

Dremio v26 S3-Compatible Storage Validation Kit

# Introduction

This validation kit provides a comprehensive guide for S3-compatible storage vendors to set up Dremio v26 in an on-premises Kubernetes environment and run the TPC-DS 1TB (SF1000) benchmark. By following this guide, vendors can evaluate how Dremio performs on their object storage and ensure compatibility and optimal configuration. We will cover everything from cluster deployment and dataset preparation to ingestion strategies and benchmarking. The guide includes scripts for automating data ingestion and optimization using Apache Iceberg tables (both partitioned and clustered layouts), as well as instructions for running the 99 TPC-DS queries and measuring performance.

***By completing the steps in this guide, your storage solution will be certified as a Dremio Validated Storage Solution.***

**Why TPC-DS?**

TPC-DS is a widely accepted decision support benchmark that simulates a complex retail analytics workload with 24 tables (7 fact tables and 17 dimensions) and 99 SQL queries . It is an industry-standard way to measure query engine performance on large-scale data (1 TB in this case). Multiple vendors and researchers use TPC-DS to compare systems. Using TPC-DS allows us to assess both storage throughput and query optimization features on Dremio as well as storage side configurations.

### **Scope:**

This guide assumes the user is an engineer familiar with Kubernetes and data lake technologies, but perhaps new to Dremio’s specifics. We focus on Dremio’s *on-premises* (self-hosted) deployment and features relevant to Iceberg tables on object storage. Cloud services (like Dremio Cloud) are outside our scope. We will use Kubernetes for deployment, an S3-compatible object store as the data source, and the Iceberg table format for storing data in the object store.

By the end of this guide, you will have:

* **Deployed Dremio v26 on Kubernetes** with an appropriate configuration for 1TB scale.
* **Loaded the TPC-DS 1TB dataset** into your S3-compatible storage in Parquet format (one file per partition subdirectory to avoid small files).
* **Ingested the dataset into Dremio** as Iceberg table
* Created View Driven Reflections for each of the 99 TPC-DS queries.
* **Run the full TPC-DS 99 query suite** in a single-user session and measured the total execution time (wall-clock).
  + One run with No Reflections
  + One run with Reflections
* Prepared a **results checklist and template** for consistent result submission (including environment details and performance outcomes for both ingestion strategies).

Throughout the guide, we include example commands, SQL scripts, and best practices. Short explanations accompany each step, and references to official documentation or related benchmarks are provided for further insight.

[Introduction](#_pxudamm1u1k9)

[Scope:](#_x6l825g04924)

[Validation Checklist](#_jq1yed69tky9)

[Hardware Infrastructure Setup](#_k7nxqriwhplt)

[Dremio Setup](#_srssjlbl1xgw)

[Data Staging](#_mizbmdlj9gqm)

[Performance Testing](#_8am5bs95v25d)

[Access to Dremio v26](#_7s799b18bqz2)

[Environment Setup & Installation](#_eo2jng817vuu)

[Environment Setup & Installation](#_c2iap04vizsw)

[Kubernetes Cluster and Hardware](#_mxbwe27glwd0)

[Dedicated Storage Buckets](#_8ej93a4xcarm)

[Dremio Compute Resources](#_v1023urnqhzr)

[Performance Testing Node](#_devzfzwzyqb5)

[Install Kubernetes](#_onpwdb2j8e3z)

[Configure Access to your S3 buckets](#_ndckhq79yv05)

[Installing Dremio v26 via Helm](#_o6af4ur2lts0)

[1. Install Helm](#_t7b7zkgojwwo)

[2. Prepare Kubernetes Environment](#_w3652hsdtrp9)

[2. Set Quay Login Information](#_eimtwr5qrpgr)

[3. Create values-overrides.yaml Configuration](#_qunctiuq2v7b)

[4. Install Dremio with Helm](#_xkm7a1ga85uv)

[Iceberg Catalog Setup](#_a38f729g56zn)

[Connecting the S3-Compatible Object Store in Dremio](#_rnz0wplfqtjz)

[Create Folders for Datasets](#_czmaq855iqwo)

[Access the Dataset in Parquet](#_fp8issezgg34)

[Generating Test Data](#_je6cirb2ox5n)

[Uploading Data set to S3-compatible Storage](#_wo4hchobcy1l)

[Creating Iceberg Tables](#_9zxfncqwwhar)

[Creating Iceberg Views](#_nzh48a3m23jh)

[Performance Testing](#_y7zscqmrct7y)

# Validation Checklist

### **Hardware Infrastructure Setup**

* S3-Compatible Storage Deployed
* Dremio Compute Resources Deployed
* Kubernetes Deployed
* Performance Testing Compute Resource Deployed

### **Dremio Setup**

* Helm installed successfully
* Dremio installed successfully
* Storage paths correctly mapped in Helm (paths.dist and Iceberg catalog)
* All Object Sources available in Dremio (tpcds, iceberg, dremio)

### **Data Staging**

* TPC-DS data generated
* TPC-DS data staged in tpcds bucket
* TPC-DS data ingested as Iceberg tables
* TPC-DS views created from Iceberg tables
* Dremio reflections created and tested

### 

### **Performance Testing**

* JMeter test setup and tested
* Non-Reflections Test plan execution
* Reflections Test planexecution
* Query results and performance metrics captured
* Output submitted to Dremio

# Access to Dremio v26

* First, you must sign up for a quay.io profile at <https://quay.io> . Use the email address associated with your download in the next step.
  + *Request access to the dremio helm charts by emailing or slacking your support specialist.*
* Next, you will need to obtain a license for Dremio v26. To do so, visit <https://www.dremio.com/get-started> and select “Enterprise Edition”.
  + *This will provide you with a license key.*
* Finally, request access to the performance testing repository located at <https://github.com/nateMeinzer/perf_testing/blob/main/README.md>

# Environment Setup & Installation

This section describes how to provision a Dremio v26 cluster on Kubernetes and configure it to use an S3-compatible storage system. The environment is designed to support a 1TB TPC-DS benchmark using Apache Iceberg tables and Dremio’s distributed query engine. Instructions have been updated to reflect changes in the v26 Helm-based deployment architecture.

## Kubernetes Cluster and Hardware

To replicate realistic performance results, the Dremio cluster should be sized according to internal recommendations for 1TB-scale Iceberg benchmarking.

### 

### 

### 

### 

### Dedicated Storage Buckets

Provision **3TB** of S3-compatible storage with **three separate buckets**:

| **Bucket Name** | **Purpose** | **Size Estimate** |
| --- | --- | --- |
| dremio | Dremio’s internal distributed storage for metadata, uploads, job results, and reflections | 1TB |
| iceberg | Iceberg table metadata and data files (partitioned parquet tables) | 2TB |
| tpcds | Raw Parquet files converted from the TPC-DS dataset | 2TB |

Each bucket should be hosted on an S3-compatible object store (e.g., MinIO, Cloudian, Wasabi) and exposed to Dremio via HTTP endpoints. TLS and IAM are not required for this validation kit.

Dremio requires consistent and performant object storage. **Ensure high-throughput, low-latency connectivity** between the Kubernetes nodes and the object store (i.e., same VPC or subnet where possible). If using MinIO, deploy it in the same Kubernetes cluster or a peered one.

### Dremio Compute Resources

Deploy Dremio using Helm with the following node configuration:

| **Node Type** | **Count** | **Instance Type** | **Specs** |
| --- | --- | --- | --- |
| Coordinator | 1 | m5d.2xlarge | 8 vCPU, 32 GB RAM |
| Executors | 8 | m5d.8xlarge | 32 vCPU, 128 GB RAM |

Executors should be configured with fast local storage (via Kubernetes emptyDir backed by NVMe or SSD) for:

* Query spill directories
* Reflection materializations
* Local caching

Helm charts will automatically configure local volumes as spill/cache directories. If running on a cloud provider, ensure node groups support NVMe or attach SSD volumes manually.

### Performance Testing Node

Set up a dedicated machine for running the JMeter test harness and data generation.

| **Purpose** | **Instance Type** | **Processor** | **Storage** |
| --- | --- | --- | --- |
| JMeter | m5d.2xlarge | 8 vCPU, 32 GB RAM | 1.5Tb Storage |

This machine must have:

* Java 11+
* JMeter installed
* Network access to the Dremio coordinator over JDBC (default port 31010)
* Python 3
* Various requirements listed in the repository
* 1.5TB Storage

***The script is designed to be run on a MacOS, so preference for that if available. If not, Linux may be used with minor alterations.***

### Install Kubernetes

You may use your own Kubernetes management tool. We recommend minikube or MicroK8s for testing.

Upon setup, create 9 nodes, and ensure you have the available CPUs and RAM as required above.

Make the above resources accessible to Kubernetes.

For more information, please see [Kubernetes Options](https://docs.dremio.com/current/deploy-dremio/kubernetes-deployment-options/).

### Configure Access to your S3 buckets

Now, you will need to configure access for your S3-compatible storage:  
  
Follow the instructions here <https://docs.dremio.com/current/deploy-dremio/configuring-kubernetes/#configuring-storage-for-dremio-catalog>

1. Create a Kubernetes secret named catalog-server-s3-storage-creds to access the configured location. Here is an example for S3 using an access key and secret key:

export AWS\_ACCESS\_KEY\_ID=<username>

export AWS\_SECRET\_ACCESS\_KEY=<password>  
 kubectl create secret generic catalog-server-s3-storage-creds \  
 --namespace $NAMESPACE \  
 --from-literal awsAccessKeyId=$AWS\_ACCESS\_KEY\_ID \  
 --from-literal awsSecretAccessKey=$AWS\_SECRET\_ACCESS\_KEY

  
 For S3-compatible storage providers (e.g., MinIO), the access keys should be the username and password.

## Installing Dremio

#### 1. Install Helm

brew install helm

For more information, please visit <https://helm.sh/docs/intro/install/>

#### 2. Prepare Kubernetes Environment

Create a namespace for Dremio:

kubectl create namespace dremio-v26



Set Context to Namespace:

kubectl config set-context --current --namespace=dremio-v26



#### 2. Set Quay Login Information

Enter your username and login when prompted:

helm registry login quay.io

#### 

**If you do not have access to the repository on quay, please contact your Dremio support specialist.**

#### 3. Create values-overrides.yaml Configuration

Create a file named values-overrides.yaml locally in your working directory. This file contains custom deployment values used during the Helm install process.

dremio:

trialImagePullCredentials: "" <-- use quotes

license: "" <-- use quotes

tag: 26.0.0

image:

repository: quay.io/dremio/dremio-enterprise

coordinator:

web:

auth:

enabled: true

type: "internal"

resources:

requests:

cpu: 6

memory: 24Gi

limits:

cpu: 8

memory: 30Gi

distStorage:

type: aws

aws:

bucketName: "dremio"

path: "/dremio-internal"

authentication: "accessKeySecret"

credentials:

accessKey: "minioadmin"

secret: "minioadmin"

extraProperties:

<property>

<name>fs.s3a.endpoint</name>

<value>http://minio:9000</value>

</property>

<property>

<name>fs.s3a.path.style.access</name>

<value>true</value>

</property>

<property>

<name>fs.s3a.connection.ssl.enabled</name>

<value>false</value>

</property>

catalog:

name: iceberg

externalAccess:

enabled: false

tls:

enabled: false

storage:

location: s3://dremio/iceberg

type: S3

s3:

region: us-east-1 # Dummy but required

roleArn: arn:aws:iam::000000000000:role/dummy # Dummy but required

endpoint: http://minio:9000 # Your S3-compatible endpoint

pathStyleAccess: true # Required for MinIO

skipSts: true # Required for MinIO

useAccessKeys: true # Required for MinIO

credentials:

accessKey: minioadmin

secret: minioadmin

extraProperties: |

<property>

<name>fs.s3a.endpoint</name>

<value>http://minio:9000</value>

</property>

<property>

<name>fs.s3a.path.style.access</name>

<value>true</value>

</property>

<property>

<name>fs.s3a.connection.ssl.enabled</name>

<value>false</value>

</property>

<property>

<name>fs.s3.compat</name>

<value>true</value>

</property>

service:

type: LoadBalancer

Save this in a folder for your project, and then navigate to this folder.

#### 4. Install Dremio with Helm

Run the helm install and wait for all Dremio pods to be up. You should see the coordinator and all executors in the Running state. You can check the logs to ensure no errors on startup.

helm install dremio

--version=26.0.0-p1-build-4 -f values-overrides.yaml

-n dremio-v26



Monitor pod status:

kubectl get pods -n dremio-v26



Once all pods are running, port-forward to access the UI:

kubectl port-forward svc/dremio-coordinator 9047:9047 -n dremio-v26



Create an admin username/password if prompted.

At this point, you have a running Dremio v26 cluster on Kubernetes. The next step is to create your Dremio catalog and connect your S3-compatible storage to prepare the dataset.

https://docs.dremio.com/current/deploy-dremio/configuring-kubernetes/

## Data Lakehouse Setup

### Iceberg Catalog Setup

**Please note:** the installation above using the yaml file should have created a Dremio catalog for Iceberg objects*. If not, you may set up a catalog manually following the steps below, but this should not be necessary.*

After deploying Dremio, the next step is to configure the Dremio Catalog to enable Iceberg operations backed by your S3-compatible object store. This will integrate your iceberg bucket to Dremio.

* Open the Dremio UI and navigate to Project Settings > Catalogs.
* Click + Add Catalog and select Dremio Catalog.

For S3-compatible systems, select **AWS** as the storage type.

In the dialog that appears, choose Master Credentials when prompted. *Vended credentials do not currently work with the Dremio query engine for S3-compatible backends.*

Enter the following values based on your environment:

* Storage Location: `s3://iceberg`
* Region: `us-east-1` (or your actual region)
* Endpoint: `http://<s3-compatible-host>:<port>`
* Enable *Path Style Access*
* Optionally configure the STS Endpoint if your object store supports STS; otherwise, enable *Skip STS*
* Provide the same Access Key and Secret Key used during Kubernetes secret creation

Once saved, the Dremio Catalog is active and ready to manage Iceberg metadata. All Iceberg tables you create will use this catalog for tracking metadata, and it will be used to validate reads/writes for your performance tests.

## 

### Connecting the S3-Compatible Object Store in Dremio

Dremio treats external storage as **Data Sources**. Our goal is to register the bucket that will hold tcpds parquet data to a S3-compatible system as a data source in Dremio so that Dremio can read/write datasets from it.

* In the Dremio UI, select “Add Source”
* Select AWS S3
* General Tab
  + Name: “tpcds”
  + Pass authentication (no need to Assume an IAM role unless required by storage)
    - AWS Access Key: your username
    - AWS Access Secret: your password
  + De-select “Encrypt Connection”
* Advanced Options Tab
  + Select “*Enable asynchronous access when possible”*
  + Select “*Enable compatibility mode”*
  + Select “*Enable file status check”*
  + Root Path: “/”
  + Default CTAS Format: Parquet
  + Connection Properties:
    - fs.s3a.path.style.access = true
    - fs.s3a.endpoint = “S3 endpoint” (do not include https://)
    - fs.s3a.connection.ssl.enabled = false
  + Allowlisted buckets
    - tpcds
  + Enable local caching when possible
* Reflection refresh Tab
  + Disable reflection refresh
* Metadata
  + Select *“Remove dataset definitions if underlying data is unavailable.”*
  + **Select *“Automatically format files into physical datasets when users issue queries”.***
* Save

### Create Folders for Datasets

For each sample size you run, you will want a folder to separate the sample data.

You can set these values in the .env file (ICEBERG\_FOLDER\_NAME & S3\_FOLDER\_NAME). You need to create these folders in both:

* Tpcds bucket - for raw parquet files
* Iceberg catalog store - for iceberg tables

We recommend creating a folder for a sample (qualifying set) and use this to store all objects for that sample size. For example, if you create 1GB of sample data to test a query run, then create a folder “small\_sample” in both tpcds and iceberg data sources in Dremio.

Be sure to set these values in the .env file before generating data.

### 

Preparing the TPC-DS 1TB Dataset

## Access the Dataset in Parquet

With the infrastructure in place, the next step is to get the 1TB TPC-DS data into your S3 store in the required format. Our goal is to have each of the 24 TPC-DS tables in Parquet format, with **one file per subdirectory** (i.e., avoid multiple small files in any one folder).

## Generating Test Data

We will not generate test data using a repository with a test generation tool. Repository can be found here: [**https://github.com/nateMeinzer/perf\_testing/blob/main/README.md**](https://github.com/nateMeinzer/perf_testing/blob/main/README.md)

For purposes of this test, please generate 1TB data (i.e. Scale = 1000), although you may run the queries against a smaller qualifying set with Scale set to 1.   
  
If you do intend to run two sets of datasets of different sizes, please create a folder for each dataset size and ensure the environment variables set in your repository are configured properly to pull and deploy the objects.

Instructions in the repository will walk you through this step, but from the repository root directory, execute:

python tpcds.py generate --scale X

Where scale is an integer.

This will generate data .dat format.

## 

## Uploading Data set to S3-compatible Storage

Now that the files have generated, we will convert them to parquet and upload them to S3 bucket “tcpds”.

***Important: Please set your S3 variables in the .env file.***From the repository directory, execute:

python tpcds.py upload

This will transform the .dat files to parquets, store them in your repository, and then upload them to S3-compatible storage.

Again, if you are generating a qualifying set of data with scale=1 (recommended), please create a subfolder for this purpose.

## Creating Iceberg Tables

Next, we will create iceberg tables from the uploaded parquet tables. Again, ensure the .env variables are set properly, and then from the root directory execute:

python iceberg.py tables

This will create an iceberg tables for the files in your S3\_BUCKET\_NAME, S3\_FOLDER\_NAME from the S3 object store called “tpcds”.

## Creating Iceberg Views

Next, we will create views from the 99 queries from the tpc-ds benchmark. We will use these views when we benchmark Dremio reflections, later on.

python iceberg.py views

This will create an iceberg view for each query file in the query directory.

## Creating Reflections

[TBD]

# Performance Testing

With the data in place, the final step is to execute the TPC-DS query suite (99 queries) on each dataset variation and measure the performance. The queries should be run in a single session sequentially (one after the other) to simulate a single-user throughput test (concurrency of 1). We will detail how to run them and how to measure the time properly.

## Prerequisites:

### Java 11+

If Java is not already installed on your machine:

* Install Java 11 or later
* On macOS: run java -version to confirm. If it's missing, install via Homebrew:

brew install openjdk



Make sure it is in your PATH.

### jMeter 5.6.3

This guide uses **Apache JMeter 5.6.3**.

* Download from: <https://jmeter.apache.org/download_jmeter.cgi>
* Unzip the folder somewhere safe (e.g., ~/jmeter-5.6.3)
* Make sure the bin/ directory is in your PATH

### Dremio Driver

We will be using the legacy JDBC driver and ***not the Arrow Flight driver.***

* Download here: <https://repo1.maven.org/maven2/org/apache/arrow/flight-sql-jdbc-driver/17.0.0/flight-sql-jdbc-driver-17.0.0.jar>

Please copy this into your main jmeter installation under:

<jmeter-folder>/lib/ ← it should be with the other drivers.

**Important:** You need to add the following line to your shell configuration file to persist these changes:

Edit your ~/.zshrc or ~/.bashrc by executing the following:

export JVM\_ARGS="-Dio.netty.tryReflectionSetAccessible=true --add-opens=java.base/java.nio=ALL-UNNAMED --add-opens=java.base/sun.nio.ch=ALL-UNNAMED"

  
Reset your terminal window.

### Dremio Cache

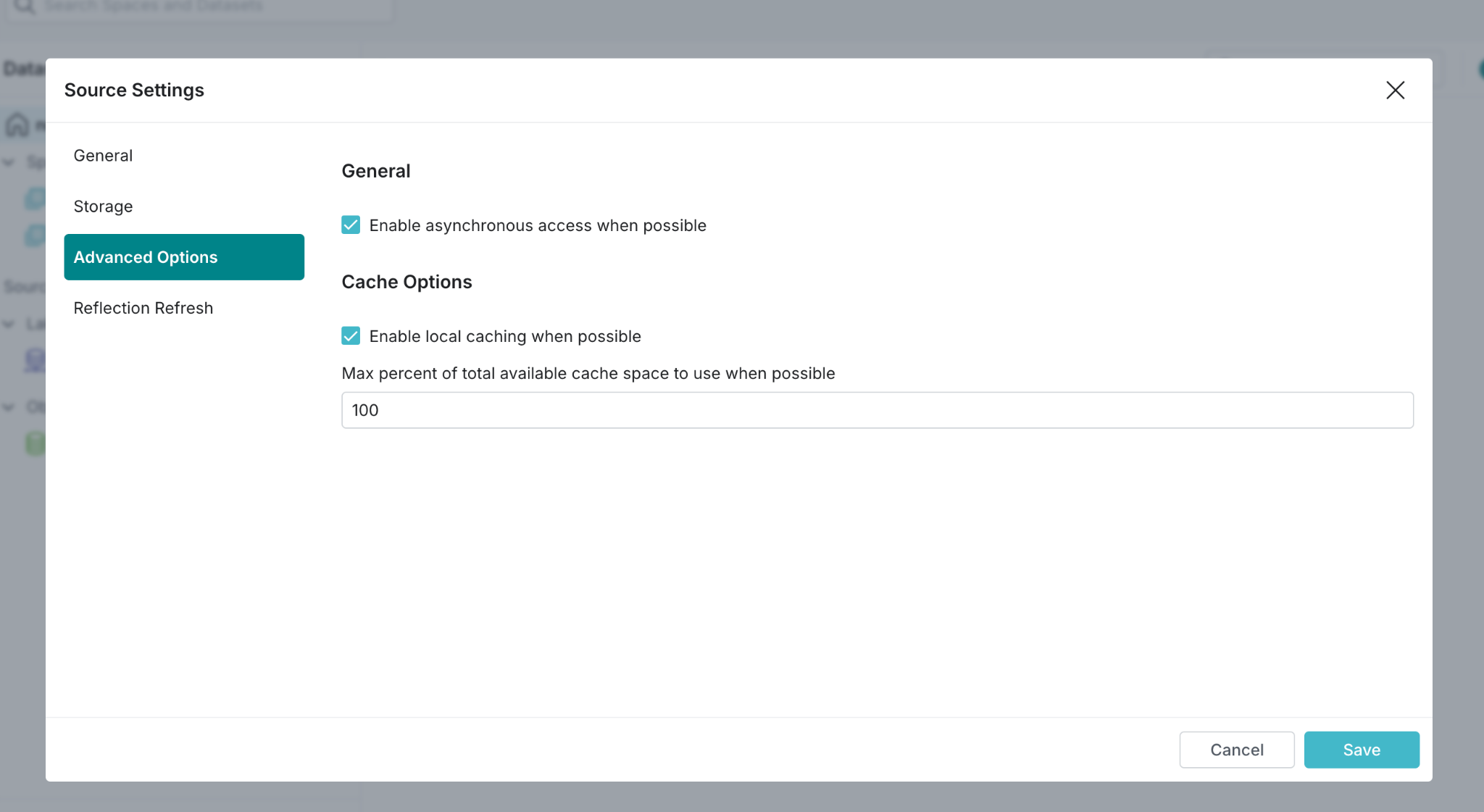
There are two caching mechanisms in Dremio: C3 & Results Cache.  
  
We wil enable C3 and disable Results Cache for all query runs.

#### Enable C3 Cache

You can locate this setting in the UI.  
  
Locate the data source: “iceberg”, right click it, and select “settings”.

On the Advanced Options tag,

* select *Enable asynchronous access when possible*
* Cache Options:
  + Select *Enable Location Caching*
  + Select *100* as value



*C3 Cache is enabled for this data source.*

#### Disable Results Cache

You will need to disable the results cache. To do this, please contact your support specialist to execut ethe following:

ALTER SYSTEM SET "dremio.results\\_cache\\_enabled" = FALSE;



## Testing Jmeter

### Testing in UI

Navigate to the folder: <repository root folder> /benchmark-kit in terminal and execute the following command: jMeter.

You should see a window open and jMeter running.

* In the JMeter UI, go to **File > Open**
* Navigate to: benchmark-kit/testplans/partial\_test\_plan.jmx
* Open the plan
* Click the green **Start** button (play icon)
* This test will run 3 queries, and you can monitor it’s progress by clicking the yellow triangle icon in the top right corner: ⚠️

After the queries are finished executing, you can see the output in results:

* Partial\_results.csv will show the partial results of the query runs, with 1 line per query (so 3 lines here)

### Testing via CLI

**Important:** *You should never run all 99 queries via the UI. Always use the CLI for large datasets and query jobs.*

Navigate to the folder: <repository root folder> /benchmark-kit in terminal and execute the following command: jMeter.

jmeter -n -t testplans/partial\\_test\\_plan.jmx -j results/debug.log



Your test sould be running. Note that you can find the read the debug.log live as the tests are running.

Verify all is working, and then prepare to run all 99 queries.

## Benchmark Testing

Each benchmark run will execute a fixed workload of **99 queries**, derived from a modified TPC-DS suite. The goal is to compare query performance across storage backends and Dremio configurations with and without reflections enabled.

We will run a total or 10 executions.

* No Reflections - 5 sets of 99 queries run with *Reflections Disabled*
* With Reflections - 5 sets of 99 queries run with *Reflections Enabled*

For each of the 99 queries:

* All 5 execution times per mode (with and without reflections) will be collected
* The **minimum** and **maximum** times will be **discarded** to reduce the impact of outliers
* The remaining **3 runs** will be averaged to produce a stable latency metric

The output will be written to a results log, and we have a script that automates the computation of the averages. The output of this script will be returned to Dremio for publication and recording.

### No Reflections Testing

To execute the full test plan without reflections, navigate to the benchmark-kit folder and execute the following:

### jmeter -n -t testplans/full\_test\_plan\_noref.jmx -j results/debug\_full\_noref.log

* Results: full\_results.csv
* Log: debug\_full\_noref.log

After each test run, check your logs, and run the test again. Do this a total of 5 times. Ensure that the logs reflect 99 queries having successfully run.

### With Reflections Testing

Now we will execute with reflections.

First, be sure to enable reflections.

[TBD]

To execute the full test plan without reflections, navigate to the benchmark-kit folder and execute the following script:

### jmeter -n -t testplans/full\_test\_plan\_withref.jmx -j results/debug\_full\_withref.log

* Results: full\_results.csv
* Log: debug\_full\_withref.log

After each test run, check your logs, and run the test again. Do this a total of 5 times. Ensure that the logs reflect 99 queries having successfully run.

### Reporting Results

In order to understand the results, we ask that all submitters fill out a form validating the architecture of their solution and append the validated results.

In order to validate your results, from the benchmark-kit file, execute:  
  
python process\_results.py

python process\_results.py

If successful, this will output a file “validated\_results.csv” that can be attached to the submission form.