



Dremio Software

# Initial Workload Management Settings

When Dremio is first installed, no guardrails are put in place out of the box to restrict how much memory any one queue or query in a queue can consume out of the total amount of memory available. This leaves Dremio open to potential out-of-memory issues if a user issues a large query that requires more memory than is available on the Dremio executors. In addition, the default queue concurrencies are a little high and could lead to memory exhaustion if many smaller queries (or up to 10 large ones) also need to collectively consume more memory than is available on the Dremio Executors.

This document provides recommendations for how to set up workload management queue and query memory limits and queue concurrency limits immediately after Dremio is installed to ensure Dremio will remain operational under memory pressure. The document covers the situations where Dremio is configured with and without engines.











## Configure Workload Management for New Installations

The following sections provide recommendations for how to set up some initial queue and query memory limits based on whether Dremio is or isn't configured to use multiple engines.

### No Engines or Single Engine Configuration

For Dremio 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.

The default queue settings for an out-of-the-box Dremio install are shown below; notice how no limits are set:

Queues						Average node memory: 16384.00 MB	Add New
Name *	CPU Priority	Concurrency Limit	Queue Memory Limit per Node	Job Memory Limit per Node	Engine Name	Actions	
High Cost Reflections	Background	1	-	-	-	 	
High Cost User Queries	Medium	10	-	-	-	 	
Low Cost Reflections	Background	10	-	-	-	 	
Low Cost User Queries	Medium	100	-	-	-	 	
UI Previews	Critical	100	-	-	-	 	

One important value to note is the *Average node memory*, which in this example is 16384 (or 16GB). This value tells us the maximum direct memory available on any executor. The value in this example is low. Consider that Dremio recommends 64GB or 128GB node sizes for Executors, which, after giving memory to the OS Kernel and heap memory, would leave 52GB or 112-116GB of direct memory per executor.

Regarding Queue Memory Limit per Node settings, the most important thing we can do when the initial installation is complete is to ensure every queue has a limit set. Even if you set the value on every queue to 90-95% of the *Average node memory*, this will significantly reduce the potential for the nodes to "lose communication" with Zookeeper and will prevent the executors from crashing if they encounter memory-intensive queries because a small memory will always be there to keep the communication with zookeeper.

As a rule of thumb, the total queue memory limit per node summed across the Low and High Cost User Queries queues should not exceed 120% of the *Average node memory* value. We

allow this to exceed 100% because it is reasonably unlikely that both queues will experience maximum memory usage simultaneously; therefore, we allow some degree of overlap.

The Low and High Cost Reflections queue memory limit should be set to at most the same values as the queue memory limit for the Low and High Cost User Queries queues. Reflections typically run far less frequently than other query types and are often triggered to run outside of regular working hours; therefore, we allow the sum of these values to exceed the *Average node memory* value.

However, suppose after making changes to conform to the rule of thumb above you have too many queries failing due to insufficient memory being available to a particular queue. In that case, it is safe to increase the amount of memory allocated to the queue where queries are failing; however, never go beyond 95% of the *Average node memory* on any one queue.

In terms of job memory limits, for high-cost user queries we want to allow Dremio to execute the biggest queries, therefore we will let the biggest job consume up to approximately 50-70% of the total memory available, depending on the *Average node memory* setting. Low-cost user queries typically consume far less memory and at most we would set a job memory limit of 50% of the queue memory limit or 5GB for one of these jobs, whichever is lower.

For UI Previews, Dremio recommends setting both the queue and job memory limit to the maximum memory allocated to a job in the High Cost User Queries queue. It is highly unlikely that these memory limits will ever get reached, but this provides guardrails in case they do.

Regarding concurrency limits, Dremio recommends the following initial concurrency settings, regardless of what the memory settings are:

Queue Name	Max Concurrency Limit
High Cost Reflections	1
High Cost User Queries	5
Low Cost Reflections	25
Low Cost User Queries	25
UI Previews	50

The following table summarizes the rules discussed above:

**Table 1: Rules for Zero Engines or Single Engine**

Queue Name	Max Concurrency Limit	Queue Memory Limit per Node	Job Memory Limit per Node
High Cost Reflections	2	0.75 x Average Node Memory	0.5 x Average Node Memory
High Cost User Queries	3	0.75 x Average Node Memory	0.5 x Average Node Memory
Low Cost Reflections	5	0.4375 x Average Node Memory	0.1875 x Average Node Memory*
Low Cost User Queries	20	0.4375 x Average Node Memory	0.1875 x Average Node Memory*
UI Previews	100	0.5 x Average Node Memory	0.5 x Average Node Memory

\* 0.1875 x Average Node Memory or 5GB, whichever is lower

The following sections provide examples of sensible memory settings based on various *Average node memory* values.

**Average Node Memory = 16384 (16GB)**

Queue Name	Max Concurrency Limit	Queue Memory Limit per Node	Job Memory Limit per Node
High Cost Reflections	2	12GB	8GB
High Cost User Queries	3	12GB	8GB
Low Cost Reflections	5	7GB	3GB
Low Cost User Queries	20	7GB	3GB
UI Previews	100	8GB	8GB

**Average Node Memory = 32768 (32GB)**

Queue Name	Max Concurrency Limit	Queue Memory Limit per Node	Job Memory Limit per Node
High Cost Reflections	2	24GB	16GB
High Cost User Queries	3	24GB	16GB
Low Cost Reflections	5	14GB	5GB
Low Cost User Queries	20	14GB	5GB
UI Previews	100	16GB	16GB

**Average Node Memory = 53248 (52GB)**

Queue Name	Max Concurrency Limit	Queue Memory Limit per Node	Job Memory Limit per Node
High Cost Reflections	2	40GB	25GB
High Cost User Queries	3	40GB	25GB
Low Cost Reflections	5	20GB	5GB
Low Cost User Queries	20	20GB	5GB
UI Previews	100	25GB	25GB

**Average Node Memory = 114688 (112GB)**

Queue Name	Max Concurrency Limit	Queue Memory Limit per Node	Job Memory Limit per Node
High Cost Reflections	2	84GB	56GB
High Cost User Queries	3	84GB	56GB
Low Cost Reflections	5	50GB	5GB
Low Cost User Queries	20	50GB	5GB
UI Previews	100	56GB	56GB

## Multi-Engine Configuration (AWSE or Kubernetes)

For Dremio installations on AWSE or Kubernetes where multiple engines are configured, we must understand 1) how many nodes are in the engine and 2) whether a dedicated engine will service reflections or whether all nodes across all engines will service the reflection refreshes. The example below assumes a dedicated engine **does NOT service** reflections.

We also assume a 1-to-1 mapping between a query queue and an engine.

The reason for the **-5GB** in these calculations is to ensure that when reflections run there is always a portion of each node in the engine that won't get utilized by the queries that run on that engine, which ensures there will always be some free memory available on each node to service reflections.

**Table 2: Rules for Multiple Engines**

Queue Name	Max Concurrency Limit	Queue Memory Limit per Node	Job Memory Limit per Node
High Cost Reflections	2	Average Node Memory / 2	Average Node Memory / 4
High Cost User Queries	3	(Average Node Memory - 5GB) / 2	(Average Node Memory - 5GB) / 4
Low Cost Reflections	5	Average Node Memory / 3	0.1875 x Average Node Memory*
Low Cost User Queries	20	(Average Node Memory - 5GB) / 3	0.1875 x Average Node Memory*
UI Previews	100	(Average Node Memory - 5GB) / 2	(Average Node Memory - 5GB) / 2

\* 0.1875 x Average Node Memory or 5GB, whichever is lower

Note: Average Node Memory needs to be calculated per engine

For cases where we provision a dedicated engine for reflections, we should follow Table 1 instead.

## Configure Queues Programmatically

A newly installed Dremio cluster comes with five queues with default configurations. Dremio provides APIs to update the memory configuration of queues. An approach to programmatically configure queues, based on cluster size, is provided below.

- Dremio needs the auth token to make any API call. You can obtain an auth token using the API /apiv2/login POST method. You can find more details here: [Dremio Login](#).

- Next, you should calculate the average direct memory by engine by querying Dremio system tables `sys.nodes` and `sys.memory`. You can submit the query via [API](#) or JDBC.
- Using the average direct memory, you can compute the following parameters for each queue:
  - Max Concurrency Limit
  - Queue Memory Limit per node
  - Job Memory Limit per node (based on your implementation, use Tables 1 & 2 above to compute these values from average direct memory)
- For each queue, Dremio generates a unique Queue ID. Get the current queue configuration using GET `/api/v3/wlm/queue`. You can find additional details in [Dremio All Queues](#).
- For each queue, update Dremio with the modified queue configuration using the API PUT `/api/v3/wlm/queue/{id}`. You can find additional details in [Update Queue](#).