RCOS Project Proposals

Project 1: Expose cluster config options to Telemetry (Python)

Background

Ceph is a software-defined distributed storage system used to securely and scalably store data. One of Ceph's features, the "telemetry module," [1] sends anonymous data about a cluster back to the Ceph developers to help us understand how Ceph is used and what problems users may be experiencing.

Problem

In telemetry, we currently collect the names of all user-modified configuration options in a cluster. This helps us know which options users have changed from the default settings. However, for some of the config options, we are interested in collecting the values in addition to their names. We are currently missing the values for two configuration options: `osd_memory_target` [2] and `osd_op_queue` [3].

Goal

Add a new metric collection to the telemetry module that reports two config values:

1. `osd_memory_target`
2. `osd_op_queue`

Knowing the value of `osd_memory_target` would help us understand how people are optimizing their heap memory usage object storage devices (OSDs). Knowing the value of `osd_op_queue` would help us understand how people are prioritizing operations in their clusters based on the op scheduler they set (i.e. "wpq" or "mClock").

Getting Started

1. Read up about how the telemetry module [1] works and familiarize yourself with the commands used to operate it in the documentation.
2. Start up a test cluster and play around with the telemetry commands to see how it works. Relevant commands include:
    a. `ceph telemetry show` - generates a report
    b. `ceph telemetry collection ls` - lists all the current metric collections
3. Visit the relevant code [4] to check out how the telemetry module is implemented. Focus on the collections you viewed in `ceph telemetry collection ls` are implemented.

Relevant Links

1. Learn about the telemetry module: https://docs.ceph.com/en/latest/mgr/telemetry/
2. Learn about `osd_memory_target`: https://docs.ceph.com/en/latest/rados/configuration/bluestore-config-ref/#confval-osd_memory_target
3. Learn about `osd_op_queue`: https://docs.ceph.com/en/latest/rados/configuration/osd-config-ref/?highlight=osd_op_queue#confval-osd_op_queue
4. Visit the relevant code: https://github.com/ceph/ceph/blob/main/src/pybind/mgr/telemetry/module.py

Project 2: Add rocksdb metric collection to telemetry (C++ and Python)

Background

Ceph is a software-defined distributed storage system used to securely and scalably store data. Ceph clusters are made up of various components that all work together to store and manage data.

One of those components, the OSD (object storage daemon) [1], is where data is stored. OSDs use what's called the "RocksDB key/value database" [2] to manage internal OSD metadata, such as mapping object names to their location on a disk.

As developers, we are able to see what version of RocksDB users are running in their clusters by checking the metrics that are reported by another one of Ceph's features, the "telemetry module." [3]. The telemetry module sends anonymous data about a cluster back to the Ceph developers to help us understand how Ceph is used and what problems users may be experiencing.

Problem

In addition to knowing the RocksDB version that users are running in their clusters, we lack the ability to track other more detailed information about how RocksDB is functioning in people's clusters.

Goal

Add extended RocksDB metrics as a new collection to the telemetry module to help us understand more how RocksDB is working overall in Ceph clusters.

This project can be achieved in two parts:

1. **C++**: Test out a new command called `dump_rocksdb_stats` that has already been written in this unmerged PR [4]. This command adds the ability to fetch extended RocksDB metrics. We do not have any unit test coverage for this command, so Part 1 would involve adding a unit test for this command to ensure it is working correctly. This part involves C++ knowledge.
2. **Python**: Add a new RocksDB metrics collection to the telemetry module code [5], which will utilize the CLI command from Part 1. This part involves Python knowledge.

See the "Getting Started" breakdown below. Given that the fundamental code for Part 1 is already written, Part 1 and Part 2 may be done in parallel, or sequentially, depending on how your group members want to divide the work.

Getting Started

1. Getting Started on Part 1 (C++):
    a. Using git, clone the code for the `dump_rocksdb_stats` command [4].
    b. Start a test cluster using the following command:
        i. `MON=1 OSD=1 MDS=0 RGW=0 NFS=0 ../src/vstart.sh -l -n -o rocksdb_perf=true`
    c. Test out the command by running the following:
        i. `./bin/ceph tell osd.* dump_rocksdb_stats telemetry`
    d. Once familiar with the command, check the sample unit test for an existing command that dumps performance counters [6] to see how a new unit test might be implemented for the new rocksdb command.
2. Getting Started on Part 2 (Python):
    a. Read up about how the telemetry module [1] works and familiarize yourself with the commands used to operate it in the documentation.
    b. Start up a test cluster and play around with the telemetry commands to see how it works. Relevant commands include:
        i. `ceph telemetry show` - generates a report
        ii. `ceph telemetry collection ls` - lists all the current metric collections
    c. Visit the telemetry code [4] to check out how the telemetry module is implemented. Focus on the collections you viewed in `ceph telemetry collection ls` are implemented.
    d. If you haven't already from Part 1, clone the code for the `dump_rocksdb_stats` command [4].
    e. Test out the command by running the following:
        i. `./bin/ceph tell osd.* dump_rocksdb_stats telemetry`
        ii. Now you can use the command in the telemetry code to add a new collection!

Relevant Links

1. Learn about Ceph OSDs: https://docs.ceph.com/en/latest/rados/configuration/storage-devices/#rados-configuration-storage-devices-ceph-osd
2. Learn about the RocksDB key/value database: https://docs.ceph.com/en/latest/rados/configuration/storage-devices/#bluestore
3. Learn about the telemetry module: https://docs.ceph.com/en/latest/mgr/telemetry/
4. Prior work: https://github.com/ceph/ceph/pull/45346
5. Visit the relevant code for telemetry: https://github.com/ceph/ceph/blob/main/src/pybind/mgr/telemetry/module.py
6. Unit test for perf counters: https://github.com/ceph/ceph/blob/cadfc96393e3189109a7a314ce3078d02896c5af/src/test/perf_counters.cc#L66-L81

Project 3: Extend mgr CLI command output for disabled modules (C++)

Background

Ceph is a software-defined distributed storage system used to securely and scalably store large amounts of data. One component of Ceph, the manager daemon [1], manages the overall health and performance by performing background actions like gathering metrics. The manager daemon also hosts various modules that offer services such as the front-end dashboard and telemetry reporting.

Problem

One of the manager CLI (command line interface) commands, `ceph mgr module ls -f json-pretty` [2], is used to display information in a json format about all the different mgr modules. This command currently displays detailed information about disabled modules, but not about “enabled” or “always-on” modules.

Goal

Extend the json output of `ceph mgr module ls -f json-pretty` so it displays detailed information about “enabled” and “always-on” modules to match the “disabled” module output.

Getting Started

1. Read about using Ceph mgr modules [2] and familiarize yourself with the commands related to listing them, enabling them, disabling them:
    a. `ceph mgr module ls` - lists the modules in plaintext

    b. `ceph mgr module ls -f json-pretty` - lists the modules in json form (many Ceph CLI commands have a plaintext form and a json form. You will ultimately only be changing the json form).
    c. `ceph mgr module enable <module>` - enables a module
    d. `ceph mgr module disable <module>` - disables a module
2. Start up a test cluster and actually run the above commands to see how they work.
3. Visit the relevant code [3] to see how `ceph mgr module ls -f json-pretty` is implemented. Observe the difference between how “enabled” vs. “disabled” modules are handled.

Relevant Links

1. Learn about the Ceph manager daemon: https://docs.ceph.com/en/latest/mgr/
2. Learn about the CLI command: https://docs.ceph.com/en/latest/mgr/administrator/?#using-modules
3. Visit the relevant code: https://github.com/ceph/ceph/blob/main/src/mon/MgrMonitor.cc