allows for an Apache Iceberg representation of that data to be created, which opens up all the planning and performance benefits that come with it.

Similar to the JSON/CSV situation described above, another use of raw reflections is simply to offload heavy queries from an operational data store. Often DBAs do not want their precious operational data stores (e.g. OLTP databases) being overloaded with analytical queries while they are busy processing billions of transactions, so in this situation, you can leverage Dremio raw reflections again to create that Apache Iceberg representation of the operational table and when a query comes in that needs the data it will read the reflection data as opposed to going back to the source.

Another important use case for raw reflections is joining on-premises data to cloud data. In this situation you will typically find that retrieving the on-premises data becomes the bottleneck for queries due to the latency in retrieving the data from the source system, therefore leveraging a raw reflection on the VDS where the data is joined together can almost always yield significant performance gains.

If you have connected Dremio to client tools and those client tools are issuing different sets of group-by queries against a VDS, if those group-by statements are taking too long to process compared to the desired SLAs then you might want to consider adding an aggregate reflection to the VDS to satisfy the combinations of dimensions and measures that are being submitted from the client tool.

For further best practices when considering how and where to apply reflections, visit this page.

For detailed instructions on how to create and update reflections, visit this page.

Failure to use Dremio reflections means potentially missing out on significant performance enhancements for some of your poorest-performing queries. However, creating too many reflections can also negatively impact the system as a whole. The misconception is often that

more reflections must be better, but considering the overhead in maintaining and refreshing them at intervals, the reflection refreshes can end up stealing valuable resources from your everyday workloads if they are not routed to appropriate engines.

Where possible, organize your queries by pattern. The idea here is that you try to create as few reflections as possible to service as many of your queries as possible, so finding those points in your semantic tree where a lot of queries go through can help accelerate a larger number of queries; the more reflections you have that may be able to accelerate the same query patterns the longer the planner will need to take evaluating which reflection will be best suited for accelerating the query being planned.

Evaluate the Removal of Unused Reflections

Analysis of the information Dremio captures about queries that have been executed which is available in sys.project.history.jobs, joined with data in system tables like sys.project.reflections and sys.project.materializations can provide details of the frequency each reflection is being leveraged. For any reflections not being leveraged, you can perform further analysis to determine if any of them are still being refreshed and if they are, how many times they have been refreshed in the reporting period and how many hours of execution time they have been consuming.

Checking for and removing unused reflections every month is good practice because it can reduce clutter in the reflection configuration. Additionally, it can often free up many hours of execution cycles that can be used for more critical workloads.

Evaluate Current Worst Performing Queries

Every month Dremio recommends you analyze the jobs submitted to Dremio by reviewing sys.project.history.jobs. One of the simplest analyses you can perform is the performance of our queries, which can take in several factors.

The first and probably easiest factor to consider is the overall execution time of a query, you want to identify the top 5-10 longest-running queries every month to understand why they are taking so long; is it the time taken to read data from the source, are you lacking CPU cycles, is the query spilling to disk, was the query queued at the start? Did it take a long time to plan? Many of these scenarios and how to look out for them in a query profile are covered in the Dremio Professional Services Training Module 7 called Query Tuning. Be sure to review the details of that module if possible.

The query data also allows us to focus on planning times; although a query might not appear in the top 5-10 slowest queries overall, it might still have high planning time. These queries

should also be investigated to determine the cause for the planning time, this could be due to the complexity of the query, or it could be because there are a lot of reflections being considered which could be an indicator that there are too many reflections defined in the environment - see the section called Evaluate the Removal of Unused Reflections for further details on identifying if there are redundant reflections in your project.

Evaluate Query Errors

Every week Dremio recommends analyzing the latest week of query history to identify all query failures. A query failure is any query that did not reach the COMPLETED state in the query history data. You can categorize your failures into groups and focus your efforts on the most serious or most frustrating for end users. Usually, the vast majority of errors you encounter will be syntax errors introduced as users write queries, these can usually be ignored.

It is important to understand the nature of your query failures. It can lead to opportunities to retrain or upskill users, highlight issues in Dremio, or find connectivity issues with data sources or client tools.

Ignoring your errors means issues may go undetected for longer, which might lead to an unwanted build-up of frustration from your user community.

Self-Serve Engine Usage Information

The query history provides valuable information to operations personnel who track Dremio usage over time. Dremio recommends tracking usage information at various levels of granularity every week so that you can spot the early signs of changes in activity trends on the Dremio project. The data captured in the query history allows you to visualize and report on aggregated query volumes by users, by queues, by time of day or day of week/month etc, allowing you to anticipate the need for changes to engine sizes in the project.

By not keeping track of usage information you risk operating an under-sized engine which puts strain on system resources and leads to higher query times if queries incur wait times during planning or execution. It also potentially adds to the risk of an increased number of Out Of Memory errors for individual queries on the project as load increases.

Operations Tasks for Dremio Arctic Catalog

This section presents all the Arctic project-related activities that a Dremio operative needs to perform regularly. The table below summarizes Dremio's recommended cadence for performing each activity that should be performed for every Arctic catalog. Please refer to the relevant subsections for further details on each activity.

Activity	Frequency	Method
Evaluate Table Optimization frequency for large datasets	Quarterly or before adding a large dataset to a branch for use in a Sonar project	Dremio Console
Evaluate Table cleanups and cut-off policies	Quarterly or before adding a large dataset to a branch for use in a Sonar project	Dremio Console
Evaluate Branches	Semi-annually or before merging a branch into main	Dremio Console
Audit User Accessibility	Semi-annually or after a new use case is added	Dremio Console
Get content out of a Dremio Arctic catalog	During deployment cycles	Dremio REST API calls
Put content into a Dremio Arctic catalog	When Required	Dremio REST API calls

Evaluate Table optimization frequency for large datasets

Query performance may be impacted as you ingest and make changes to data. For example, small files produced by data ingestion jobs result in slower queries because the query engine needs to read more files.

Every quarter or whenever introducing a new large physical dataset to a Sonar project, Dremio recommends that you evaluate the table optimization frequency for large datasets and the engines allocated on the Arctic project for the same.

For information on managing resources for table optimization or how to schedule optimizations, please see here.

Evaluate Table cleanup and policies for large datasets

Dremio supports DML and time travel queries on Iceberg tables by maintaining manifests lists and metadata files for every snapshot, depending on the number of changes to existing data the metadata grows in size and complexity, and this in turn impacts query performance. Every quarter Dremio recommends evaluating the table cleanup and cutoff policies set in each Arctic catalog. These are in place to determine which snapshots and associated policies to delete. For more information see here.

Evaluate Branches

Over time, architects/developers may create multiple branches as they explore and test data. Ideally, they merge their branches with the main branch at the appropriate time or delete the branch if they are abandoning the effort. Not doing so could mean having too many branches that if not documented appropriately will add to confusion.

Semi-annually basis Dremio recommends reviewing each Arctic catalog and evaluating all the branches. Branches that are abandoned or no longer in use can be deleted. Merge branches as needed. Some Application or Business perspectives may require some branches to have a multi-month or multi-year lifecycle, the emphasis here is to review and keep the Arctic catalog in a manageable state and the caveat that a branch once merged cannot be reversed.

Audit User Accessibility

Dremio recommends a semi-annual audit of user accessibility within the Arctic catalog. This ensures users and roles are only given access to resources they need to have access to. Arctic access privileges are listed here.

Dremio lists the recommended best practices for access controls here.

Get Content out of a Dremio Arctic Catalog

During deployment cycles, several activities are performed repeatedly, for example moving a small number of resources from one Dremio environment to another. Dremio exposes REST APIs for performing programmatic catalog operations such as retrieving metadata for specific objects. By combining sequences of REST API calls you can retrieve sets of metadata from the Arctic Catalog. The Dremio Arctic REST APIs are listed here.

The combination of REST API calls can export entire semantic layers from a Dremio Arctic Catalog to disk. Furthermore, you can develop scripts to check in the semantic objects into a version control repository such as GIT, if desired.

Put Content into a Dremio Arctic Catalog

During deployment cycles, several activities are performed repeatedly, for example moving a small number of resources from one Dremio environment to another. Dremio exposes REST APIs for performing programmatic catalog operations such as importing metadata from a local disk into an Arctic catalog. By combining sequences of REST API calls you can import metadata into an Arctic catalog in a controlled manner.

Assuming you have already exported the resources for your chosen folder onto a local disk, you could import the semantic objects to a different Arctic catalog or recreate them in the same Arctic catalog. The Dremio Arctic REST APIs are listed here.

Operations Tasks for AWS Glue Catalog

Dremio recommends using Arctic on Dremio Cloud, some may choose other catalogs such as AWS Data Glue for their Dremio Sonar Projects.

This section presents all the AWS Glue catalog activities that a Dremio operative needs to perform regularly. The table below summarizes Dremio's recommended cadence for performing each activity that should be performed for every AWS Glue catalog used in Dremio Cloud. Please refer to the relevant subsections for further details on each activity.

Activity	Frequency	Method
Table Optimization for large datasets	Quarterly or when adding a large dataset to a branch for use in a Sonar project	Manual, SQL to optimize table
Vacuum Table	Quarterly or per business needs	Manual, SQL to Vacuum Table

Table optimization for large datasets

Query performance may be impacted as you ingest and make changes to data. For example, small files produced by data ingestion jobs result in slower queries because the query engine needs to read more files.

Every quarter or whenever introducing a new large physical dataset to a Sonar project, Dremio recommends optimizing tables, either manually or by scheduling a job that submits a SQL command via Dremio SQL API or a JDBC/ODBC or Flight client. Details on the Optimize Table functionality can be found here.

Vacuum Table

Dremio supports DML and time travel queries on Iceberg tables by maintaining manifests lists and metadata files for every snapshot. Depending on the number of changes to existing data, the metadata grows in size and complexity, which impacts query performance. Every quarter or as per business needs, Dremio recommends performing a Vacuum Table command against the table. This can be done manually or by scheduling a job that submits a SQL command via Dremio SQL API or a JDBC/ODBC or Flight client. The Vacuum Table command removes snapshots you no longer need and the files (data files, the manifest file, the manifest list file, and partition stats files) associated only with them. Details on the Vacuum Table command can be found here.