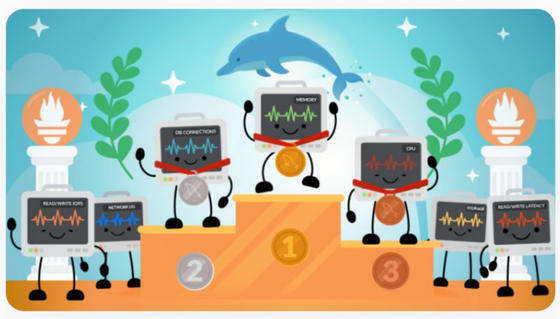
BACK TO BLOG

Top key metrics for monitoring MySQL



BY DAVID DE TORRES HUERTA - NOVEMBER 4, 2021

TOPICS: MONITORING

SHARE:

SHOW TABLE OF CONTENTS +

Monitoring MySQL with Prometheus is easy to do thanks to the MySQL Prometheus Exporter. MySQL doesn't need an introduction – it's one of the most used relational databases in the world, and it's also open-source!

Being such a popular database means that the community behind it is also huge. So don't worry: you won't be alone.

In this article, you'll learn how to start monitoring a MySQL Server with Prometheus – from configuring the exporter to what are the top metrics, with alert examples. If you're running MySQL with RDS, don't miss our top 5 key metrics for monitoring Amazon RDS.

Setting things up

Before deploying the exporter in the cluster, you have to create the user and password for the exporter in the database. Enter your MySQL database and execute the following queries:

```
CREATE USER 'exporter'@'%' IDENTIFIED BY 'YOUR-PASSWORD' WITH
MAX_USER_CONNECTIONS 3;
GRANT PROCESS, REPLICATION CLIENT, SELECT ON *.* TO 'exporter'@'%';
```

Substitute the user name and the password in the SQL sentence for your custom ones.

Once you have the user and password for the exporter in the database, you have to create a `mysql-exporter.cnf` file with the credentials of the exporter. Also, in this file you will include the url to connect your MySQL database in the field `host`:

```
[client]
user = exporter
password = "YOUR-PASSWORD"
host=YOUR-DB-IP
```

In your Kubernetes cluster, create the secret with the `mysql-exporter.cnf` file. This file will be mounted in the exporter to authenticate with the database:

```
kubectl create secret generic mysql-exporter \
  --from-file=.my.cnf=./mysql-exporter.cnf
```

How to deploy the MySQL Prometheus Exporter

To start monitoring MySQL with Prometheus, you'll need to deploy the MySQL Exporter in your cluster.

To deploy the exporter, you can use the example files provided below.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-exporter-exporter
spec:
  selector:
    matchLabels:
      app: mysql-exporter
  replicas: 1
  template:
    metadata:
      labels:
        app: mysql-exporter
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "9104"
    spec:
      containers:
        - name: mysql-exporter
          image: prom/mysqld-exporter:latest
          args:
            - --config.my-cnf=/tmp/.my.cnf
          ports:
            - containerPort: 9104
          volumeMounts:
            - name: my-cnf
              mountPath: /tmp/.my.cnf
              subPath: .my.cnf
          resources:
            limits:
              memory: "256Mi"
              cpu: "256m"
      volumes:
        - name: my-cnf
          secret:
            defaultMode: 420
            secretName: mysql-exporter
            items:
              - key: .my.cnf
                path: .my.cnf
```

After applying the Deployment for the exporter, Prometheus will automatically start scraping the MySQL metrics as it already has the standard annotations.

Top 5 metrics to monitor MySQL with Prometheus

#1 Availability

There are two metrics that monitor the availability of your MySQL instance. The first one is `mysql_up`. If this metric equals zero, the exporter cannot access the database, which can be a symptom of an unhealthy or failed database.

You can create an alert to notify you in case of a database down with the following query:

```
mysql_up == 0
```

Also, the metric `mysql_global_status_uptime` can give you an idea of quick restarts that can pass under the radar of the previous metric. You can create an alert with the following Prometheus query to detect instances that have an uptime of less than half an hour:

```
mysql_global_status_uptime < 1800
```

#2 Connections

One of the main sources of errors in databases are connection errors. The metric `mysql_global_status_connection_errors_total` allows you to detect when the database is generating these errors:

```
rate(mysql_global_status_connection_errors_total[5m])
```

One common cause of connection error is the lack of available connections. You can check the percent of available connections used with this Prometheus query:

```
100 * mysql_global_status_threads_connected /
mysql_global_variables_max_connections
```

If this value is near 100, the database can start refusing new connections. A solution for this can be increasing the number of maximum connections. Before doing that, be sure to check the available number of open files in the system. You can check it with the following Prometheus query:

```
mysql_global_variables_open_files_limit -
mysql_global_variables_innodb_open_files
```

Want to dig deeper into PromQL? Read our getting started with PromQL guide to learn how Prometheus stores data, and how to use PromQL functions and operators.

#3 Slow queries

Like many databases, MySQL keeps a log for slow queries. The number of entries in this log can be consulted with the metric `mysql_global_status_slow_queries`. You can create an alert with the following Prometheus query to notify when there are new entries in the slow queries log, which can mean that there is a performance issue:

```
rate(mysql_global_status_slow_queries[5m])
```

#4 Cache Hit Rate

MySQL uses in-memory cache to optimize the disk read and write operations. A low hit rate in the cache will impact the performance of the database. This Prometheus query will give you the value of the open tables cache hit rate:

```
rate(mysql_global_status_table_open_cache_hits[5m]) /
(rate(mysql_global_status_table_open_cache_hits[5m]) +
rate(mysql_global_status_table_open_cache_misses[5m]))
```

Also, you can monitor the buffer pool cache with the following promQL query:

```
(rate(mysql_global_status_innodb_buffer_pool_read_requests[5m]) -
rate(mysql_global_status_innodb_buffer_pool_reads[5m])) /
rate(mysql_global_status_innodb_buffer_pool_read_requests[5m]))
```

You can tune the buffer size for open tables and pool caches in the MySQL configuration, but keep in mind that doing so will affect the memory usage of your instance. Be sure that there is enough memory. You can check the available memory of the container using cAdvisor metrics:

```
sum by (namespace,pod,container)
((container_memory_usage_bytes{container!="POD",container!=""} -  on
(namespace,pod,container) avg by (namespace,pod,container)
(kube_pod_container_resource_limits{resource="memory"})) * -1 >0 ) /
(1024*1024*1024)
```

If your MySQL instance is in AWS RDS, check our blog post on how to monitor your RDS instance

#5 Query types

You can get statistics of the usage of each kind of SELECT query in your MySQL database, using the metric that gives information on each of them:

- Full join: mysql_global_status_select_full_join
- Full range join: mysql_global_status_select_full_range_join
- Select range check (joins without keys that check for key usage after each row): mysql_global_status_select_range_check
- Select scan: mysql_global_status_select_scan

All of them are counters, so remember to use the function rate in Prometheus queries, like this:

```
rate(mysql_global_status_select_full_join[5m])
```

Similarly, you can also have statistics on the use of sorting queries in MySQL:

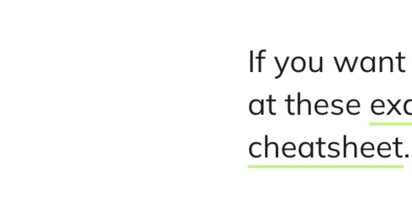
- Sort rows: mysql_global_status_sort_rows
- The number of sorts that were done using ranges: mysql_global_status_sort_range
- Sort merge (The number of merge passes that the sort algorithm has had to do): mysql_global_status_sort_merge_passes
- The number of sorts that were done by scanning the table: mysql_global_status_sort_scan

Next steps

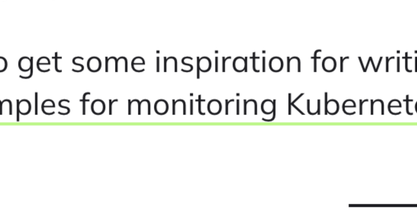
In this article, you learned how easy it is to start monitoring MySQL with Prometheus. You just need the right exporter and configuration.
You also learned the top Prometheus queries you should have in mind when monitoring MySQL with Prometheus.

If you want to get some inspiration for writing Prometheus queries, you can take a look at these examples for monitoring Kubernetes, or download our PromQL getting started cheatsheet.

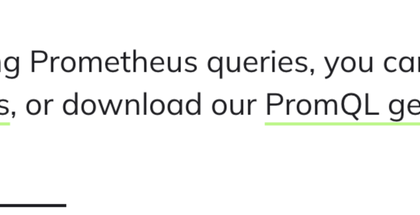
Related Content



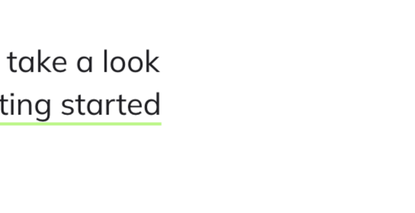
How to monitor an Oracle database with Prometheus



How to monitor Microsoft SQL Server with Prometheus



How to monitor Redis with Prometheus



Top metrics in PostgreSQL monitoring with Prometheus