Dremio Cluster Monitoring

With Example Monitoring Solution

Preface

It's important to monitor Dremio Cluster to ensure overall cluster health and performance. Dremio Cluster provides a large set of metrics that can be utilized for a monitoring solution along with host-based metrics such as CPU, Memory, etc.

Dremio metrics that are important for monitoring can be collected via API, JDBC/ODBC and JMX.

A monitoring solution can utilize Prometheus, Grafana, AppDynamics and other tools. It's important to align the monitoring solution with existing monitoring infrastructure.

Provided here is an example implementation of monitoring to illustrate options and recommendations for monitoring. Example will monitor multiple Dremio clusters that have been deployed on the Hadoop clusters in YARN deployment mode. Example deploys Monitoring infrastructure and illustrates Alerting assuming infrastructure in place. The example is built on Prometheus and Grafana. Example can be expanded to monitor Dremio specific metrics on the similar technologies or to integrate with other existing solutions.

Example Dremio Monitoring Solution

Monitoring

1. Example runs multiple Dremio clusters in production and development environments in Hadoop (Yarn).
    a. Other environments will be similar, but sources and components may vary.
2. The example provides a dashboard with a summary status of all clusters.
3. Example monitoring solution provides information on both hot and stand-by coordinator nodes for each cluster.
4. Example also monitors the state of data sources.

Alerting

1. The example provides alerting capability based on customizable thresholds.
2. Alerts should be generated when:
    a. a cluster switches to a Stand-By coordinator,
    b. an Executor goes down,
    c. a data source becomes unavailable,
    d. available disk space becomes critically low,
    e. potentially, for other metrics that can be identified in the future.
3. Alerts should be delivered via email.

Actions on Events

1. In Dremio HA if an active coordinator goes down a stand-by coordinator is expected to take a role of the active coordinator and a load balancer shall start using the new active coordinator.
    a. HA is relevant to Standalone environments (not AWSE, AKS, EKS or any other Kubernetes deployment)

Solution Architecture

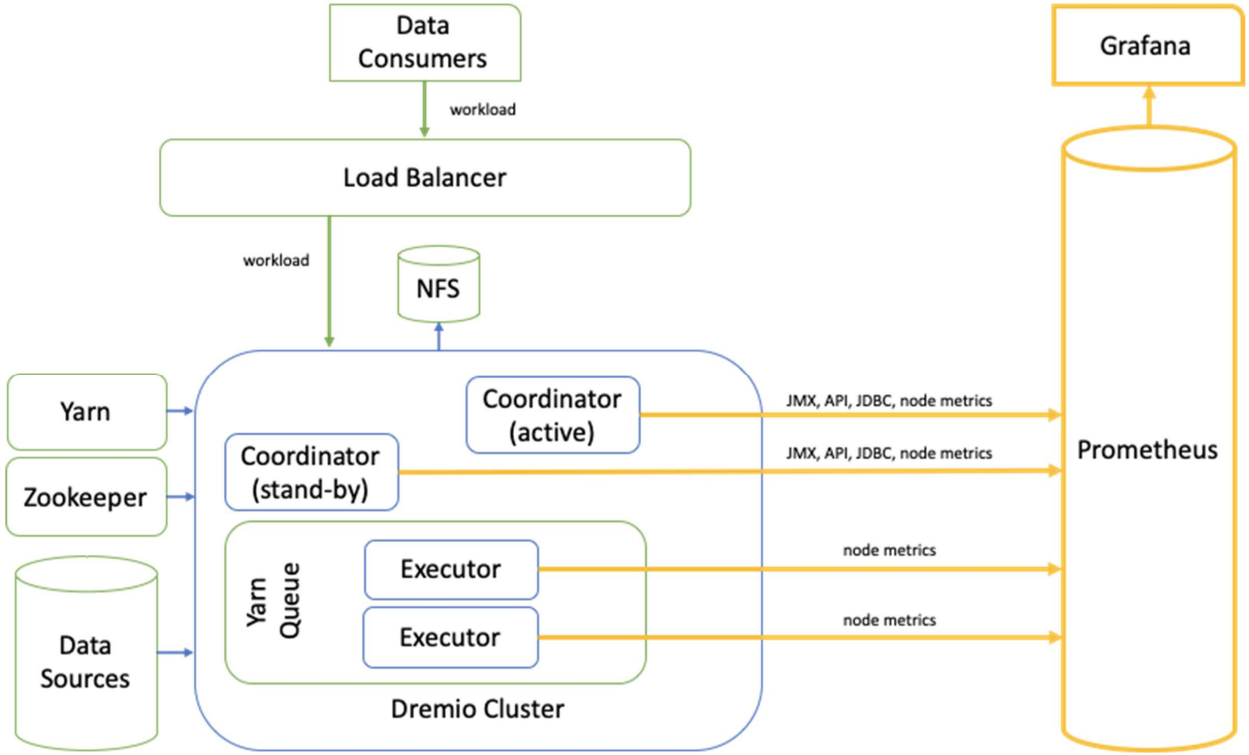
Existing Monitoring Infrastructure

This example assumes that:

- Customer has or will deploy a comprehensive monitoring and alerting infrastructure based on Prometheus and Grafana.
    - Setup examples for Prometheus and Grafana provided
- The infrastructure is already capable of collecting metrics via JDBC and API.
- The infrastructure requires additional tooling to collect metrics via JMX.
- Optional: additional related infrastructure metrics can be incorporated.

Implementation

The objective of this example monitoring solution is to ensure overall Dremio cluster health and performance. This example does not focus on query performance optimization. The following diagram depicts a high level view of a Dremio Cluster with dependencies and a monitoring solution deployed in Yarn. The implementation steps and documentation are presented later in the document.



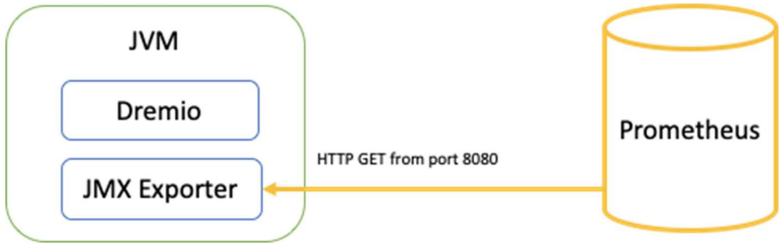
It's important to approach the monitoring solution with a holistic approach and monitor Dremio metrics, as well as metrics produced by related infrastructure, such as nodes, data sources, load balancer, NFS, Zookeeper, etc.

Dremio provides various metrics suitable for monitoring via JMX interface, Dremio API, and SQL. Metrics include coordinator-level metrics, executor-level metrics, cluster-level metrics.

Collecting JMX Metrics

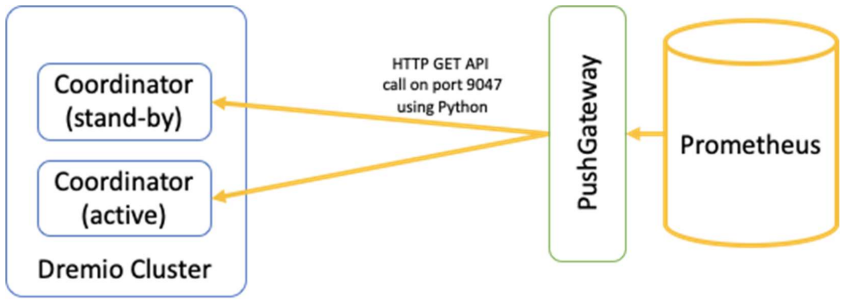
Process level metrics (coordinator, executor) can be collected via JMX. Dremio proposes to use JMX to Prometheus Exporter module to ingest metrics data into Prometheus database. Follow the JMX Exporter documentation to download and install it on a coordinator JVM.

The JMX Exporter can be configured locally on the JVM or remotely. Both configurations are described in the Exporter Documentation. The JVM local configuration is a preferred approach according to the Exported Documentation. The following diagram depicts this option (note the dependency arrow). The implementation steps and documentation are presented later in the document.



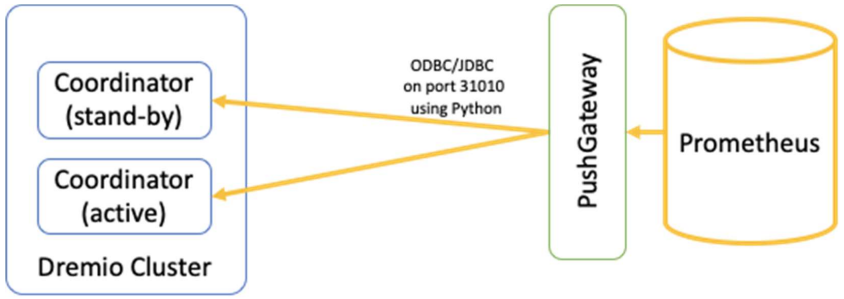
Collecting API Metrics

Metrics such as Coordinator status and Data Source state can be monitored via Dremio REST API. The recommended API metrics to monitor and how to implement in an overall monitoring solution are covered later in this document.



Collecting SQL Metrics

Cluster-level metrics such as direct memory utilization can be monitored via JDBC, ODBC, or API. It's recommended to use the Load Balancer endpoint for this purpose.



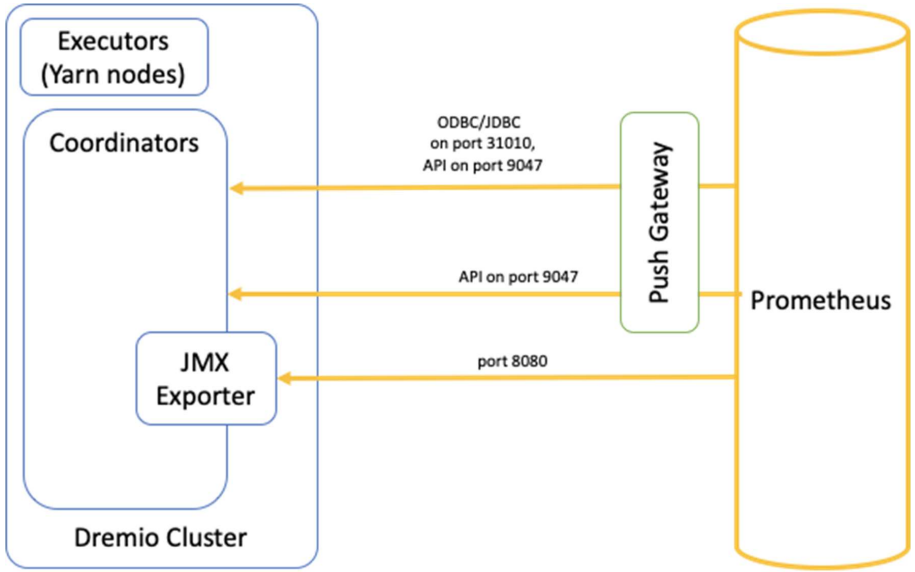
Collecting Executor Metrics

Collecting Dremio or Node-level metrics for an Executor is very complex due to random allocation processes by Yarn. Most of the Dremio Executor metrics are available in an aggregate shape via SQL and are sufficient for monitoring. Executor specific node metrics

can be important for troubleshooting. The node level metrics are constantly collected by CUSTOMER monitoring infrastructure for all Yarn nodes and can be pulled together with Dremio metrics into a troubleshooting dashboard when and how required.

Security Considerations

In the scope of the new capabilities put in place with this monitoring solution, there is communication between Prometheus, Load Balancer, and Dremio Coordinator nodes as depicted in the diagram below.



Please note:

- The diagrams depicts default ports that can be re-configured
- Ports 9047 and 31010 on the Load Balancer are expected to be open to Existing Monitoring Infrastructure as well as to any data consumer in the organization.
- Access to ports 9047 and 8080 on the Coordinator nodes can be limited to the Existing Monitoring Infrastructure only.

Commented [1]: Is this diagram missing the Load Balancer?

Commented [2R1]: I will need to modify the diagram after I can get the original graphic.

- API communication can be encrypted.
- JDBC/ODBC communication can be encrypted.
- Prometheus JMX Exporter does not support encryption on its service port 8080. However, it is possible to establish an ssl tunnel between Prometheus and Coordinator nodes if this encryption is necessary.
- Data exchange in the monitoring solution contains monitoring metrics only and does not contain any actual data.

Dremio Configuration Summary

Dremio recommends monitoring the following metrics:

- Heap memory usage and GC frequency
- Direct memory usage
- Used lightweight threads

Heap Memory Usage and GC Frequency

Dremio uses heap memory for planning, coordination, UI serving, query management, connection management, and some types of record reading etc. type of tasks. Heap memory is expected to be higher in coordinator nodes in high concurrency deployments. When observed together, continued high garbage collection (GC) frequency and high heap memory usage would indicate an undersized cluster/node.
Heap usage can be tracked via memory.heap.usage.
Garbage collection can be tracked by monitoring cumulative counts and times over time: gc.PS-MarkSweep.count, gc.PS-MarkSweep.time, gc.PS-Scavenge.count, gc.PS-Scavenge.time. Please note that garbage collection logging is already enabled by default on all Dremio nodes.

Direct Memory Usage

Dremio uses direct memory for query execution tasks — directly affecting performance and concurrency. Dremio also uses direct memory for RPC communication between executor and coordinator nodes, as well as communicating with the end

users. Direct memory is expected to be used heavily during query execution on the executor nodes. Continued high direct memory usage would indicate the cluster/node approaching it's capacity.
Direct memory allocated/used by the execution engine can be tracked via dremio.memory.direct_current. Total direct memory given to the JVM can be tracked via dremio.memory.jvm_direct_current.

Used Lightweight Threads

Depending on how much work Dremio is doing, the system might be aggressively parallel. Sometimes this can mean that Dremio is designed to allow for even a single query to use as many cores as available to the process. The number of threads running on each executor node describes the total amount of parallelization Dremio is using. This number may be substantially more than the number of cores as Dremio is very effective at scheduling between different threads.
If this metric goes more than 10-20 times the number of logical cores, you are probably slowing down individual queries due to contention. You can better understand the impact of contention on a per query basis by looking at the query profile and viewing "Wait Time" value under the "Thread Overview" section per phase. This describes how long each lightweight thread has work available to do but is not scheduled due to CPU contention. Note that this thread count is not directly correlated to kernel threads.
Lightweight threads can be tracked via dremio.exec.work.running_fragments.

Enabling Node JVM Metrics

Dremio enables node monitoring by default. To manually enable node JVM monitoring, add the following properties to the dremio-env file on each Dremio node in your deployment.

```
DREMIO_JAVA_SERVER_EXTRA_OPTS="
-Dcom.sun.management.jmxremote.port=<monitoring port>
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false"
```

Warning
In production environments, Dremio strongly recommends using both SSL client certificates to authenticate the client host and password authentication for user management. See Monitoring and Management Using JMX Technology (Java 7), Out-of-the-Box Monitoring and Management Properties section, for more configuration information or Monitoring and Management Using JMX Technology (Java 8)

Enabling Dremio Metrics on JMX

By default, Dremio does not publish its metrics on JMX. This functionality needs to be enabled by creating DREMIO_HOME/conf/dremio-telemetry.yaml file with the following code:

```
# Dremio telemetry configuration file
# Control if this file should automatically be reloaded, and at which interval.
auto-reload:
  enabled: True
  period: 30
  unit: SECONDS
metrics:
  - name: reporter_jmx
    reporter:
      type: jmx
```

Telemetry Configuration

Here's the current configure Dremio telemetry (metrics and trace) reporting.

If a file named dremio-telemetry.yaml is present in the application classpath, the content of the file will be parsed and monitoring will be configured according to the instructions of the file.

Format of the file is as follows:

```
---
# Dremio telemetry configuration file

# REQUIRED: Control if this file should automatically be reloaded, and at which interval.
auto-reload
  enabled: [ True | False ]
  period: 30
  unit: SECONDS

# OPTIONAL: Metrics configuration
```

```
metrics:
  -
    name: reporter1
    comment: >
      Comment regarding the reporter
    reporter:
      type: reporter_type
      key1: value1
      key2: value2
      [...]
    includes:
      - pattern1
      - pattern2
      - [...]
    excludes:
      - pattern3
      - pattern4
      - [...]
  -
    name: reporter2
    [...]

# OPTIONAL: Tracing configuration
tracing:
  type: in-memory
  sampling:
    key1: value1
    key2: value2
  otherkey: othervalue
  [...]
[...]
```

Metrics Configuration:

Monitoring can be configured using a YAML file.

If a file named dremio-metrics.yaml is present in the application classpath, the content of the file will be parsed and monitoring will be configured according to the instructions of the file.

Format of the file is as follows:

```
-
  name: reporter1
  comment: >
    Comment regarding the reporter
  reporter:
    type: reporter_type
    key1: value1
    key2: value2
    [...]
  includes:
    - pattern1
    - pattern2
    - [...]
  excludes:
    - pattern3
    - pattern4
    - [...]
-
  name: reporter2
  [...]
[...]
```

Each reporter might have its own configuration, see below for details

Metrics inclusions/exclusions

For each reporter, a set of metrics can be included/excluded. User can specify a list of regular expressions to match against the metric name. The rules are the following:

- a metric is reported if its name is included **AND** is not excluded

- if includes is empty or not specified, default is to include all metrics
- If excludes is empty or not specified, default is to not exclude any metric

Reporters

JMX

This reporter exports metrics as JMX bean.

Configuration

name	description	default
rate	the time unit to use for rate metrics	seconds
duration	the time unit to use for duration metrics	milliseconds

SLF4J

This reporter exports metrics as log statement to a specific logger

Configuration

name	description	default
logger	the logger name	metrics
rate	the time unit to use for rate metrics	seconds
duration	the time unit to use for duration metrics	milliseconds
intervalMS	the reporting interval in milliseconds	2 minutes

If you are using the SLF4J logger, be sure to update the conf/logback.xml file to enable your logger.

```
<logger name="YOUR_LOGGER_NAME_HERE">
  <level value="{dremio.log.level: -info}"/>
  <appender-ref ref="console"/>
</logger>
```

Examples

Local dremio-telemetry.yaml:

```
# Dremio telemetry configuration file

# Control if this file should automatically be reloaded, and at which interval.
auto-reload:
  enabled: True
  period: 30
  unit: SECONDS


# Tracing configuration
tracing:
  type: jaeger
  serviceName: dremio
  samplerType: const
  samplerParam: 1
  logSpans: true
```

Stackdriver config:

```
# Dremio telemetry configuration file

# Control if this file should automatically be reloaded, and at which interval.
auto-reload:
  enabled: True
  period: 30
  unit: SECONDS
```

```
# Metrics configuration
metrics:
  -
    name: reporter1
    reporter:
      type: stackdriver
    includes: []
    excludes: []

# Tracing configuration
tracing:
  type: stackdriver
```

Recommended Monitoring Metrics

To ensure the cluster operates smoothly and to proactively identify issues before they have a broader impact on the workload, cluster monitoring is needed. The table below shows the metrics that need to be collected, how they are obtained, how frequently they need to be collected. Dremio does not provide alerting infrastructure. The metrics obtained below can be obtained via Python, Shell scripting and many JMX monitors and integrated with existing enterprise monitoring tools.

The “How Obtained” column in the table below shows the API that exposes the metric as well as the method that will be used to obtain it. The different API’s and how they can be called are:

SQL - SQL commands can be run through ODBC, JDBC through any generic ODBC/JDBC tool or through REST interfaces. The only limitation with running SQL against the REST interface is that the API call doesn’t return the result set of the query submitted. It instead returns the query ID of the query that was submitted.
REST - Dremio coordinator exposes the REST API on port 9047. How to connect, authenticate and submit requests can be found here.
JMX - (Default behavior) on the cluster. JMX monitoring requires enablement as described in section above.

General Monitoring Recommendations

- GC - # of GC per Min
    - Look for GC spikes or long running
- Heap memory at constant state
    - Normal Dremio usage expects a “saw-tooth” pattern
        - Planning spike with decline during execution
    - Look for Heap memory remaining constant in the 90-95% range
- Reflections - number of existing verses failed reflections
    - Use reflections_* data to create alerts

Detailed Monitoring Recommendations

Dremio recommend metrics 1 - 5 are implemented as alertable metrics. These metrics would raise an alarm and the on duty operator gets alerted as they indicate cluster health issues that need immediate attention. All metrics below should be implemented as trending alerts that capture these metrics periodically and display them on internal monitoring dashboards.

The column “Frequency” is environment specific and needs to be tuned as needed. In addition, Dremio Customer Support or Dremio Professional Services might recommend additional metrics that need to be monitored depending on the workload. All metrics need to be collected by a username that’s defined as an administrator on the cluster.

Metric #	Description	How obtained (API, method)	Expected values	Frequency
1	Coordinator Status. Used to check if the coordinator node is up and responding to requests.	**REST** GET /apiv2/server_status	“OK”. Anything other than “OK” or a slow response indicates that coordinator is either down or unhealthy	Every 10 seconds

2	Cluster's ability to run a distributed query. This test confirms connectivity with metadata, security credentials of sources the VDS/PDS is on and connectivity with a random executor the coordinator choses to executor the LIMIT 1 query.	**SQL** SELECT * from <PDS|VDS> LIMIT 1	One row returned with a consistent response time.	Every 30 seconds
3	ZooKeeper connectivity . ZooKeeper is a critical component that ensures both high availability works and cluster nodes can talk to each other. Monitor for ZooKeeper disconnects and suspended messages to proactively identify connectivity issues	**POSIX** grep -i "SUSPENDED" server.log	No lines that match	Every 30 seconds
4	Connectivity errors between executors	**POSIX** grep -i "FabricChannelClosed" server.log	No lines that match	Every 5 minutes

5	Catalog DB free space. Catalog DB must have free space to create reflections, update profiles, run jobs. If space available is critically low, raise a high severity alarm	**POSIX** df -h <directory where rocks DB is mounted> #directory pointed to by paths.local on coordinator	<50 MB free space or >99% utilized	Every 30 seconds
6	Catalog DB free space. Sometimes rocks DB does compactions which can take 2x space temporarily. So keeping free space at >2x used space is recommended	**POSIX** df -h <directory where rocks DB is mounted> #directory pointed to by paths.local on coordinator	Utilization should be less than 40% of available space.	Every 24 hours
7	RPC Failures. Monitor Communication failures between processes (coordinator <-> executors, executors <-> executors) that have occurred in the last 15 minutes	**JMX** rpc.failure_15m	0 - no communication failures in the last 15 minutes	Every 30 seconds

8	CPU and memory usage of executors	**SQL** select name, hostname, cpu, memory, status from sys.nodes where is_executor = true	Depends on workload. Initial recommendation is to have 2 different alert levels. Warning for Any executor whose memory is above 70% for more than 10 minutes or Any executor whose CPU is above 80% for more than 10 minutes Critical for Any executor whose memory is above 85% for more than 20 minutes or Any executor whose memory is above 90% for more than 5 minutes or Any executor whose CPU is above 90% for more than 10 minutes	Every 30 seconds

9	Executor health status.	**SQL** select status, count(*) from sys.nodes where is_executor = true group by status /* sys.nodes only contains Executor nodes that are successfully registered with Zookeeper, it does not report on nodes that exist but not yet in operation.*/	All nodes with status=green. Raise a CRITICAL ALERT Even if one node has a status that's not green OR the number of rows returned by this query!=number of nodes this cluster has.	Every 30 seconds
10	Queries waiting in ENQUEUED state	**JMX** jobs.queue.<queuename>.waiting	Depends on workload. Recommend starting with ">5 queries in waiting " and tune as needed	Every 30 seconds
11	Check data lake and external source status	**REST** GET /api/v3/source	All sources return "good"	Every 30 seconds
12	Failed reflections. Reflections should not fail. They should always run to completion. If reflections are not up to date, end users may not get the SLA's and results they	**REST** GET /api/v3/reflection/summary	No reflections in "FAILED" State. Processing of the JSON will be required.	Every 30 seconds

	want. If a reflection fails, generate a critical alert			
13	Long metadata refresh times	**POSIX** grep -i “refresh details in “ metadata_refresh.log |cut -d’ ‘ -f12	If resulting number>300 seconds (adjust as needed) generate an alert to look into the source and why it’s taking so long	Every 5 minutes
14	Background metadata refresh job skipping metadata refresh	**POSIX** grep -i “skipping metadata refresh” metadata_refresh.log	No skipped metadata refreshes.	Every 5 minutes
15	Catalog DB space used	**POSIX** du -h <rocks DB directory> #directory pointed to by paths.local on coordinator	<150 GB. If >150 GB, run dremio-admin clean and rerun space test, if space is still high open a support case	Every 24 hours
16	Long running jobs (queries + reflections)	**JMX** jobs.long_running	Workload dependent. Start with 5 and increase/decrease as needed	Every 4 hours

Commented [3]: For these POSIX commands, how are you intending the users to issue the commands and monitor them in their monitoring tools, these look like commands that will be run directly on the nodes so how will the results of the command be fed back to the monitoring too? I think a little more detail on how the customer is supposed to set that up would be beneficial.

Commented [4R3]: Will need to follow up on these items related to remote execution of commands on remote nodes. These were a collection of different customer implementations and the "art of the possible"

17	Coordinator heap memory utilization as a % of total available	**JMX** memory.heap.usage	80%. Coordinator's JVM monitor automatically kills queries when heap utilization is > 85%. So 80% is a good threshold for warning bells to start ringing.	Every 30 seconds
18	Executor heap memory usage	**SQL** SELECT heap_current*100/heap_max from sys.memory	Heap usage is <90% of configured heap max	Every 30 seconds
19	Executor direct memory usage	**SQL** SELECT (direct_current+jvm_direct_current)*100/direct_max from sys.memory	Direct usage is <90% of configured direct max	Every 30 seconds
20	Light weight threads utilization	**JMX** dremio.exec.work.running_fragments	Workload dependent. Recommend starting with this metric being <20x number of cores. If it trends towards this high water mark, work with Dremio Professional Services to analyze this and consider	Every 30 seconds

			decreasing parallelism.	
21	Coordinator garbage collection counts	**JMX** gc.PS-Scavenge.count gc.PS-MarkSweep.count	<2 minor or major GC's reported per second	Every 30 seconds
22	Failed jobs	**JMX** jobs.failed_15m	0. Some jobs fail due to syntax errors and wrong queries. Start with a low number and adjust as needed depending on how many failed jobs are submitted.	Every 30 seconds
23	Reflections currently refreshing	**JMX** reflections.refreshing	Workload dependent. Depends on how many reflections are configured to run using WLM rules. Start with 1 and increase as needed.	Every 15 minutes
24	Space available in Dremio install folder - $DREMIO_HOME/ on coordinators and executors	**POSIX** df -h $DREMIO_HOME on all cluster nodes	>10% free space on $DREMIO_HOME on all nodes	Every 1 minute

25	Space available in Dremio spill space on executors	**S3** **<df -h paths.dist.spill>**	Workload dependent. Start with <50%, Or <50GB per node utilized and tune as needed	Every 15 minutes

Accessing Node Metrics

You can access your Dremio node metrics using jconsole or another Java Agent that collects JMX metrics. (see Prometheus and Grafana examples below)
Jconsole example: https://docs.dremio.com/advanced-administration/monitoring/#accessing-node-metrics

Enabling JMX Metrics for Prometheus

JMX Exporter

Source: https://github.com/prometheus/jmx_exporter

JMX to Prometheus exporter: a collector that can be configurable to scrape and expose mBeans of a JMX target.

This exporter is intended to be run as a Java Agent, exposing a HTTP server and serving metrics of the local JVM. It can run as an independent HTTP server and scrape remote JMX targets, but this has various disadvantages, such as being harder to configure and being unable to expose process metrics (e.g., memory and CPU usage). Running the exporter as a Java Agent is thus strongly encouraged.

Prometheus Configuration

This section assumes that Prometheus is already available in CUSTOMER monitoring infrastructure. For any information regarding Prometheus installation, please refer to Appendix A or Prometheus Documentation.

Scrape Configuration

We highly recommend using cluster names to uniquely identify each scrape job. Clusters in the dashboards inherit their names from this configuration. Here's a sample configuration. Also, it's highly recommended to put a load balancer in front of the coordinators and use that endpoint in the configuration, since active coordinator can go down, in which case, standby takes over serving cluster load.

```
global:
  scrape_interval: 5s

scrape_configs:
  - job_name: "mycluster1"
    static_configs:
    - targets: ["<LB HOST>:<LB PORT>"]

scrape_configs:
  - job_name: "mycluster2"
    static_configs:
    - targets: ["<LB HOST>:<LB PORT>"]
```

With this configuration, Prometheus will now be scraping JMX metrics only. For JDBC and API based metrics , we recommend using Push Gateway or some other mechanism to push the metrics to Prometheus. It's recommended to use the same approach for pushing custom metrics as well. We recommend running the monitoring infrastructure behind firewalls and access JMX interface via a server inside the firewall or via a tunnel.

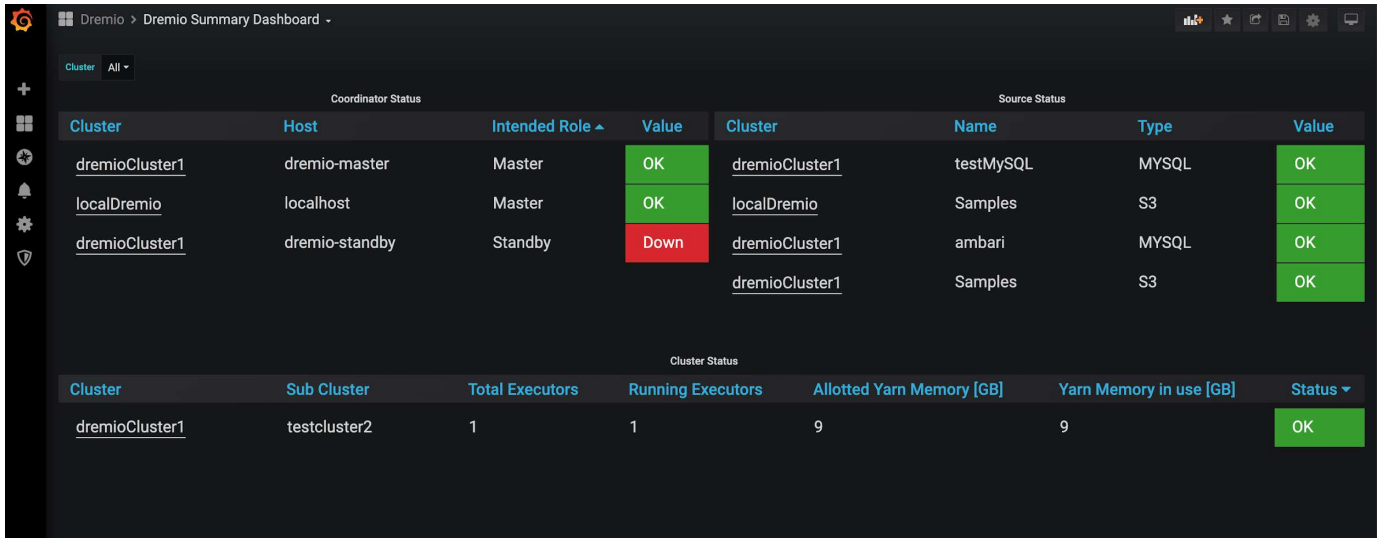
Grafana Dashboards

This section assumes that Grafana is already available in CUSTOMER monitoring infrastructure. For any information regarding Grafana installation, please refer to Grafana Documentation.

This monitoring solution is implemented with two separate dashboards, one to provide a bird's eye view on the overall status of all clusters and another to provide deeper insight into an individual cluster.

Summary Dashboard

The Summary dashboard provides a high level status of all clusters. Following screenshot shows how a summary dashboard will look like. There are 3 sections: Coordinator Status, Cluster Status and Source Status. It also has a drop-down menu, which allows one to select one or multiple clusters.



Coordinator status panel shows both master and standby coordinators for all configured clusters. Status is color coded (OK, OK, DOWN), based on the cluster availability.Status OK means coordinator is running with the right role (Master/Standby) on the host. In the screenshot above it shows that the Master coordinator is running on the dremio-master node while it's running in standby mode on the dremio-standby node, as they were configured to. Status OK means coordinator is running with the wrong role on the host (Eg: If coordinator was running in Master role on dremio-standby node, it would show this status).

Dashboard also displays master/standby coordinator nodes for each cluster (IP or hostname in DNS). This metric is ingested into Prometheus via Dremio health check API (/apiv2/server_status). Note, a cluster specific dashboard is accessible from this dashboard by following a link on the cluster name.

Cluster status panel provides status for all the sub clusters for the cluster. It shows the number of expected executors, actual executor count, memory allocation on YARN and in use by executors. Finally, it provides a color coded status for each sub cluster.

Source status panel provides status for each of the sources on the cluster.

Here's an explanation of metrics available on the dashboard.

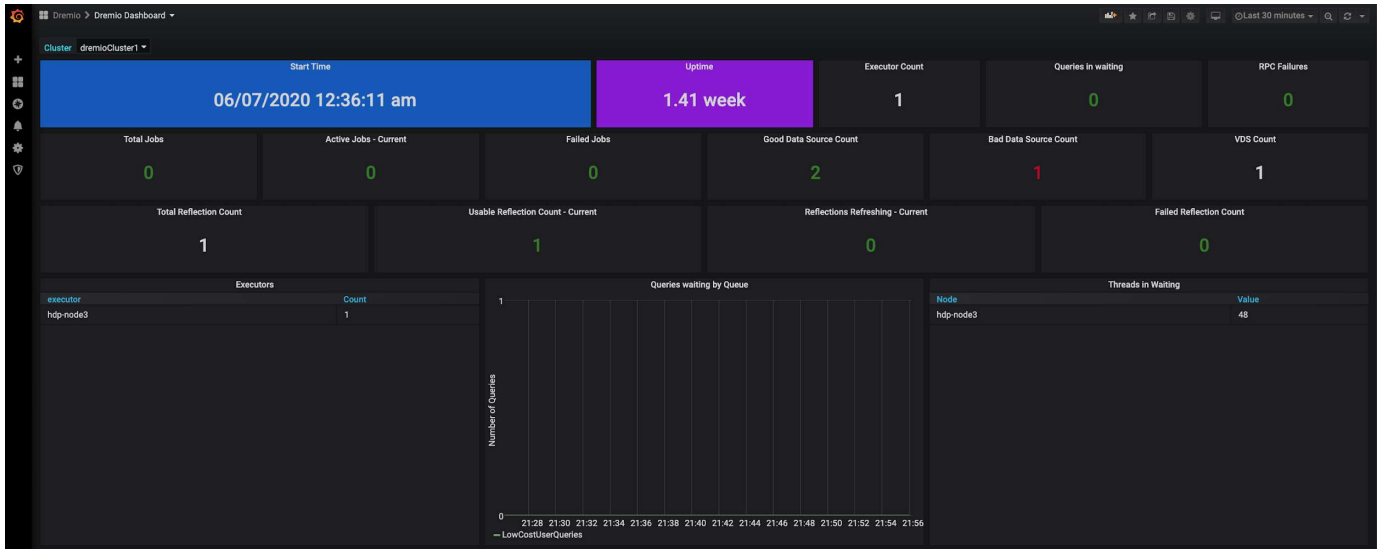
Dashboard Metric	Metric Name	Source	Description	Alert Recommendation
Coordinator Status	dremio_api_coordinator_up	API	Status of the Dremio Coordinator node (Master/Standby). This metric is derived from /apiv2/server_status API. The value depends on the coordinator's intended role (Master/Standby) and its actual status (Master/Standby/Down). Values are: 0 - down, 1 - not in intended role, 2 - in intended role.	**Warning**: When the Standby coordinator goes down and there's just one coordinator available. **Critical**: When there are no coordinators available.
Source Status	dremio_api_source_status	API	Status of each source for the cluster. This metric is derived from /api/v3/catalog/ API.	**Critical**: When the source goes down

Cluster Status	dremio_api_total_executors_Value dremio_api_current_executors_Value dremio_api_cluster_allocated_memory_Value dremio_api_cluster_used_memory_Value dremio_api_cluster_up	API	Status of each sub cluster.	**Critical**: When the cluster goes down.

Dremio will provide the JSON file for the dashboard that CUSTOMER can import and customize if required.

Cluster Dashboard

The Cluster Dashboard will list all the key metrics (provided with explanation in the next section) for a given cluster. This dashboard can be accessed from the summary dashboard, by clicking on the cluster name. Following pictures show panels available on the dashboard. Please note that choose the time frame from the drop down menu on the top right corner of the dashboard. Choosing a random time frame from graph panels may lead to wrong values in single stat (counters) panels since they don't understand non-standard time frames well.


Dremio > Dremio Dashboard
Last 30 minutes
Cluster dremioCluster1
Start Time
06/07/2020 12:36:11 am
Uptime
1.41 week
Executor Count
1
Queries in waiting
0
RPC Failures
0
Total Jobs
0
Active Jobs - Current
0
Failed Jobs
0
Good Data Source Count
2
Bad Data Source Count
1
VDS Count
1
Total Reflection Count
1
Usable Reflection Count - Current
1
Reflections Refreshing - Current
0
Failed Reflection Count
0
Executors
executor
Count
hdp-node3
1
Queries waiting by Queue
Number of Queries
1
0
21:28 21:30 21:32 21:34 21:36 21:38 21:40 21:42 21:44 21:46 21:48 21:50 21:52 21:54 21:56
LowCostUserQueries
Threads in Waiting
Node
Value
hdp-node3
48


Heap memory Usage - Coordinator
4.0 GB
3.5 GB
3.0 GB
2.5 GB
2.0 GB
1.5 GB
1.0 GB
500 MB
0 B
21:28 21:30 21:32 21:34 21:36 21:38 21:40 21:42 21:44 21:46 21:48 21:50 21:52 21:54 21:56
dremio-master: Used
dremio-master: Max
Disk Usage - Coordinators
Host
Mounted On
Capacity
Available
Used %
dremio-standby
/
7.99 GiB
3.93 GiB
50.80%
dremio-master
/
499.99 GiB
479.48 GiB
4.10%
Garbage Collection Time - Coordinator
4.10 ms
4.05 ms
4.00 ms
3.95 ms
3.90 ms
21:28 21:30 21:32 21:34 21:36 21:38 21:40 21:42 21:44 21:46 21:48 21:50 21:52 21:54 21:56
dremio-master:PS MarkSweep
dremio-master:PS Scavenge
Memory Usage - Coordinators
15.83 GiB
15.37 GiB
14.90 GiB
14.44 GiB
13.97 GiB
13.50 GiB
13.04 GiB
12.57 GiB
21:28 21:30 21:32 21:34 21:36 21:38 21:40 21:42 21:44 21:46 21:48 21:50 21:52 21:54 21:56
dremio-master:Total
dremio-standby:Total
dremio-master: Available
dremio-standby: Available
Garbage Collection Counts - Coordinator
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
21:28 21:30 21:32 21:34 21:36 21:38 21:40 21:42 21:44 21:46 21:48 21:50 21:52 21:54 21:56
dremio-master:PS MarkSweep
dremio-master:PS Scavenge
CPU Usage - Coordinators
10.00%
8.00%
6.00%
4.00%
2.00%
0%
21:28 21:30 21:32 21:34 21:36 21:38 21:40 21:42 21:44 21:46 21:48 21:50 21:52 21:54 21:56
max
avg
current
dremio-master
8.918%
2.108%
1.150%
dremio-standby
0.350%
0.141%
0.150%

Here's an explanation of metrics available on the dashboard.

Dashboard Metric	Metric Name	Source	Description	Alert Recommendation
Start Time	process_start_time_seconds	JMX	Start time of the coordinator process.	None
Uptime	process_start_time_seconds	JMX	Uptime for the cluster. Standard Prometheus metric	None
Executor Count	dremio_sql_executors_Value	SQL	Total number of executors for the cluster.	**Warning**: 50% of expected count **Critical**: 25% of expected count
Queries in Waiting	metrics_jobs_command_pool_queue_size	JMX	Number of queries in waiting state	**Warning**: 10 **Critical**: 20

	_Value			
VDS Count	dremio_sql_vds_count_Value	SQL	Total number of VDS for the cluster	None
Good Data Source Count	dremio_api_source_status_Value	API	Total number of Data Sources for the cluster that are in good state	None
Bad Data Source Count	dremio_api_source_status_Value	API	Total number of Data Sources for the cluster that are in bad state	None
RPC failures	metrics_rpc_failure_15m_Value	JMX	Communication failures between processes (coordinator <-> executors, executors <-> executors)	**Warning**: Based on cluster load **Critical**: Based on cluster load
Total Reflection Count	metrics_reflections_active_Value metrics_reflections_unknown_Value	JMX	Total number of reflections in the cluster. This included usable and unusable reflections.	None
Usable Reflection Count - Current	metrics_reflections_active_Value	JMX	Number of reflections that are usable. Each reflection adds to the workload on the cluster, due to refresh requirements. If the number keeps increasing, Administrators will need to work with users to understand the usage pattern and perhaps extend existing reflections to satisfy additional workloads.	**Warning**: Custom **Critical**: Custom
Reflections Refreshing - Current	metrics_reflections_refreshing_Value		Number of reflections that are refreshing right now.	**Warning**: Custom **Critical**: Custom
Failed Reflection Count	metrics_reflections_failed_Value	JMX	Number of reflections that failed refresh. A high number could indicate an	**Warning**: Custom **Critical**: Custom

			undersized cluster.	
Total Jobs	metrics_jobs_active_15m_Value	JMX	Cumulative number of queries in the last 15 min.	None
Active Jobs - Current	metrics_jobs_active_Value	JMX	Count of currently running queries. A high number is not a cause for concern, unless it's seen in conjunction with a high number for failed jobs.	**Warning**: Custom **Critical**: Custom
Failed Jobs	metrics_jobs_failed_15m_Value	JMX	Cumulative number of jobs that have failed in the last 15 min. A high number could indicate an overworked cluster.	**Warning**: Custom **Critical**: Custom
Executors	dremio_sql_executors_Value	SQL	List of executors running currently. It shows the host and the count of executors for that host.	None
Queries waiting by Queue	metrics_jobs_queue_<QUEUE>_waiting_Value	JMX	Number of queries waiting in a queue	**Warning**: 10 **Critical**: 20
Threads in Waiting	dremio_sql_threads_waiting_Value	SQL	Number of threads in waiting state per executor node	None
Heap Memory Usage - Coordinator	metrics_memory_heap_used_Value metrics_memory_heap_max_Value	JMX	Dremio Coordinator uses heap memory for planning, coordination, UI serving, query management, connection management, and some types of record reading etc. type of tasks. When observed together, continued high garbage collection (GC) frequency and high heap memory usage would	**Warning**: 80% of allocated capacity **Critical**: 90% of allocated capacity

			indicate an undersized cluster/node.	
Heap Memory Usage - Executors	dremio_sql_executor_heap_current_Value dremio_sql_executor_heap_current_Value		Heap memory in use by executors. This is provided from informational point only.	None
Direct Memory Usage - Executors	metrics_dremio_executor_direct_current_Value metrics_dremio_executor_direct_max_Value	SQL	Direct memory is expected to be used heavily during query execution on the executor nodes. Continued high direct memory usage would indicate the cluster/node approaching it's capacity.	**Warning**: 80% of allocated capacity **Critical**: 90% of allocated capacity
Garbage Collection Counts - Coordinator	jvm_gc_collection_seconds_count	JMX	Rate (per second) at which minor (PS Scavenge) and major (PS MarkSweep) garbage collections are happening on the coordinator node. Minor GC events are supposed to be high compared to major GC events.	**Warning**: 2 **Critical**: 5
Garbage Collection Time - Coordinator	jvm_gc_collection_seconds_sum	JMX	Amount of time each GC event takes. This number should be in seconds, at max.	**Warning**: 5 **Critical**: 30
Disk Usage - Coordinators	node_filesystem_size_bytes node_filesystem_avail_bytes	Node	This is provided by Node JVM Exporter. Customize this to monitor disk space in spill directories.	**Warning**: 80% **Critical**: 90%

	node_filesystem_free_bytes			
Memory Usage - Coordinators	node_memory_MemTotal_bytes node_memory_MemAvailable_bytes	Node	This is provided by Node JVM Exporter.	**Warning**: 80% **Critical**: 90%
CPU Usage - Coordinators	node_cpu_seconds_total	Node	This is provided by Node JVM Exporter.	**Warning**: 80% **Critical**: 90%
Network Usage - Coordinators	node_network_receive_bytes_total node_network_transmit_bytes_total	Node	This is provided by Node JVM Exporter.	**Custom**
Disk IO - Coordinators	node_disk_read_bytes_total node_disk_written_bytes_total	Node	This is provided by Node JVM Exporter.	**Custom**

Example source files can be downloaded from here: https://github.com/kevin-dremio/dremio_monitoring

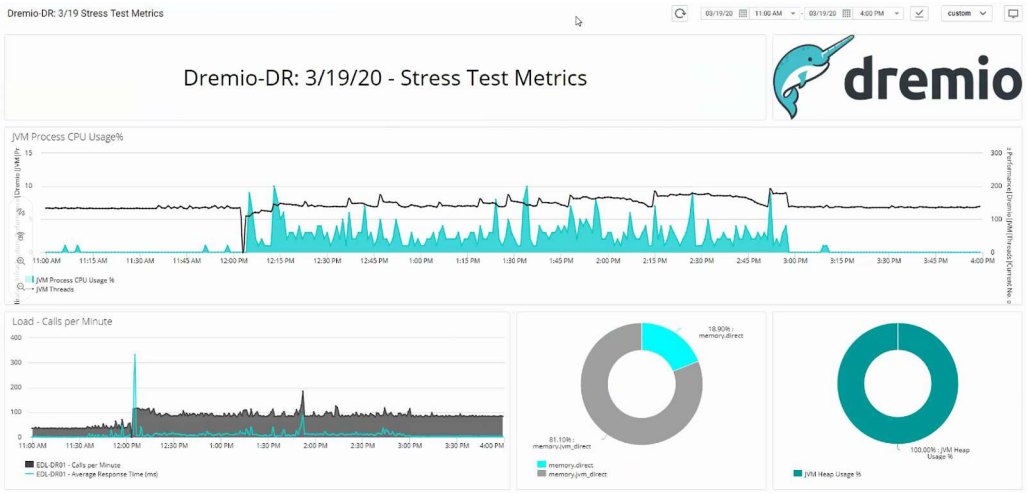
JMX Enablement for App Dynamics

Modify the dremio-env file in /{$dremio_home}/conf folder (ex: /opt/dremio/conf) to include:

```
DREMIO_JAVA_SERVER_EXTRA_OPTS='
-javaagent:/opt/appdynamics/ver4.5.19.29348/javaagent.jar
-Dappdynamics.controller.hostName=app_host_site.saas.appdynamics.com
-Dappdynamics.controller.port=443
-Dappdynamics.controller.ssl.enabled=true
-Dappdynamics.agent.applicationName=EDL-DR01
-Dappdynamics.agent.tierName=Dremio
-Dappdynamics.agent.nodeName=DR01'
```

Please note the single quotes encapsulating the string

Example AppDynamics Dashboard



SQL Queries

Get list of Executors and count of executors running on each of them

```
SELECT hostname, count(*)
FROM sys.nodes GROUP BY hostname
```

Get current heap memory usage on the coordinator node

```
SELECT heap_current
FROM sys.memory
```

Get max heap memory configured on the coordinator node

```
SELECT heap_max
FROM sys.memory
```

Get current direct memory in use on an executor node

```
SELECT direct_current
FROM sys.memory
```

Get direct memory allocated to an executor node

```
SELECT direct_max
FROM sys.memory
```

Get number of threads in waiting state on an executor node

```
SELECT hostname, count(*)
FROM sys.threads
WHERE threadState in ('WAITING')
```

Get total number of VDS

```
SELECT count(*)
FROM  information_schema."TABLES"
WHERE table_type = 'VIEW' and table_schema NOT LIKE '@%'
```

Appendices

Prometheus and Grafana Installation

Prometheus Push Gateway

Pushgateway is the mechanism that we use to ingest metrics from API/JDBC sources into Prometheus.

Download the Package

Download the latest version of Push Gateway from the downloads page. For example, run the following command to download the latest version.

```
wget https://github.com/prometheus/pushgateway/releases/download/v1.2.0/pushgateway-1.2.0.linux-amd64.tar.gz
```

Configure Push Gateway

Push Gateway is not a resource intensive application. We recommend running it on the same host as Prometheus.

- Create a user
  `sudo useradd --no-create-home --shell /bin/false prometheus`

- Create needed directories
  sudo mkdir -p /var/local/pushgateway
  sudo chown prometheus:prometheus /var/local/pushgateway

- Extract the tar and copy required files
  tar xvzf pushgateway-1.2.0.linux-amd64.tar.gz
  cd pushgateway-1.2.0.linux-amd64
  sudo cp pushgateway /usr/local/bin
  sudo chown prometheus:prometheus /usr/local/bin/pushgateway

- Create a systemd service file.
  $ cat /etc/systemd/system/pushgateway.service
  [Unit]
  Description=Prometheus Pushgateway

  [Service]
  Type=Simple
  User=prometheus
  Group=prometheus
  ExecStart=/usr/local/bin/pushgateway --web.listen-address=:9091 --web.telemetry-path=/metrics --
  persistence.file=/var/local/pushgateway/pushgateway.store --web.enable-admin-api
  ExecReload=/bin/kill -HUP $MAINPID
  KillMode=process
  Restart=on-failure

  [Install]
  WantedBy=multi-user.target

- Start the service

sudo systemctl daemon-reload
sudo systemctl restart pushgateway
systemctl status pushgateway - Should show that process is running

- Verify that Push gateway is working by issuing curl <host>:9091/metrics on the command line. You should see metrics for the push gateway process.

Note that we are enabling the Admin API for Push Gateway. This needs to be used by Admins to maintain metrics, when the source of metric disappears. This is well documented in Push Gateway documentation. It's possible to automate the cleanup process though.

Here's one example of when the cleanup needs to be run. Let's say we have 2 sub clusters called mycluster1 and mycluster2 running on Dremio. At some point, mycluster2 gets deleted (or renamed). However, push gateway will continue to expose the metric for the sub cluster. We will then need to invoke the Admin API to delete the metric like so:

curl -X DELETE http://<push gateway>:9091/metrics/job/dremioCluster1/cluster/mycluster2

Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit. We recommend running Prometheus along with Push Gateway and Grafana on the same host. A 2CPU, 8GB host should be good enough to run all three.

Download the Package

Download the latest version of Prometheus from the downloads page. For example, run the following command to download the latest version.

wget https://github.com/prometheus/prometheus/releases/download/v2.18.0-rc.0/prometheus-2.18.0-rc.0.linux-amd64.tar.gz

Configure Prometheus

- Create a user
  sudo useradd --no-create-home --shell /bin/false prometheus

- Create needed directories
  sudo mkdir -p /etc/prometheus /var/local/prometheus
  sudo chown prometheus:prometheus /etc/prometheus /var/local/prometheus

- Extract the tar and copy required files
  tar xvzf prometheus-2.18.0-rc.0.linux-amd64.tar.gz
  cd prometheus-2.18.0-rc.0.linux-amd64
  sudo cp prometheus /usr/local/bin
  sudo cp promtool /usr/local/bin
  sudo chown prometheus:prometheus /usr/local/bin/prometheus
  sudo chown prometheus:prometheus /usr/local/bin/promtool

  sudo cp -r consoles /etc/prometheus
  sudo cp -r console_libraries /etc/prometheus
  sudo chown -R prometheus:prometheus /etc/prometheus/consoles
  sudo chown -R prometheus:prometheus /etc/prometheus/console_libraries

- Configure scrape settings
  sudo touch /etc/prometheus/prometheus.yml
  sudo chown prometheus:prometheus /etc/prometheus/prometheus.yml

  Here's a sample configuration. Please note that it's a best practice to set the job_name to the name of the cluster that's getting monitored. Also note that we're scraping metrics from the push gateway.

  ```
  global:
    scrape_interval: 5s

  scrape_configs:
    - job_name: "prometheus"
      static_configs:
      - targets: ["localhost:9090"]
  ```

```
  - job_name: "mycluster1"
    static_configs:
    - targets: ["<DREMIO COORDINATOR HOST1>:8080"]
    - targets: ["<DREMIO COORDINATOR HOST2>:8080"]
  - job_name: "push"
    honor_labels: true
    static_configs:
    - targets: ["<PUSH GATEWAY HOST>:9091"]
```

- Create a systemd service file. Location where data is stored should have enough space (with faster disks, ideally).

```
$  cat /etc/systemd/system/prometheus.service
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target
[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
    --storage.tsdb.path /var/local/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries
[Install]
WantedBy=multi-user.target
```

- Start the service

```
sudo systemctl daemon-reload
sudo systemctl restart prometheus
```

systemctl status prometheus - Should show that process is running

- Verify that Prometheus Web UI is up and running, by visiting `<host>:9090` on the browser.

Grafana

Grafana is open source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics no matter where they are stored. We highly recommend to run this on the same host as Prometheus.

Configure the YUM repo

Grafana is available in a YUM repository. Create a repository file `/etc/yum.repos.d/grafana.repo` with following contents.

```
[grafana]
name=grafana
baseurl=https://packages.grafana.com/oss/rpm
repo_gpgcheck=1
enabled=1
gpgcheck=1
gpgkey=https://packages.grafana.com/gpg.key
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
```

Install Grafana

Run `sudo yum install grafana` to install grafana.

Start the service

There's no need to configure grafana. The package installs systemd script. Run following commands to start the service and verify that it's running.

```
sudo systemctl daemon-reload
sudo systemctl start grafana-server
sudo systemctl status grafana-server
```

Open <HOST>:3000 on the browser and you should see grafana Web UI.

Prometheus Python Client

We use this to push metrics to Prometheus via Push Gateway. This is needed only on the host that runs the metric collection scripts.

Install

This is installed via pip. Collection scripts make use of python3, so install it using the following command.

```
pip3 install prometheus_client
```

Pushing metrics into Prometheus via shell script

This is done using Prometheus Push Gateway. Simply use a command-line HTTP tool like curl.

Push a single sample into the group identified by {job="some_job",instance="some_instance"}:

```
echo "metrics_dremio_up{role="Master"} 1.0" | curl --data-binary @-
http://pushgateway.CUSTOMER.com:9091/metrics/job/some_job/instance/some_instance
```

Since no type information has been provided above, metrics_dremio_up will be of type untyped.

Push something more complex into the group identified by {job="some_job",instance="some_instance"}:

```
cat <<EOF | curl --data-binary @- http://pushgateway.CUSTOMER.com:9091/metrics/job/some_job/instance/some_instance
  # TYPE metrics_dremio_executor_direct_max_Value gauge
  # HELP metrics_dremio_executor_direct_max_Value Maximum direct memory allocated to an executor.
  metrics_dremio_executor_direct_max_Value 8589934592.0
```

```
 # TYPE metrics_dremio_executor_direct_current_Value gauge
 # HELP metrics_dremio_executor_direct_current_Value Direct memory in use by an executor.
 metrics_dremio_executor_heap_max_Value 408934592.0
EOF
```

Note how type information and help strings are provided. Those lines are optional, but strongly encouraged for anything more complex.

Complete list of Dremio JMX Metrics

Most current Dremio JMX metrics can be found in the Dremio documentation at: https://docs.dremio.com/advanced-administration/monitoring/#monitoring-jmx-metrics
The complete list of Dremio metrics available in the dremio coordinator is available on http://<coordinator node>:8080.

Dremio Log Files

The location of dremio log files is configured during Dremio installation. The default location is /var/log/dremio/ and it contains server.log file that is a rolling log file. The logging configuration including rolling configuration can be found in /etc/dremio/logback.xml or /opt/dremio/conf/logback.xml.

There are other log files in /var/log/dremio folder, such as garbage collection logging, access logging, query history, etc.