CONFIGURING CEPH

When you start the Ceph service, the initialization process activates a series of daemons that run in the background. A Ceph Storage Cluster runs three types of daemons:

- Ceph Monitor (`ceph-mon`)
- Ceph Manager (`ceph-mgr`)
- Ceph OSD Daemon (`ceph-osd`)

Ceph Storage Clusters that support the Ceph Filesystem run at least one Ceph Metadata Server (`ceph-mds`). Clusters that support Ceph Object Storage run Ceph Gateway daemons (`radosgw`).

Each daemon has a series of configuration options, each of which has a default values. You may adjust the behavior of the system by changing these configuration options.

OPTION NAMES

All Ceph configuration options have a unique name consisting of words formed with lower-case characters and connected with underscore (_) characters.

When option names are specified on the command line, either underscore (_) or dash (-) characters can be used interchangeable (e.g., `--mon-host` is equivalent to `--mon_host`).

When option names appear in configuration files, spaces can also be used in place of underscore or dash.

CONFIG SOURCES

Each Ceph daemon, process, and library will pull its configuration from several sources, listed below. Sources later in the list will override those earlier in the list when both are present.

- the compiled-in default value
- the monitor cluster's centralized configuration database
- a configuration file stored on the local host
- environment variables
- command line arguments
- runtime overrides set by an administrator

One of the first things a Ceph process does on startup is parse the configuration options provided via the command line, environment, and local configuration file. The process will then contact the monitor cluster to retrieve configuration stored centrally for the entire cluster. Once a complete view of the configuration is available, the daemon or process startup will proceed.

BOOTSTRAP OPTIONS

Because some configuration options affect the process's ability to contact the monitors, authenticate, and retrieve the cluster-stored configuration, they may need to be stored locally on the node and set in a local configuration file. These options include:

- `mon_host`, the list of monitors for the cluster
- `mon_dns_serv_name` (default: *ceph-mon*), the name of the DNS SRV record to check to identify the cluster monitors via DNS
- `mon_data`, `osd_data`, `mds_data`, `mgr_data`, and similar options that define which local directory the daemon stores its data in.
- `keyring`, `keyfile`, and/or `key`, which can be used to specify the authentication credential to use to authenticate with the monitor. Note that in most cases the default keyring location is in the data directory specified above.

In the vast majority of cases the default values of these are appropriate, with the exception of the `mon_host` option that identifies the addresses of the cluster's monitors. When DNS is used to identify monitors a local ceph configuration file can be avoided entirely.

SKIPPING MONITOR CONFIG

Any process may be passed the option `--no-mon-config` to skip the step that retrieves configuration from the cluster monitors. This is useful in cases where configuration is managed entirely via configuration files or where the monitor cluster is currently down but some maintenance activity needs to be done.

CONFIGURATION SECTIONS

Any given process or daemon has a single value for each configuration option. However, values for an option may vary across different daemon types even daemons of the same type. Ceph options that are stored in the monitor configuration database or in local configuration files are grouped into sections to indicate which daemons or clients they apply to.

These sections include:

`global`

Description: Settings under `global` affect all daemons and clients in a Ceph Storage Cluster.
Example: `log_file = /var/log/ceph/$cluster-$type.$id.log`

`mon`

Description: Settings under `mon` affect all `ceph-mon` daemons in the Ceph Storage Cluster, and override the same setting in `global`.
Example: `mon_cluster_log_to_syslog = true`

`mgr`

Description: Settings in the `mgr` section affect all `ceph-mgr` daemons in the Ceph Storage Cluster, and override the same setting in `global`.
Example: `mgr_stats_period = 10`

`osd`

Description: Settings under `osd` affect all `ceph-osd` daemons in the Ceph Storage Cluster, and override the same setting in `global`.
Example: `osd_op_queue = wpq`

`mds`

Description: Settings in the `mds` section affect all `ceph-mds` daemons in the Ceph Storage Cluster, and override the same setting in `global`.
Example: `mds_cache_size = 10G`

`client`

Description: Settings under `client` affect all Ceph Clients (e.g., mounted Ceph Filesystems, mounted Ceph Block Devices, etc.).
Example: `objecter_inflight_ops = 512`

Sections may also specify an individual daemon or client name. For example, `mon.foo`, `osd.123`, and `client.smith` are all valid section names.

Any given daemon will draw it’s settings from the global section, the daemon or client type section, and the section sharing its name. Settings in the most-specific section take precedence, so for example if the same option is specified in both `global`, `mon`, and `mon.foo` on the same source (i.e., in the same configurationfile), the `mon.foo` value will be used.

Note that values from the local configuration file always take precedence over values from the monitor configuration database, regardless of which section they appear in.

METAVARIABLES

Metavariables simplify Ceph Storage Cluster configuration dramatically. When a metavariable is set in a configuration value, Ceph expands the metavariable into a concrete value at the time the configuration value is used. Ceph metavariables are similar to variable expansion in the Bash shell.

Ceph supports the following metavariables:

`$cluster`

Description: Expands to the Ceph Storage Cluster name. Useful when running multiple Ceph Storage Clusters on the same hardware.
Example: `/etc/ceph/$cluster.keyring`
Default: `ceph`

`$type`

Description: Expands to a daemon or process type (e.g., `mds`, `osd`, or `mon`)
Example: `/var/lib/ceph/$type`

`$id`

Description: Expands to the daemon or client identifier. For `osd.0`, this would be `0`; for `mds.a`, it would be `a`.
Example: `/var/lib/ceph/$type/$cluster-$id`

`$host`

Description: Expands to the host name where the process is running.

`$name`

Description: Expands to `$type.$id`.
Example: `/var/run/ceph/$cluster-$name.asok`

`$pid`

Description: Expands to daemon pid.
Example: `/var/run/ceph/$cluster-$name-$pid.asok`

THE CONFIGURATION FILE

On startup, Ceph processes search for a configuration file in the following locations:

1. `$CEPH_CONF` (*i.e.,* the path following the `$CEPH_CONF` environment variable)
2. `-c path/path` (*i.e.,* the `-c` command line argument)
3. `/etc/ceph/$cluster.conf`
4. `~/.ceph/$cluster.conf`
5. `./$cluster.conf` (*i.e.,* in the current working directory)
6. On FreeBSD systems only, `/usr/local/etc/ceph/$cluster.conf`

where `$cluster` is the cluster's name (default ceph).

The Ceph configuration file uses an *ini* style syntax. You can add comments by preceding comments with a pound sign (#) or a semi-colon (;). For example:

```
# <--A number (#) sign precedes a comment.
; A comment may be anything.
# Comments always follow a semi-colon (;) or a pound (#) on each line.
# The end of the line terminates a comment.
# We recommend that you provide comments in your configuration file(s).
```

CONFIG FILE SECTION NAMES

The configuration file is divided into sections like `[global]` or `[mon.foo]`, where the section is a valid Ceph section name (*global*, a daemon type, or a daemon name) surrounded by square brackets.

Global settings affect all instances of all daemon in the Ceph Storage Cluster. Use the `[global]` setting for values that are common for all daemons in the Ceph Storage Cluster. You can override each `[global]` setting by:

1. Changing the setting in a particular process type (*e.g.,* `[osd]`, `[mon]`, `[mds]`).
2. Changing the setting in a particular process (*e.g.,* `[osd.1]`).

Overriding a global setting affects all child processes, except those that you specifically override in a particular daemon. For

example,

```
[global]
debug ms = 0

[osd]
debug ms = 1

[osd.1]
debug ms = 10

[osd.2]
debug ms = 10
```

MONITOR CONFIGURATION DATABASE

The monitor cluster manages a database of configuration options that can be consumed by the entire cluster, enabling streamlined central configuration management for the entire system. The vast majority of configuration options can and should be stored here for ease of administration and transparency.

A handful of settings may still need to be stored in local configuration files because they affect the ability to connect to the monitors, authenticate, and fetch configuration information. In most cases this is limited to the `mon_host` option, although this can also be avoided through the use of DNS SRV records.

SECTIONS AND MASKS

Configuration options stored by the monitor can live in a global section, daemon type section, or specific daemon section, just like options in a configuration file can.

In addition, options may also have a *mask* associated with them to further restrict which daemons or clients the option applies to. Masks take two forms:

1. `type:location` where *type* is a CRUSH property like *rack* or *host*, and *location* is a value for that property. For example, `host:foo` would limit the option only to daemons or clients running on a particular host.
2. `class:device-class` where *device-class* is the name of a CRUSH device class (e.g., `hdd` or `ssd`). For example, `class:ssd` would limit the option only to OSDs backed by SSDs. (This mask has no effect for non-OSD daemons or clients.)

When setting a configuration option, the *who* may be a section name, a mask, or a combination of both separated by a slash (`/`) character. For example, `osd/rack:foo` would mean all OSD daemons in the `foo` rack.

When viewing configuration options, the section name and mask are generally separated out into separate fields or columns to ease readability.

COMMANDS

The following CLI commands are used to configure the cluster:

- `ceph config dump` will dump the entire configuration database for the cluster.
- `ceph config get <who>` will dump the configuration for a specific daemon or client (e.g., `mds.a`), as stored in the monitors' configuration database.
- `ceph config set <who> <option> <value>` will set a configuration option in the monitors' configuration database.
- `ceph config show <who>` will show the reported running configuration for a running daemon. These settings may differ from those stored by the monitors if there are also local configuration files in use or options have been overridden on the command line or at run time. The source of the option values is reported as part of the output.
- `ceph config assimilate-conf -i <input file> -o <output file>` will injest a configuration file from *input file* and move any valid options into the monitors' configuration database. Any settings that are unrecognized, invalid, or cannot be controlled by the monitor will be returned in an abbreviated config file stored in *output file*. This command is useful for transitioning from legacy configuration files to centralized monitor-based configuration.

HELP

You can get help for a particular option with:

```
ceph config help <option>
```

Note that this will use the configuration schema that is compiled into the running monitors. If you have a mixed-version cluster (e.g., during an upgrade), you might also want to query the option schema from a specific running daemon:

```
ceph daemon <name> config help [option]
```

For example,:

```
$ ceph config help log_file
log_file - path to log file
  (std::string, basic)
  Default (non-daemon):
  Default (daemon): /var/log/ceph/$cluster-$name.log
  Can update at runtime: false
  See also: [log_to_stderr,err_to_stderr,log_to_syslog,err_to_syslog]
```

or:

```
$ ceph config help log_file -f json-pretty
{
    "name": "log_file",
    "type": "std::string",
    "level": "basic",
    "desc": "path to log file",
    "long_desc": "",
    "default": "",
    "daemon_default": "/var/log/ceph/$cluster-$name.log",
    "tags": [],
    "services": [],
    "see_also": [
        "log_to_stderr",
        "err_to_stderr",
        "log_to_syslog",
        "err_to_syslog"
    ],
    "enum_values": [],
    "min": "",
    "max": "",
    "can_update_at_runtime": false
}
```

The `level` property can be any of *basic*, *advanced*, or *dev*. The *dev* options are intended for use by developers, generally for testing purposes, and are not recommended for use by operators.

RUNTIME CHANGES

In most cases, Ceph allows you to make changes to the configuration of a daemon at runtime. This capability is quite useful for increasing/decreasing logging output, enabling/disabling debug settings, and even for runtime optimization.

Generally speaking, configuration options can be updated in the usual way via the `ceph config set` command. For example, do enable the debug log level on a specific OSD,:

```
ceph config set osd.123 debug_ms 20
```

Note that if the same option is also customized in a local configuration file, the monitor setting will be ignored (it has a lower priority than the local config file).

OVERRIDE VALUES

You can also temporarily set an option using the *tell* or *daemon* interfaces on the Ceph CLI. These *override* values are ephemeral in that they only affect the running process and are discarded/lost if the daemon or process restarts.

Override values can be set in two ways:

1. From any host, we can send a message to a daemon over the network with:

   ```
   ceph tell <name> config set <option> <value>
   ```

   For example,:

   ```
   ceph tell osd.123 config set debug_osd 20
   ```

   The *tell* command can also accept a wildcard for the daemon identifier. For example, to adjust the debug level on all OSD daemons,:

   ```
   ceph tell osd.* config set debug_osd 20
   ```

2. From the host the process is running on, we can connect directly to the process via a socket in `/var/run/ceph` with:

   ```
   ceph daemon <name> config set <option> <value>
   ```

   For example,:

   ```
   ceph daemon osd.4 config set debug_osd 20
   ```

Note that in the `ceph config show` command output these temporary values will be shown with a source of `override`.

VIEWING RUNTIME SETTINGS

You can see the current options set for a running daemon with the `ceph config show` command. For example,:

```
ceph config show osd.0
```

will show you the (non-default) options for that daemon. You can also look at a specific option with:

```
ceph config show osd.0 debug_osd
```

or view all options (even those with default values) with:

```
ceph config show-with-defaults osd.0
```

You can also observe settings for a running daemon by connecting to it from the local host via the admin socket. For example,:

```
ceph daemon osd.0 config show
```

will dump all current settings,:

```
ceph daemon osd.0 config diff
```

will show only non-default settings (as well as where the value came from: a config file, the monitor, an override, etc.), and:

```
ceph daemon osd.0 config get debug_osd
```

will report the value of a single option.