dremio

Dremio Software

Operating Dremio Runbook

Introduction

This guide provides details of the tasks to periodically complete to maintain an operationally healthy Dremio cluster. The document is divided into infrastructure operations and use case/semantic layer operations. Each section contains a set of tasks. For each task there is information about what the task is, 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.

Infrastructure Operations Tasks

This section presents the infrastructure-related activities a Dremio operative needs to perform regularly. The infrastructure activities are common across multiple use cases that share a common Dremio cluster. The table below summarizes Dremio's recommended cadence for completing each activity. Please refer to the relevant subsections for further details on each activity.

Activity	Frequency	Method
Upgrade Dremio Major Version	Yearly	Follow upgrade steps for the infrastructure Dremio is installed on
Patch Dremio Minor Version	Quarterly	Follow upgrade steps for the infrastructure Dremio is installed on
Hotfix Dremio	As required	Follow upgrade steps for the infrastructure Dremio is installed on
Backup KV Store	Daily	dremio-admin backup on CRON schedule
Backup Non-KV Store resources	Daily	Needs script and CRON schedule
Clean KV Store	6-monthly once over 100GB	dremio-admin clean
Test restore of KV Store from backup	monthly	dremio-admin restore
Evaluate the number of executors	Monthly as standard, or twice daily after notable events	Analyze query history using Dremio. The query history is in queries.json
Evaluate Cluster/Engine Sizing	Semi-annually	Perform load tests and then analyze the results in queries.json using Dremio
Monitor Dremio cluster health	Real-time	Analyze JMX metrics and alert appropriately

Major Release Upgrades, Minor Release Patches and Hotfixes

Periodically, Dremio will create new major releases of Dremio Software to introduce new features and functionality and valuable updates to the product. It is recommended on approximately a yearly basis to ensure that you upgrade to some version of Dremio's major release number.

In addition, on approximately a quarterly basis, Dremio will create minor releases, known as patches, to introduce minor features, improve existing functionality and fix issues in a major release. It is sensible to stay no more than a couple of patch releases behind the most recent patch release, ensuring you remain well within the boundaries of the supported releases.

On an ad-hoc basis, Dremio may also introduce hotfixes to address immediate critical issues identified in the product; an example of this could be a hotfix to resolve a security vulnerability. Dremio typically releases Hotfixes for several recent patch versions, so when this happens Dremio recommends you install the hotfix that is most closely associated with the patch version of Dremio that you currently use.

Whether upgrading to a new major release version or applying a patch to your current major version, the process for performing the upgrade/patch is identical; the process only differs depending on which infrastructure Dremio is installed on.

- For Kubernetes upgrade steps you can refer to here.
- For Standalone or YARN-based upgrade steps using the rpm file refer to here.
- For Standalone or YARN-based upgrade steps using the tarball file refer to here.

For AWSE upgrades, the process is slightly different. You need first to stop the project in your current AWSE environment. Then you need to start up a new AWS Edition environment again from the AWS Marketplace, but this time choose the version of Dremio you are upgrading to. Once you have launched Dremio from the Marketplace, you can open your existing project in the new Dremio version.

You must upgrade Dremio on the cadences recommended above to stay within the range of supported versions; failing to do this puts you at risk of Dremio Support being unable to assist with issues that may arise. The list of currently supported Dremio versions is available here.

Be sure to incorporate upgrades into your overall Dremio operations strategy.

Backup KV Store

During regular operation Dremio stores essential metadata in a metastore local to the Master coordinator node; we refer to this metastore as Dremio's KV Store.

For customers with Iceberg metadata enabled, Dremio stores additional metadata for all parquet and iceberg datasets in the distributed storage area of the data lake.

Dremio recommends that every organization ensures daily backups of the Dremio Production KV store are taken. Dremio metadata related to all objects created in the semantic layer (e.g., data sources, PDS, VDS, reflections, wiki, tags, scripts etc.), profiles for executed jobs and user-uploaded files can be backed up.

Note: It is not currently possible to back up the iceberg metadata in the distributed store.

Automated KV Store backups can be set up as a CRON job on most non-kubernetes environments and Dremio recommends taking nightly backups. However, the frequency often depends on any RTO or RPO objectives an organization might have. For example, if the RPO is to have data as fresh as yesterday, then you'd need a daily backup, but if the RPO is to have it as fresh as last week, then a weekly backup may suffice.

Performing backups requires using the `dremio-admin backup` CLI command, which ships with Dremio. The official documentation states instructions for executing the dremio-admin backup commands in isolation on non-kubernetes clusters.

For details on how to perform backups on kubernetes environments, please refer to the following official documentation. Please note that the `dremio-admin backup` command must be run while Dremio is online and the command must be executed on the master-coordinator pod and NOT the Dremio Admin pod.

You can automate backups in kubernetes environments by creating a CronJob container to execute the relevant kubectl exec command. Details of setting up kubernetes CronJob for backups can be found in the Dremio-published Backing up and Restoring Dremio white paper.

In the event of a catastrophic failure, if there is no available backup, then this might result in the need to reinstall your entire Dremio cluster and manually recreate the entire semantic layer without any previous reference to the objects in the environment.

Backup Non-KV Store Resources

The KV Store backup mentioned in the previous section does not include the contents of the distributed store, such as iceberg metadata, acceleration cache, downloaded files and query results, nor does it include configuration files from the <dremio_home>/conf folder, any keytab files or certificate stores, logs from the <dremio_home>/logs folder or any installed community connectors.

Of these items, the ones that Dremio recommends you backup separately daily are the contents of the /conf folder and any files that do not form part of the standard install, such as the keytab files, certificates and community connector jar files. Logs can be helpful as backups for historical reference, but they are not critical resources for cluster restoration.

These files are as critical to restoring a Dremio cluster as the data in the KV Store since the /conf folder contains some useful configuration information, the keytab files and certificates contain the keys for facilitating secure communications with client tools and intra-cluster nodes and the community connector facilitate connections to community-supported data sources.

Automated nightly file backups can be set up using a combination of a CRON job and a script to perform the file copies.

With regards to Iceberg metadata, there is no benefit to backing this up since Iceberg files contain many internal references to absolute file paths. If you copy the Iceberg metadata from the distributed storage into another distributed storage location, everything breaks because of the absolute path references. So the only time you could re-use the Iceberg metadata when restoring a Dremio cluster is if the distributed store location of the new cluster is EXACTLY the same as the distributed store location of the Dremio cluster from which the backup was taken. If the distributed store location differs in the restored Dremio cluster, users must refresh the metadata for all Parquet data sets.

Clean KV Store

Over time, as data is added and removed in conjunction with the changes made in the semantic layer, the KV store can become fragmented and thus bloated in size. Dremio recommends a regular clean of the KV Store every six months once it reaches 100GB in size to avoid this becoming a significant issue.

The clean operation will delete any orphaned metadata entries (excluding parquet, ORC read-only and AVRO whose metadata is stored in distributed storage), orphaned profiles, old

jobs, old profiles and temporary dataset versions. It also re-indexes the data and compacts it to reduce the fragmentation on disk.

If the clean operation is not performed every six months once it is over 100GB in size, then this can cause a slowdown in metadata retrieval for non-parquet datasets, affecting the planning time of queries.

The clean operation is performed using Dremio's in-build `dremio-admin clean` CLI command. Please note that you must completely shut down ALL cluster nodes (coordinators and executors) before running this command. It is also recommended to take a cluster backup before running the command.

This link details the flags required when using `dremio-admin clean` to clean the KV store in non-kubernetes environments.

In kubernetes environments, the command must be run in the Dremio Admin pod while Dremio is in offline mode, meaning ***Dremio must not be running***.
Details of how to start the Dremio Admin pod and run the `dremio-admin clean` command are documented here.

Test Restore of KV Store from Backup

Dremio recommends testing the backups being created every month by importing them into a temporary environment. This will ensure that if a disaster hits the Dremio production server, you know you have a working backup you can restore into a new Dremio environment. The Dremio environment that a KV Store backup is restored into MUST be the same version of Dremio as the backup was originally taken from, therefore be careful to note the version of your Dremio Production environment and also note what the new version is if you upgrade it.

Please note there is no means to successfully backup and restore the acceleration folder in the distributed storage, which means after the metadata and all other backed-up files, such as the contents of the /conf folder, keytabs, certificates and community connector jars have been restored all of the reflections will need to be refreshed. The reflections can be refreshed via the Dremio UI or the catalog /refresh REST API call.

Failure to regularly test the restoration from a backup can, at worst, cause the loss of any recent functioning backup, leading to the possibility of forfeiture of your entire Dremio cluster. Dremio has experienced a situation where daily backups were corrupted due to a failure during the writing of the backup, caused by changes in folder access permissions by an administrator on the Dremio Coordinator node. Since nobody cared to test the backups periodically, the

problem went unnoticed. Only when a restore was required in the production environment did the customer realize they hadn't had a historical working backup for over four months.

The restore operation is performed using Dremio's in-build `dremio-admin restore` CLI command. This link provides details of the various flags and steps that are required when using `dremio-admin restore`.

In kubernetes environments, the command must be run in the Dremio Admin pod while Dremio is in offline mode, meaning ***Dremio must not be running***.
Details of how to start the Dremio Admin pod and run the `dremio-admin restore` command are documented here.

Evaluate Number of Executors

Monthly 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 the number of executors in our engines is necessary. That query history is stored in a file on the Dremio Master Coordinator called queries.json in Dremio Software.

When the volume of queries being simultaneously executed by the current set of executor nodes in an engine starts to reach a saturation point, Dremio exhibits several key symptoms. One of the most significant 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 in Dremio Software 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 very 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 by the queue settings.

These symptoms can be identified by analyzing the query history, which reports on sleep time and reasons why queries fail.

Failure to address these symptoms can result in increasing query times and an increasing number of queries failing due to Out Of Memory issues, leading to a bad end-user experience and poor satisfaction.

In both circumstances, in a cluster that supports engines, you can alleviate the issue by adding executor nodes to an existing engine or creating a new engine, then altering the Workload Management settings to use the engine changes. Bear in mind that queries cannot be executed across multiple engines.

Dremio has a best practice stating no engine should exceed ten executor nodes, assuming 128GB of memory and 16/32 cores per executor.

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 in 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 a standalone cluster, you can easily add nodes to the overall cluster. Dremio allocates queries to nodes based on how much memory is allocated to the various queues. You can set a new node as an executor in standalone deployments by editing the dremio.conf file, as described here.

As described here, you can configure new engines on-demand from the Dremio UI in YARN-based deployments.

In Kubernetes deployments, you can either add extra individual executors, add more executors to existing engines, or create new engines with however many executors you require. Of course, the requirement here is that your configured node pools for your Kubernetes cluster have enough available nodes. This page describes how to alter the number of standard executors, it is related to Amazon EKS but the steps apply to all flavors of Kubernetes. You can add engines to the values.yaml file by adding an engine name to the engines array inside the `executor` section and then creating an `engineOverride` section with the various engine settings; this is described in the Permanently Scaling Executors section here.

Cluster Sizing

Dremio recommends performing a cluster sizing exercise twice yearly to gauge whether the production cluster is sized correctly for its current workload and how much concurrency expansion the cluster could handle if the current workload increased. A cluster sizing exercise

cannot be used to gauge how much more capacity a cluster might need for workloads that aren't already present in Dremio; it is based purely on existing workloads.

A typical cluster sizing exercise involves gathering a representative set of 50-100 queries from the production cluster and using some load testing framework to send those queries into Dremio at increasing concurrency rates. The tests are often repeated with increased engine sizes.

Since the tests run against Dremio, the results, just like any other queries, are available in the queries.json file and this can be reviewed and aggregated by individual queries per test to deduce at which concurrency Dremio starts to be put under wait time and sleep time pressure. From this information, you can infer what percentage-wise concurrency improvements you can expect by scaling up the number of executors in the cluster or specific engines.

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

Though this is not an exact indicator of what size your cluster needs to be, it allows you to retrieve real metrics that you can use as predictors to plan your scale-up of Dremio nodes appropriately for what you anticipate future concurrency levels to be. Without performing any such sizing exercise, it is tough to say how many extra nodes you might need if your concurrency doubles or triples, for example.

Monitor Dremio Cluster Health

Dremio exposes system metrics via its JMX interface, which allows for near-real-time monitoring of what is happening regarding heap and direct memory consumption, garbage collection frequencies, lightweight thread usage, active, completed and failed jobs, jobs waiting in queues and more.

Dremio recommends connecting third-party monitoring tools to Dremio to capture and monitor these metrics. These third-party tools can often also be configured to access Dremio's REST API endpoints to issue periodic queries to Dremio to obtain further system information for display on monitoring dashboards.

Upon certain critical or warning thresholds being met for these various JMX and SQL metrics, the tools can be configured to send alerts to operations personnel who can investigate if issues are building in the Dremio cluster. Without such monitoring and alerting, the first time you hear of an issue with the Dremio cluster will be when you receive a call from an end user

complaining their queries are not executing, or their BI dashboard is not refreshing. Monitoring the system allows you to identify and fix potential issues before they happen.

This document provides information on some of the more useful metrics to capture and sensible alerting thresholds you can set.

Use Case / Semantic Layer Operations Tasks

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

Activity	Frequency	Method
Re-balance Workload Management	Monthly or automated daily	Analyze queries.json data manually or use scripts (to automate) to deduce queue thresholds.
Evaluate reflection usage	Monthly	Analyze queries.json data
Review and adjust metadata refresh frequencies	Quarterly	Dremio UI
Evaluate current worst-performing queries	Monthly	Analyze queries.json data
Evaluate query errors	Weekly	Analyze queries.json data
Self-serve cluster usage information	Weekly	Analyze queries.json data
Get content out of Dremio spaces	During deployment cycles	Dremio REST API calls
Put content into Dremio spaces	During deployment cycles	Dremio REST API calls

Re-balance Workload Management (rules, queues, engines)

The Workload Management (WLM) feature of Dremio Software provides the capability to manage cluster resources and workloads.

Dremio has guidelines for setting up some sensible initial guardrails for queue and job memory limits and initializing the query cost thresholds for when queries will get routed to either the Low or High Cost Queries queues when the cluster is first created.

For Dremio Software installations where no engines or just a single engine is configured and therefore all queries get routed to the same set of executors, it is essential to set up queue and query memory limits and set sensible concurrency limits to prevent rogue queries from bringing down Executors unnecessarily. It is a lot better to have Dremio identify and cancel a single query because it recognizes it exceeds the set memory limits than it is to let that query run and cause out-of-memory issues on an Executor, which will then cause all queries being handled by that executor to fail.

Since the workloads and volumes of queries change over time, every month the WLM settings for query cost thresholds should be re-evaluated and adjusted to re-balance the proportion of queries that flow to each of the query cost-based queues.

The document containing these guidelines can be found here.

Evaluate Reflection Usage

On a monthly basis Dremio recommends evaluating the reflection usage strategies being employed on the cluster. 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 cluster. You only need to add reflections when the circumstances are right. You must also diligently evaluate and remove reflections that do not provide 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 to begin with. Then, as you complete iterations, you can run the queries and analyze the data in the query history to deduce which ones take 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 five 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 five 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 that the underlying data wasn't initially optimized for. This is a critical pattern since it means you can leverage a small number of reflections to speed up potentially many workloads.

Raw reflections can be helpful in cases where you have large volumes of JSON or CSV data. Whenever this type of data is queried, the entire data set must 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 and opens up all of those 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 significant use case often requiring 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 you could miss out on significant performance enhancements for some of your poorest-performing queries. However, creating too many reflections can also have a negative impact on the system as a whole. The misconception is often that more reflections must be better, but when you consider the overhead in maintaining

and refreshing them at intervals, the reflection refreshes can end up stealing valuable resources from your everyday workloads.

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 our queries as possible, so finding those points in our semantic tree where a lot of queries go through can help us 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, available in queries.json, joined with data in system tables like sys.reflections and sys.materializations can provide details of the frequency each reflection present in Dremio is being leveraged. For any that are 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 cluster execution time they have been consuming.

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

Review and Adjust Metadata Refresh Frequencies

Dremio recommends reviewing the refresh frequencies for all your data sources every quarter at a minimum or whenever you add a new data source. This is to ensure they are set appropriately based on what you know about the frequency with which metadata changes in the data source.

The default metadata refresh set against data sources is every 1 hour. For the vast majority of data sources this is far too frequent. If the data in the sources only gets updated for example once every 6 hours, then it doesn't make sense to refresh the data sets every 1 hour; instead, you could change the refresh schedule to every 6 hours in the data source settings. See Scheduling Metadata Refreshes in the Dremio docs.

Further to the above recommendation, since metadata refreshes can be scheduled at the data source level, overridden at each individual PDS level, and performed programmatically, it makes sense to review each new data source to understand where best to place the most appropriate setting. For example, for data lake sources it could be acceptable to set a long metadata refresh schedule (e.g. 3000 weeks) at the data source level so that the scheduled

refresh is very unlikely to fire, and then perform an `ALTER TABLE .. REFRESH METADATA` command as part of the ETL process because there you know when the data generation has completed. For relational sources it's usually OK to set a high value (like days) for the refresh schedule at the source level, but then for PDSs where you know they will be changing more frequently, you can override the source setting directly on the PDS.

Often datasets get updated as part of overnight ETL runs. In this situation it doesn't make sense to refresh the metadata of the dataset until you know the ETL process is finished. To handle this, organizations create a script that triggers the manual refresh of each dataset in the ETL process once they see the dataset ETL has been completed. The script can either call the Dremio SQL REST API or make JDBC/ODBC queries to exercise the `ALTER TABLE..REFRESH METADATA` command. An extension to this approach is that sometimes the ETL processes do not fully update an existing dataset; instead they might change specific partitions or create new partitions. In this situation, to speed up the metadata refresh process you can use the script to tell Dremio only to refresh the changed or new partitions (note this will work with Parquet and iceberg data sets but not CSV or JSON).

If you have a data source with many datasets in it but the vast majority never change their structure or never have new files added, then it makes little sense to refresh those sources on a fixed schedule. Instead, set the metadata to never refresh at the source and set up scripts to trigger a manual refresh against a specific dataset using the `ALTER TABLE..REFRESH METADATA` syntax.

Suppose you set the metadata refresh schedule to never refresh and have no scripting mechanism to refresh your metadata. In that case, the fallback situation for Dremio when a query comes in, if the planner notices that the metadata is stale or invalid, is to perform an inline metadata refresh during the query planning phase. This can of course have a negative impact on the duration of the query execution since it will also incorporate that metadata refresh duration.

Dremio also recommends that you add a dedicated metadata refresh engine if you haven't added one already. This ensures the executors will service all metadata refresh activities in isolation from other workloads. This avoids any problems with metadata refresh workloads taking CPU cycles and memory away from business-critical workloads and ensures they have the best chance of finishing in a timely manner.

Evaluate Current Worst Performing Queries

Every month, Dremio recommends you analyze the jobs submitted to Dremio by writing the contents of daily queries.json files to the data lake and then exposing them as a single PDS.

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? A lot of these scenarios and how to look out for them in a query profile are covered in Dremio's Training Module 7, called Query Tuning, so be sure to review the details of that module.

The query data also allows us to focus on planning times. High planning time queries should also be investigated to determine the cause for the planning time; this could be due to the complexity of the query, it could be because it needed to do an inline metadata refresh due to stale metadata (check metadata refresh schedules, see the section called Review and Adjust Metadata Refresh Frequencies above), 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 Remove unused Reflections for further details on identifying if there are redundant reflections in your cluster).

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 queries that are most serious or frustrating for end users. Usually, the vast majority of errors you encounter will be syntax errors introduced as users write queries; these can generally be ignored.

Understanding the nature of your query failures can lead to opportunities to retrain or upskill users and highlight issues in Dremio or connectivity issues to 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 Cluster 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 weekly so that you can spot the early signs of changes in activity trends on the Dremio cluster. 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, enabling you to anticipate the need for changes to engine sizes in the cluster.

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

Migrate/Get Content out of Dremio Spaces

During deployment cycles, there are several activities that customers will perform 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 Dremio in a much more granular way than can be achieved using the standard `dremio-admin backup` functionality.

This approach is helpful when you have multiple teams working in different spaces within a Dremio environment and you want to migrate, for example, only one space from a development environment into a test environment or from a test environment into production.

Community-created scripts exist to automate the retrieval of metadata from Dremio. These scripts leverage combinations of Dremio REST API calls described above to extract the metadata for sets of objects defined in a configuration file. The scripts are packaged inside a tool called Dremio Cloner, which is available here.

Given the situation described above, you can use the Dremio Cloner tooling to export all resources associated with the chosen space from Dremio onto a local disk. From there you could develop some script to check the objects into a version control repository such as Git if desired.

Dremio Cloner is a useful example of how to perform these actions, however you are free to build your own pipelines to interact with the Dremio REST API to achieve the same outcome.

Migrate/Put Content into Dremio

During deployment cycles, there are several activities that customers will perform 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 much more granular way than can be achieved using the standard `dremio-admin restore` functionality.

This approach is useful when you have multiple teams working in different spaces within a Dremio environment and you want to migrate, for example, only one space from a development environment into a test environment or from a test environment into production.

Community-created scripts exist to automate metadata import from a local disk into Dremio. These scripts leverage combinations of Dremio REST API calls to import metadata for sets of objects defined in a configuration file previously exported from a Dremio environment. The scripts are packaged inside a tool called Dremio Cloner, which is available here.

Given the situation described above and assuming you have already exported the resources for your chosen space onto a local disk, then you can use the Dremio Cloner tooling to import those resources into the target Dremio environment without affecting any of the resources outside of the space you imported.

Dremio Cloner is a useful example of how to perform these actions, however you are free to build your own pipelines to interact with the Dremio REST API to achieve the same outcome.