

Metric Store v1.4 Documentation

Metric Store 1.4

You can find the most up-to-date technical documentation on the VMware website at:

<https://docs.vmware.com/>

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Copyright © 2023 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

Contents

Metric Store Release Notes	5
v1.4.5	5
v1.4.4	5
v1.4.3	5
Known Issues	5
v1.4.2	5
Known Issues	5
v1.4.1	6
Known Issues	6
v1.4.0	6
Known Issues	6
v0.3.0 (Beta)	6
v0.2.0 (Beta)	6
Features	6
Known Issues	6
Metric Store	8
About Metric Store	8
More Information on Open Source Metric Store	8
Product Snapshot	8
Limitations	8
Installing Metric Store	10
Prerequisites	10
Install Metric Store	10
Create a Network	10
Configure the Metric Store Tile	10
Metric Store Resources	11
Verifying installation of Metric Store	11
Querying the Metric Store	12
Overview	12
Prerequisites	12
Using Metric Store	12
Authorization & Authentication	12

For Metric Store admins	12
For Tanzu Application Service developers	12
Standard Tanzu Application Service application metrics	13
Querying via Prometheus-Compatible HTTP Endpoints	13
Notes on PromQL	13
GET /api/v1/query	13
Response Body	14
GET /api/v1/query_range	14
Response Body	15
GET /api/v1/series	15
Response Body	15
GET /api/v1/labels	16
Request	16
Response Body	16
GET /api/v1/label//values	16
Request	16
Response Body	16
Request	17
Response Body	17
Monitoring Metric Store	18
Overview	18
Key Scaling Indicators	18
Metric Emitted by Metric Store	18
Nozzle Job Metrics	18
CF Auth Proxy Job Metrics	19
Metric Store Job Metrics	19
Metric Store Remote Metrics	20

Metric Store Release Notes

This topic describes the changes in this minor release of Metric Store.

v1.4.5

Release Date: February 10, 2021

- Use Ops Manager as the CA for all certificates to facilitate easier certificate rotation on Metric Store nodes.

v1.4.4

Release Date: September 29, 2020

- Fixed bug where Metric Store was intermittently dropping points between nodes
- Added `instance_id` label to `http_*` metrics
- Removed empty errand `metric-store-smoke-tests`

v1.4.3

Release Date: August 17, 2020

- This release fixes a memory leak that would cause the Metric Store to slowly run out of memory due to not cleaning up closed network connection go routines.
- Updated debug and metrics endpoints to be on different ports.
- Updated Golang to 1.13.12 and Prometheus to 2.19.1.

Known Issues

- Found in 1.4.2, some customers are experience points being dropped when metrics are transferred between nodes.

v1.4.2

Release Date: April 28, 2020

- Resolves issue with `http_total` metric to include correct `node_index` label value.

Known Issues

- Found in 1.4.2, some customers are experience points being dropped when metrics are transferred between nodes.

v1.4.1

Release Date: April 27, 2020

- Updated the `opsmanager_syslog` property to true to allow operators to egress logs to a syslog destination.

Known Issues

This release has a bug in the `http_total` metric when the Metric Store is running on multiple nodes. Metric Store incorrectly tracks a label `node_index` value as 0 as opposed to the true node index. Resolved release 1.4.2.

v1.4.0

Release Date: March 31, 2020

- Replication is now Availability Zone aware. **Note** for those upgrading from Beta versions this will cause a loss in metrics.
- Updated our CA properties to help customers rotate both their Ops Manager and CredHub stored certificates.
- Updated read sample limit to 20,000,000, previously set to 1,000,000.
- Removed PromScraper to inject Metric Store metrics into Firehose.
 - ✦ Scraping Metric Store Prometheus endpoint via mTLS.

Known Issues

This release has a bug in the `http_total` metric when the Metric Store is running on multiple nodes. Metric Store incorrectly tracks a label `node_index` value as 0 as opposed to the true node index. Resolved release 1.4.2.

v0.3.0 (Beta)

Release Date: November 19, 2019

- Introduced `http_total` metrics and `http_duration_seconds_*` histogram for request latency.
- Fixed `metric_storage_duration_days` to report 0 when there is no data.
- Fixed degraded query performance on large points (>500,000).

v0.2.0 (Beta)

Release Date: July 9, 2019

Features

[See Overview](#)

Known Issues

This release has the following known issues.

Metric Store

About Metric Store

Metric Store provides a persistent storage layer for metrics sent through the Loggregator subsystem. It is multi-tenant aware (the auth proxy ensures that you only have access to metrics from your apps), easy to query (it is compatible with the [PromQL](#)), and has a powerful storage engine (the [InfluxDB storage engine](#) has built-in compression and a memory-efficient series index).

- **Multi-tenant aware:** Application Developers have access to their own metrics.
- **Ubiquitous Query Language:** Metric Store is compatible with PromQL and Prometheus API Query endpoints.
- **Powerful Storage Engine:** The InfluxDB storage engine has built-in compression and a memory-efficient series index.

Metric Store offers multi-node deployments, data replication, hinted handoff, and load balancing in a highly-available configuration.



Note: The only officially supported use case for the Metric Store product is as the backing Time Series Database for [App Metrics v2.0](#) and [TAS v2.9+ Apps Manager](#). Use cases outside of these bounds have no guarantee of continued support or availability.

More Information on Open Source Metric Store

- [Open Source Release Blog post](#)
- [Open Source README](#)

Product Snapshot

The following table provides version and version-support information about Metric Store.

Element	Details
Version	v1.4.5
Release date	February 10, 2021
Compatible Ops Manager version(s)	v2.6.x, v2.7.x, v2.8.x
Compatible Pivotal Application Service version(s)	v2.6.x, v2.7.x, v2.8.x
IaaS support	AWS, Azure, GCP, and vSphere

Limitations

Metric Store has the following limitations:

- Horizontally scaling Metric Store currently results in data loss. Vertical scaling is recommended at this time.
- Platforms are ever changing and we cannot guarantee metric stability. Using Metric Store through products like [App Metrics](#) or [Apps Manager](#) will be supported use cases. Using the Metric Store manually through endpoints, the user will need to be aware of changes through Platform upgrades.

Installing Metric Store

This topic tells you how to install Metric Store.

Prerequisites

For a list of compatible Ops Manager and VMware Tanzu Application Service versions, see [Product Snapshot](#).

Install Metric Store

1. Download the product file from [VMware Tanzu Network](#).
2. Navigate to the Ops Manager Installation Dashboard and click **Import a Product** to upload the product file.
3. Under the **Import a Product** button, click **+** next to the version number of Metric Store. This adds the tile to your staging area.

Create a Network

If you do not want to use an existing network for Metric Store, you can create a new network by following these steps:

1. Click the **BOSH Director** tile on the Ops Manager Installation Dashboard.
2. Navigate to **Create Networks** and click **Add Network**.
3. In the **Name** field, enter a name for the service network.
4. Enter the network details.
5. Click **Save**.

Configure the Metric Store Tile

To configure the Metric Store tile, perform the following steps:

1. Click the **Metric Store** tile on the Ops Manager Installation Dashboard.
2. Navigate to **Assign AZs and Networks**.
3. Select an Availability Zone (AZ) for placing singleton jobs.
4. Select one or more AZs for balancing other jobs. To create a highly available environment, VMware recommends selecting multiple AZs.
5. Select **Network** for installing Metric Store.
6. Click **Save**.

Metric Store Resources

You can configure Metric Store resources in the **Resource Config** section of the Metric Store tile. By default, Metric Store configuration is pre-selected for all deployments.

Resource	Instances	CPU	RAM	Ephemeral Disk	Persistent Disk	Static IP	Dynamic IP
Metric Store	3	4	32 GB	30 GB	250 GB	0	1

Verifying installation of Metric Store

Please see the Authorization and Authentication section of the topic [Using Metric Store](#).

Querying the Metric Store

This topic tells you how to configure and use Metric Store.

Overview

Metric Store implements the Prometheus Query Language for querying metrics for which you have access.

Prerequisites

- Completed [installation](#) of the Metric Store Tile.
- [curl](#)
- [CF Auth Token](#)

Using Metric Store

Authorization & Authentication

When querying the API via HTTPS, each request must have the [Authorization](#) header set with a UAA provided token.

For Metric Store admins

As a Metric Store admin, you will have access to all recorded metrics (platform and application) and can execute Label and Series queries. You will need to add either [doppler.firehose](#) or [logs.admin](#) scope to your admin account.

You then should be able to run the following query:

```
cf login
curl -vvv -H "Authorization: $(cf oauth-token)" -G https://metric-store.SYSTEM-DOMAIN/
api/v1/label/source_id/values
```

The status code will be 200 is everything is working properly.

For Tanzu Application Service developers

As a Tanzu Application Service developer, you can query metrics for applications you have access to. When querying with PromQL, you must specify the [source_id](#) label with the application [guid](#) as the label value.

For example:

```
curl -H "Authorization: $(cf oauth-token)" -G "https://metric-store.SYSTEM-DOMAIN/api/v1/query" \
  --data-urlencode "query=cpu{source_id=\"$(cf app --guid your-app-name)\"}"
```

Standard Tanzu Application Service application metrics

Metric Name	Description	PromQL Example
cpu	CPU usage percentage	<code>avg(cpu{source_id="\$app_guid"})</code>
memory	Memory usage in bytes	<code>avg(memory{source_id="\$app_guid"})</code>
memory_quota	Memory Quota allocated in bytes	<code>memory{source_id="\$app_guid"} / memory_quota{source_id="\$app_guid"}</code>
disk	Disk usage in bytes	<code>avg(disk{source_id="\$app_guid"}) / 1024 / 1024 / 1024 # in gigabytes</code>
disk_quota	Disk quota allocated in bytes	<code>disk{source_id="\$app_guid"} / disk_quota{source_id="\$app_guid"}</code>
http_total	HTTP request counts includes status	<code>sum(rate(http_total{source_id="\$app_guid"})) by (status)</code>
http_duration_seconds_*	HTTP request request latencies bucketed by duration	<code>histogram_quantile(0.95, sum(rate(http_duration_seconds_bucket[5m])) by (le))</code>

Querying via Prometheus-Compatible HTTP Endpoints

Notes on PromQL

The ultimate goal of these endpoints is to create a fully-compliant, Prometheus-compatible interface. This should allow tools such as Grafana to work directly with Metric Store without any additional translation.

A valid PromQL metric name consists of the character [a-Z][0-9] and underscore. Names can begin with [a-Z] or underscore. Names cannot begin with [0-9]. As a measure to work with existing metrics that do not comply with the above format a conversion process takes place when matching on metric names. Any character that is not in the set of valid characters is converted to an underscore. The metric is not changed in the store.

e.g., to match on a metric name `http.latency` use the name `http_latency` as a search term.

GET /api/v1/query

Issues a PromQL instant query against Metric Store data. You can read more detail in the Prometheus documentation [here](#).

Query Parameters:

- **query** is a [Prometheus expression query string](#).
- **time** is an optional UNIX timestamp in seconds or RFC3339. (e.g. `date -d '24 hours ago' +%s`).



Note: Admins (`doppler.firehose` or `logs.admin`) are permitted to use this query without specifying a `source_id` in the query parameter. Non-admins cannot use regex matchers on `source_id`.

```
$ curl -H "Authorization: $(cf oauth-token)" -G "https://metric-store.SYSTEM-DOMAIN/api/v1/query" --data-urlencode 'query=metric_name_0{source_id="source_id_0"}'
```

Response Body

```
{
  "status": "success",
  "data": {
    "resultType": "vector",
    "result": [
      {
        "metric": {"__name__": "metric_name"},
        "value": [ "<timestamp>", "<value>" ]
      },
      "... "
    ]
  }
}
```

GET /api/v1/query_range

Issues a PromQL range query against Metric Store data. You can read more detail in the Prometheus documentation [here](#).

Query Parameters:

- **query** is a [Prometheus expression query string](#).
- **start** is a UNIX timestamp in seconds or RFC3339. (e.g. `date -d '24 hours ago' +%s`). Start time is inclusive. `[start..end)`
- **end** is a UNIX timestamp in seconds or RFC3339. (e.g. `date +%s`). End time is exclusive. `[start..end)`
- **step** is a query resolution step width in [duration](#) format or float number of seconds.



Note: Admins (`doppler.firehose` or `logs.admin`) are permitted to use this query without specifying a `source_id` in the query parameter. Non-admins cannot use regex matchers on `source_id`.

```
$ curl -H "Authorization: $(cf oauth-token)" -G "https://metric-store.SYSTEM-DOMAIN/api/v1/query_range" \
  --data-urlencode 'query=metric_name_0{source_id="source_id_0"}' \
  --data-urlencode "start=$(date -d '24 hours ago' +%s)" \
  --data-urlencode "end=$(date +%s)" \
  --data-urlencode 'step=1h'
```

Response Body

```
{
  "status": "success",
  "data": {
    "resultType": "matrix",
    "result": [
      {
        "metric": {"__name__": "metric_name"},
        "values": [
          [ "<timestamp>", "<value>" ],
          "... "
        ]
      },
      "... "
    ]
  }
}
```

GET /api/v1/series

Issues a PromQL series query against Metric Store data. You can read more detail in the Prometheus documentation [here](#).



Note: Non-admins (`doppler.firehose` or `logs.admin`) are not permitted to use this query.

Query Parameters:

- **start** is a UNIX timestamp in seconds or RFC3339. (e.g. `date -d '24 hours ago' +%s`). Start time is inclusive. `[start..end)`
- **end** is a UNIX timestamp in seconds or RFC3339. (e.g. `date +%s`). End time is exclusive. `[start..end)`
- **match[]** is a [series selector](#).

```
$ curl -H "Authorization: $(cf oauth-token)" -G "https://metric-store.SYSTEM-DOMAIN/api/v1/series" \
  --data-urlencode 'match[]=metric_name_0' \
  --data-urlencode 'match[]=metric_name_1' \
  --data-urlencode "start=$(date -d '24 hours ago' +%s)" \
  --data-urlencode "end=$(date +%s)"
```

Response Body

```
{
  "status": "success",
  "data": [
    {
      "__name__": "metric_name_0",
      "source_id": "source_id_0",
      "...": ".."
    }
  ]
}
```

```

    },
    "...",
  ]
}

```

GET /api/v1/labels

Retrieve all label names for authorized source ids from the store.



Note: Non-admins (`doppler.firehose` or `logs.admin`) are not permitted to use this query.

Request

```
$ curl -H "Authorization: $(cf oauth-token)" "https://metric-store.SYSTEM-DOMAIN/api/v1/labels"
```

Response Body

```

{
  "status": "success",
  "data": [
    "__name__",
    "deployment",
    "...",
    "ip",
    "origin",
    "request_type",
    "source_id",
    "status_code",
    "unit"
  ]
}

```

GET /api/v1/label/<label_name>/values

Retrieve label values by `label_name` for all authorized source ids. The special label `__name__` can be used to retrieve metric names from the store.



Note: Non-admins (`doppler.firehose` or `logs.admin`) are not permitted to use this query.

Request

```
$ curl -H "Authorization: $(cf oauth-token)" "https://metric-store.SYSTEM-DOMAIN/api/v1/label/<label-name>/values"
```

Response Body


```
{
  "status": "success",
  "data": ["10", "1"]
}
```

Request

```
$ curl -H "Authorization: $(cf oauth-token)" "https://metric-store.SYSTEM-DOMAIN/api/v1/label/__name__/values"
```

Response Body

```
{
  "status": "success",
  "data": ["metric_name_0", "metric_name_1", "..."]
}
```

Monitoring Metric Store

This topic tells you how to monitor Metric Store. It includes key scaling indicators (KSIs) to guide Metric Store scaling decisions.

Overview

Metric Store can be scaled vertically at this time. When disk resources are reaching complete consumption, Metric Store will start dropping the oldest data first. When memory and/or CPU resources are reaching complete consumption, Metric Store should be vertically scaled.

Key Scaling Indicators

- [nozzle_dropped](#)

Metric Emitted by Metric Store

Metric Store publishes metrics for monitoring the Metric Store itself. You can use these metrics to observe the health of the Metric Store and determine if the VMs and disks are appropriately scaled.

Nozzle Job Metrics

The nozzle is the process that ingresses envelopes from Loggregator's Reverse Log Proxy (RLP).

Reports	Metric	Type	Notes
Number of envelopes sent to collocated Metric Store	<code>metric_store_nozzle_ingress_envelopes_total</code>	counter	When increasing, nozzle is correctly consuming from the RLP
Number of envelopes dropped when reading from RLP	<code>metric_store_nozzle_dropped_envelopes_total</code>	counter	If this number is increasing at a steady rate, it may indicate that you need scale up the size of your VMs
Number of points dropped in outbound channel	<code>metric_store_nozzle_dropped_points_total</code>	counter	Should always be zero. If not, it may be useful in debugging.
Number of points written to its collocated Metric Store	<code>metric_store_nozzle_egress_points_total</code>	counter	
Number of errors writing to a remote node	<code>metric_store_nozzle_egress_errors_total</code>	counter	If this number consistently increasing, it may indicate network issues or an overloaded Metric Store node

Total duration spent writing to points.	<code>metric_store_nozzle_egres_duration_seconds</code>	gauge
---	---	-------

CF Auth Proxy Job Metrics

Reports	Metric	Type	Notes
Duration in seconds of requests made to the auth proxy	<code>metric_store_auth_proxy_request_duration_seconds</code>	gauge	
Duration in seconds of external requests made to CAPI	<code>metric_store_auth_proxy_capi_request_duration_seconds</code>	gauge	

Metric Store Job Metrics

Reports	Metric	Type	Notes
Number of points ingressed to colocated Metric Store	<code>metric_store_ingress_points_total</code>	counter	This should be steadily increasing at a relatively consistent rate
Number of points successfully written to storage engine	<code>metric_store_written_points_total</code>	counter	This should be steadily increasing at a relatively consistent rate
Time spent writing points to the storage engine	<code>metric_store_write_duration_seconds</code>	gauge	
Percentage of free space on persistent disk	<code>metric_store_disk_free_ratio</code>	gauge	
Number of shards removed due to time-based expiration	<code>metric_store_expired_shards_total</code>	counter	
Number of shards removed due to disk space threshold	<code>metric_store_pruned_shards_total</code>	counter	
<code>metric_store_storage_days</code>	Days of data stored on disk	gauge	
Size of the index	<code>metric_store_index_size_bytes</code>	gauge	
Number of unique series stored in the index	<code>metric_store_series_count</code>	gauge	
Number of unique measurements stored in the index	<code>metric_store_measurements_count</code>	gauge	

Number of errors encountered reading from the storage engine	<code>metric_store_read_errors_total</code>	counter
Time spent retrieving tag values from the storage engine	<code>metric_store_tag_values_query_duration_seconds</code>	gauge
Time spent retrieving measurement names from the storage engine	<code>metric_store_measurement_names_query_duration_seconds</code>	gauge

Metric Store Remote Metrics

Reports	Metric	Type	Notes
Size of a replayer queue	<code>metric_store_replayer_disk_usage_bytes</code>	gauge	
Number of errors encountered writing to a replayer queue	<code>metric_store_replayer_queue_errors_total</code>	counter	
Number of bytes written to a replayer queue	<code>metric_store_replayer_queued_bytes_total</code>	counter	
Number of errors encountered reading from a replayer queue	<code>metric_store_replayer_read_errors_total</code>	counter	
Number of errors encountered replaying writes to a remote node	<code>metric_store_replayer_replay_errors_total</code>	counter	
Number of bytes successfully replayed to a remote node	<code>metric_store_replayer_replayed_bytes_total</code>	counter	
Number of points dropped while writing to a remote node	<code>metric_store_dropped_points_total</code>	counter	
Number of points successfully distributed to a remote node	<code>metric_store_distributed_points_total</code>	counter	
Time spent distributing points to a remote node	<code>metric_store_distributed_request_duration_seconds</code>	gauge	
Number of points collected by a metric-store instance from remote nodes	<code>metric_store_collected_points_total</code>	counter	