Dremio Software

Dremio Monitoring in Kubernetes

Introduction

This guide provides a framework for implementing Prometheus monitoring within a Kubernetes environment, tailored explicitly for tracking the performance and resource utilization of Dremio, a powerful data analytics platform.

The aim is to give users a clear understanding of the critical aspects that need monitoring. Such monitoring is crucial to accurately assess the hardware capacity constraints of a Dremio deployment and identify the appropriate times for scaling up resources, including CPUs, memory, and disk performance. This document is both an instructional guide and a resource for optimizing Dremio's operational efficiency through effective monitoring strategies.

What are Prometheus and Grafana?

Prometheus and Grafana are open-source tools widely used for monitoring and observability in software systems. They are often used together because they complement each other's capabilities.

Prometheus

Prometheus is a monitoring system and time series database. It is particularly well-suited for monitoring the performance of distributed systems, including microservices architectures. Prometheus collects and stores metrics as time series data, which means the data is stored with timestamps to indicate when it was recorded. It offers a powerful query language, PromQL (Prometheus Query Language), to let users select and aggregate data. Prometheus's architecture is simple and does not require distributed storage; the main components include a time-series database, a data retrieval worker, and an alert manager.

Grafana

Grafana is an open-source platform for analytics and interactive visualization. While it can be used with various data sources, it is particularly popular for visualizing time series data, making it an ideal companion for Prometheus. Grafana provides a rich set of features for creating dashboards with multiple panels, each displaying data from sources like Prometheus in various visual formats like graphs, charts, tables, etc. It is highly customizable and supports alerts and notifications, which can be configured based on monitored data.

Together, Prometheus and Grafana are often used to enable real-time monitoring and alerting for IT infrastructure and application performance. Prometheus collects and stores the metrics, while Grafana is used to visualize those metrics and create insightful dashboards. This combination helps in identifying issues and performance bottlenecks in a system.

Initial Prometheus Setup

Create a Namespace for Prometheus

Create a Namespace in Kubernetes called `monitoring` :

```
$ kubectl create ns monitoring
```

Install Metrics Server

> ℹ️ NOTE
> This step is not required for Azure Kubernetes Service (AKS).

This metrics server will need to be installed to use the default scaling metrics of CPU and memory usage. Installation steps are as follows:

```
# Add the kubernetes-sigs repo
$ helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/

# Install Metrics Server
$ helm install  metrics-server metrics-server/metrics-server -n monitoring
```

Install Prometheus Stack

To have the required Custom Resource Definitions (CRD), we will need to install the Prometheus Stack using the steps below.

Before following the steps, create a copy of the Prometheus stack template file values_prometheus.hc.55.7.0.yml. Walk through the file and replace all values marked with a 'TODO'. This includes node selectors, storage classes and hostnames.

```
# Add the prometheus-community repo
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm update

# Install kube-prometheus-stack
$ helm upgrade --install prometheus prometheus-community/kube-prometheus-stack -n
monitoring --version 55.7.0 -f kube-prometheus-stack/values_prometheus.hc.55.7.0.yml
```

Configure Dremio for Prometheus Monitoring

Dremio needs to be configured so that it exposes the metrics to Prometheus. The guide assumes that Dremio's official Helm Charts have been used:

Configure Dremio

When Dremio is deployed using the official Helm Charts, the `values.yaml` file needs to be copied and modified.
The following needs to be added to `values.yaml`:

```
coordinator:
 extraStartParams: >-
   -Dservices.web-admin.port=9010
   -Dservices.web-admin.enabled=true
   -Dservices.web-admin.host=0.0.0.0

executor:
 extraStartParams: >-
   -Dservices.web-admin.port=9010
   -Dservices.web-admin.enabled=true
   -Dservices.web-admin.host=0.0.0.0
```

Additionally, it is required to patch the `templates/dremio-executor.yaml` file and the `templates/dremio-master.yaml` file to add the container port for monitoring:

```
        - containerPort: 9010
          name: prometheus
```

Configure Zookeeper

To enable Prometheus metrics for Zookeeper, the `templates/zookeeper.yaml` file needs to be patched:

```
# An additional env variable needs to be added (around line 110).
- name: ZOO_CFG_EXTRA
  value:
metricsProvider.className=org.apache.zookeeper.metrics.prometheus.PrometheusMetricsProvider
metricsProvider.httpPort=9010
# Web port needs to be added (around line 126)
- containerPort: 9010
  name: prometheus
```

Add Pod Monitors

To add the pod monitors so that Prometheus starts scraping the metrics from Dremio and Zookeeper, run the following command:

```
$ kubectl apply -n <dremio namespace> -f
https://raw.githubusercontent.com/chufe-dremio/dremio-prometheus-monitoring/main/specs/
dremio-pod-monitor.yaml
$ kubectl apply -n <dremio namespace> -f
https://raw.githubusercontent.com/chufe-dremio/dremio-prometheus-monitoring/main/specs/
zookeeper-pod-monitor.yaml
```

Install the Grafana Dashboard

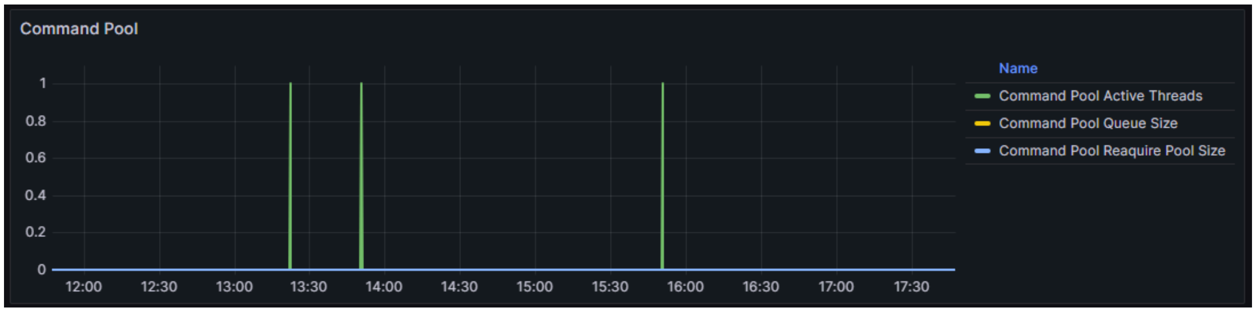
Login into Grafana, click the '+' icon at the top right and select "Import dashboard". Upload this `Dremio_Monitoring.json` file to Grafana.

What should we monitor?

Coordinator Command Pool

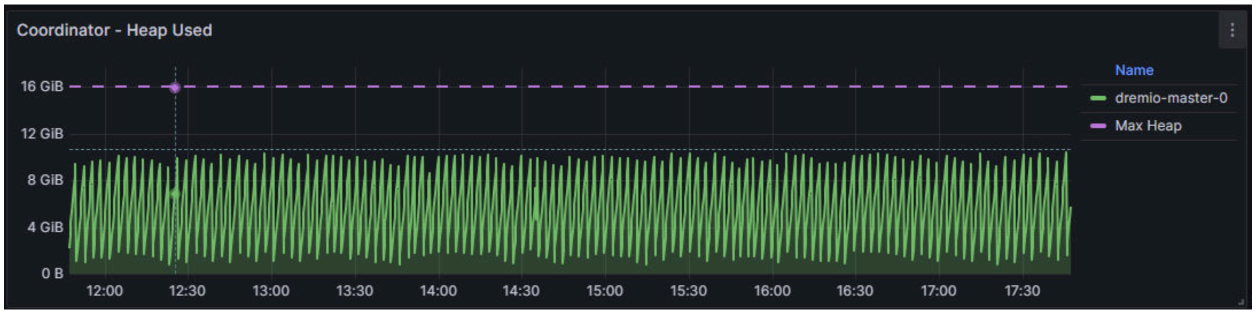
If you see “Command Pool Queue Size” going up, this shows jobs are waiting to be planned. It is an indicator because all planning threads are too busy to plan. For example, allocating 15 CPUs to the coordinator will use 15 minus 1 CPUs for planning. This means the coordinator can plan 14 queries in parallel.
If the queries need to wait until they get planned, this could indicate the following:

- Many inlined meta-data refreshes are running during planning.
- The number of CPUs needs to be higher to plan the amount of queries sent to the coordinator. Please check for high CPU utilization on the coordinator node.
- The disk is too slow, and the CPU waits for IO. Please check the disk saturation, disk IO wait time, and disk IO utilization in Kubernetes Nodes. If there is a high utilization, please add capacity for throughput and IOPS.



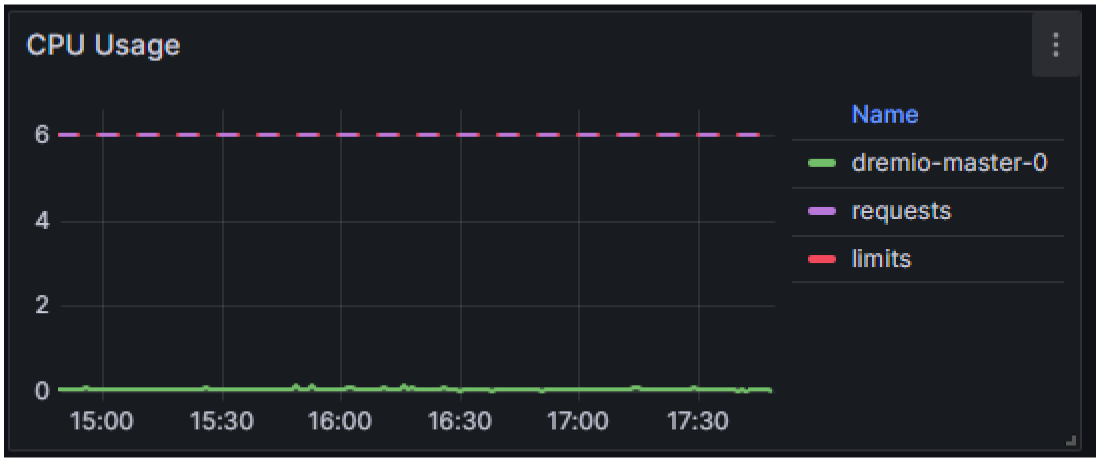
Coordinator Heap

Garbage collection times could increase if the heap allocation permanently hits its limit (Java setting `-Xmx`). Consider adding more heap memory and optimizing the GC settings so that memory is freed up much earlier.



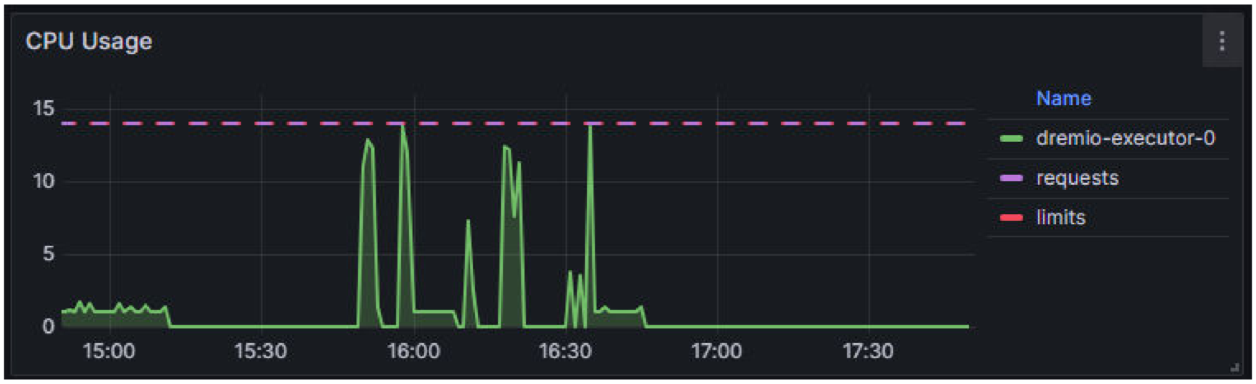
Coordinator CPU Usage

A high CPU utilization on the coordinator nodes indicates that the node is under-sized. It would most likely come together with an increased planning time and command pool wait queue (see above).



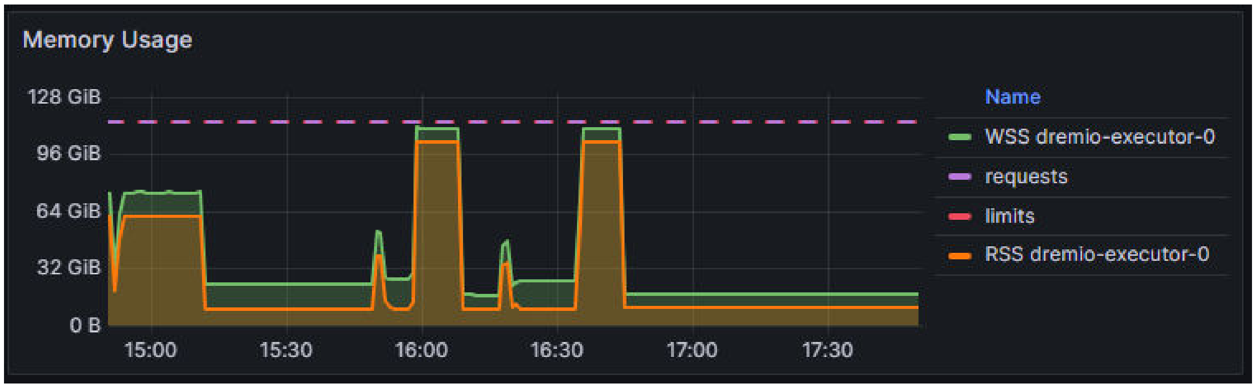
Executor CPU Usage

Your resources are fully utilized if you see a permanent high CPU utilization (not just peaks) across all executors. Adding more nodes or increasing the number of CPUs on the executor node, e.g. from 16 to 32, is recommended to improve the query speed.



Executor Memory Usage

If you constantly hit the configured requests and limits across all nodes, consider adding more capacity, e.g. nodes with more memory or more nodes. Ensure that you do not exceed the configured requested memory. If this happens, the heap and direct memory need to be reduced.



Node IO Utilization and IO Wait

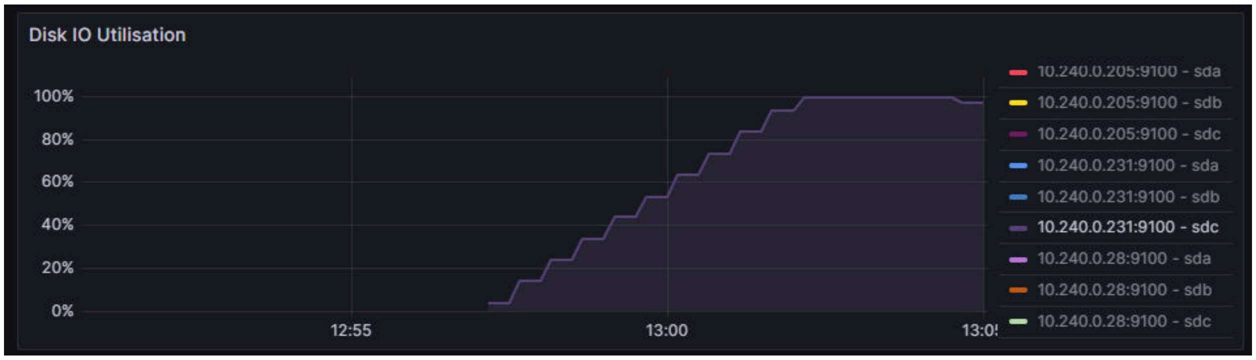
These metrics are on the node level and apply to the coordinator and executors.

Coordinators: If you see a high “Disk IO Utilisation” exceeding 70 % permanently, you must add faster disks because your disk slows down query planning. A high disk IO utilization likely comes with a high “IO wait time” and “Command Pool Wait Queue”. IO wait time means that the CPU waits for the disk or network.

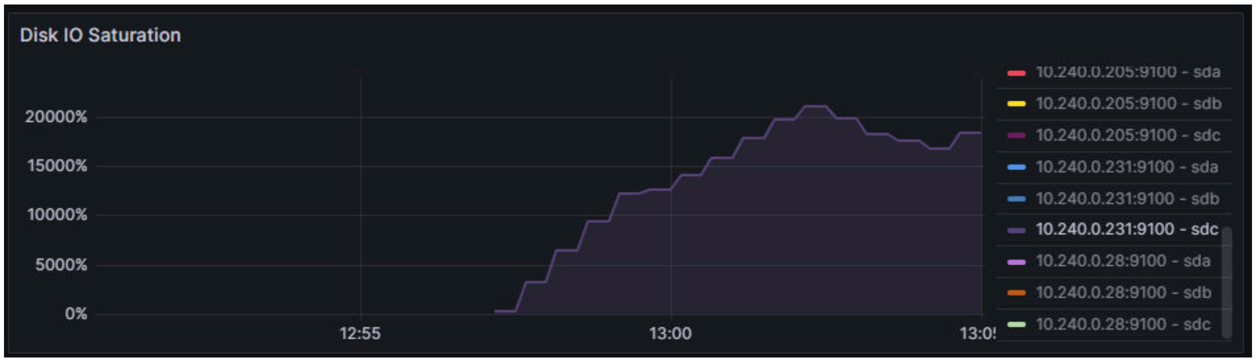
The coordinator is the central instance orchestrating the entire cluster. It should never get into an 'IO wait' state.

Executors: When you see a high disk utilization exceeding 90%, a disk saturation exceeding 400%, or an IO wait time on the executor, you should consider replacing the disks. It is less critical than the coordinator. Still, if you want to improve the query performance and utilize the CPU capacity, replacing the disks with more IOPS and throughput is recommended. The executor disks are used for spilling and cloud columnar cache.
The example below shows perfectly why a performant disk is essential. It is highly utilized and saturated with high IO wait, but on the other hand, the CPU idles with 1% utilization.

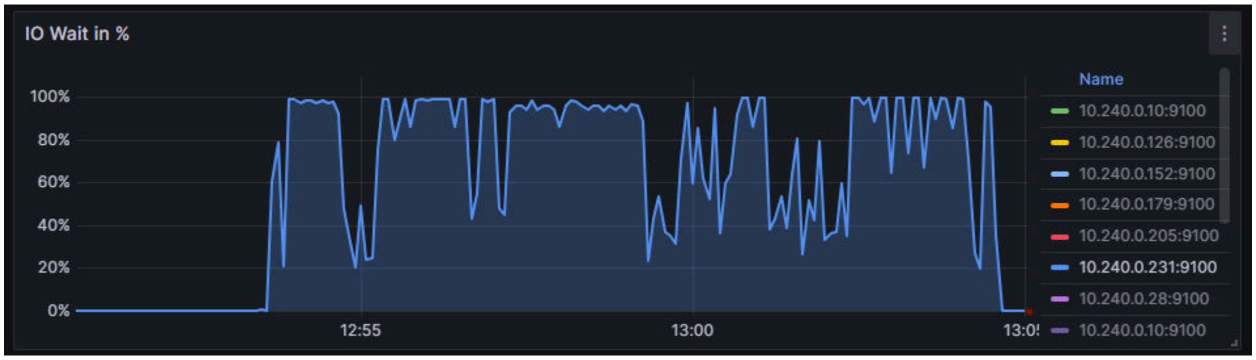
The disk is utilized 100%.



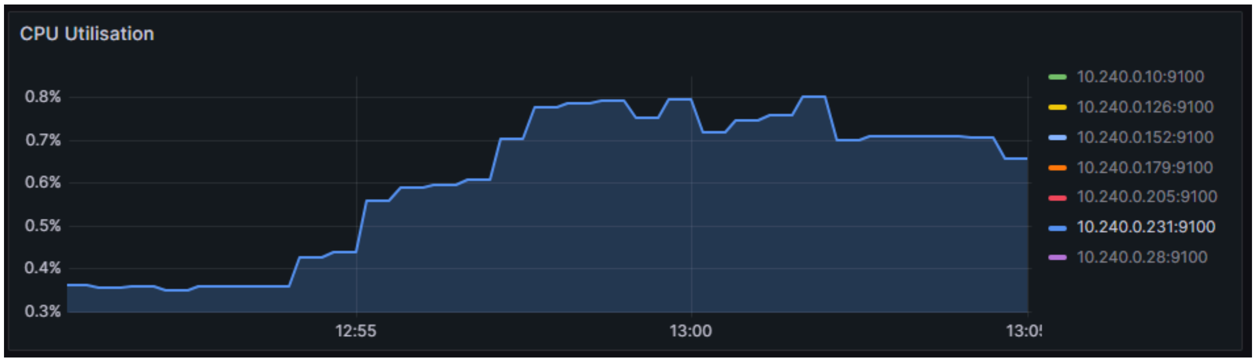
The disk is also saturated. The value should be at most 300 to 400%.



We can see that the CPU waits on IO ...



... and finally, the CPU idles because it waits on IO.



Summary

This document provides an overview of setting up Prometheus monitoring in a Kubernetes environment to monitor Dremio, a data analytics platform. The emphasis is on the necessity of monitoring for understanding and managing the hardware capacity limits of a Dremio deployment. Monitoring can determine when to scale resources such as CPUs, memory, or disk speed to maintain optimal performance.

The document details the configurations needed for Dremio and Zookeeper to ensure their metrics are accessible to Prometheus. It also guides the user through adding pod monitors for Dremio and Zookeeper, enabling Prometheus to scrape metrics effectively.

Key metrics that should be monitored are outlined. These include the Coordinator Command Pool, Coordinator Heap, Coordinator CPU Usage, Executor CPU Usage, Executor Memory Usage, and Node IO Utilization and IO Wait. These metrics are critical for assessing the performance and efficiency of the Dremio deployment, identifying potential bottlenecks, and making informed decisions about scaling and resource allocation.

In summary, the document serves as a comprehensive guide to setting up Prometheus monitoring in a Kubernetes environment for Dremio, highlighting the importance of specific metrics in understanding and optimizing the performance of the data analytics platform.