Monasca Documentation

Release 11.1.0.dev16

['OpenStack Foundation']

Jul 23, 2025

CONTENTS

The monitoring requirements in OpenStack environments are vast, varied, and highly complex. Monascas project mission is to provide a monitoring-as-a-service solution that is multi-tenant, highly scalable, performant, and fault-tolerant. Monasca provides an extensible platform for advanced monitoring that can be used by both operators and tenants to gain operational insights about their infrastructure and applications.

Monasca uses REST APIs for high-speed metrics, logs processing and querying. It integrates a streaming alarm engine, a notification engine and an aggregation engine.

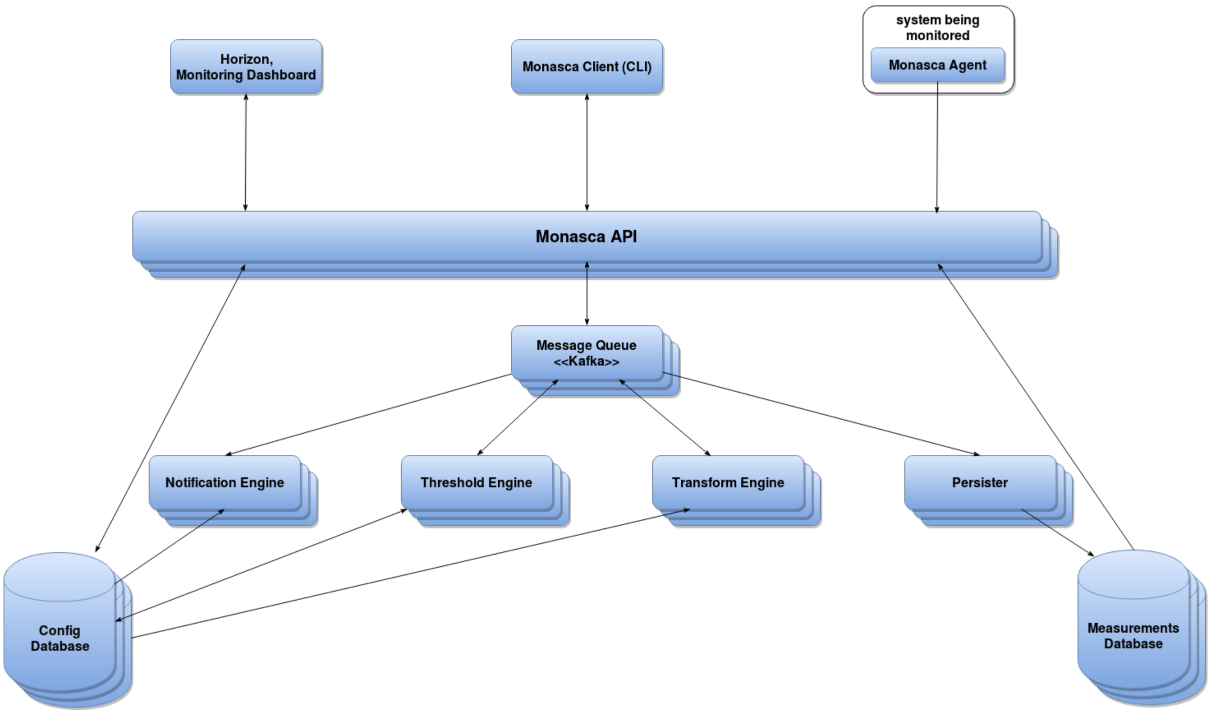
The use cases you can implement with Monasca are very diverse. Monasca follows a micro-services architecture, with several services split across multiple repositories. Each module is designed to provide a discrete service in the overall monitoring solution and can be deployed or omitted according to operators/customers needs.

CHAPTER

ONE

ARCHITECTURE

The following illustration provides an overview of Monascas metrics pipeline and the interaction of the involved components. For information on Monascas log pipeline, refer to this wiki page.



1.1 Repositories

- monasca-api: RESTful API for metrics, alarms, and notifications.
- monasca-agent: Agent for retrieving metrics data.
- monasca-persister: Writes metrics and alarm state transitions to a time-series database.
- monasca-thresh: Thresholding engine for computing thresholds on metrics and determining alarm states.
- monasca-notification: Pluggable notification engine for consuming alarm state transitions and sending notifications for alarms.
- monasca-transform: Aggregation engine based on Apache Spark.
- monasca-aggregator: Light-weight metrics aggregator.

Apart from sending requests directly to the API, the following tools are available for interacting with Monasca:

- Monasca Client: CLI and Python client.
- Horizon plugin: Plugin adding the monitoring panel to Horizon.
- Grafana app: Plugin for Grafana to view and configure alarm definitions, alarms, and notifications.

Libraries:

- monasca-common: Common code used in the Monasca components.
- monasca-statsd: StatsD-compatible library for sending metrics from instrumented applications.

Grafana integration:

- monasca-grafana-datasource: Multi-tenant Monasca data source for Grafana.
- grafana: Forked version of Grafana 4.1.2 with Keystone authentication added.

CHAPTER

TWO

FOR CONTRIBUTORS

2.1 Contribution documentation

2.1.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with Monasca.

Communication

For communicating with Monasca Team, you can reach out to us on *#openstack-monasca* IRC channel at OFTC.

We hold weekly team meetings in our IRC channel which is a good opportunity to ask questions, propose new features or just get in touch with the team.

You can also send us an email to the mailing list openstack-discuss@lists.openstack.org. Please use *[Monasca]* tag for easier thread filtering.

Contacting the Core Team

Name	IRC nick	Email
Martin Chacon Piza	chaconpiza	MartinDavid.ChaconPiza1@est.fujitsu.com
Witek Bedyk	witek	witold.bedyk@suse.com
Doug Szumski	dougsz	doug@stackhpc.com
Adrian Czarnecki	adriancz	adrian.czarnecki@ts.fujitsu.com
Joseph Davis	joadavis	joseph.davis@suse.com

New Feature Planning

Our process is meant to allow users, developers, and operators to express their desires for new features using Storyboard stories. The workflow is very simple:

- If something is clearly broken, submit a *bug report* in Storyboard.
- If you want to change or add a feature, submit a *story* in Storyboard.
- Monasca core reviewers may request that you submit a *specification* to gerrit to elaborate on the feature request.

- Significant features require *release notes* to be included when the code is merged.

Stories

New features can be proposed in Storyboard as new Story.

The initial story primarily needs to express the intent of the idea with enough details that it can be evaluated for compatibility with the project mission and whether or not the change requires a *specification*. It is *not* expected to contain all of the implementation details. If the feature is very simple and well understood by the team, then describe it simply. The story is then used to track all the related code reviews. Team members will request more information as needed.

Specifications

We use the monasca-specs repository for specification reviews. Specifications:

- Provide a review tool for collaborating on feedback and reviews for complex features.
- Collect team priorities.
- Serve as the basis for documenting the feature once implemented.
- Ensure that the overall impact on the system is considered.

Release Notes

The release notes for a patch should be included in the patch. If not, the release notes should be in a follow-on review.

If any of the following applies to the patch, a release note is required:

- The deployer needs to take an action when upgrading
- A new feature is implemented
- Plugin API function was removed or changed
- Current behavior is changed
- A new config option is added that the deployer should consider changing from the default
- A security bug is fixed
- Change may break previous versions of the client library(ies)
- Requirement changes are introduced for important libraries like oslo, six requests, etc.
- Deprecation period starts or code is purged

A release note is suggested if a long-standing or important bug is fixed. Otherwise, a release note is not required.

Task Tracking

We track our tasks in Storyboard

https://storyboard.openstack.org/#!/project_group/monasca

If youre looking for some smaller, easier work item to pick up and get started on, search for the *low-hanging-fruit* tag.

Kanban Board

Progress on implementation of important stories in Ussuri release is tracked in Monasca Board on StoryBoard.

Reporting a Bug

You found an issue and want to make sure we are aware of it? You can report them on Storyboard.

When filing a bug please remember to add the *bug* tag to the story. Please provide information on what the problem is, how to replicate it, any suggestions for fixing it, and a recommendation of the priority.

All open bugs can be found in this Worklist.

Getting Your Patch Merged

All changes proposed to Monasca requires at least one `Code-Review +2` votes from Monasca core reviewers before one of the core reviewers can approve patch by giving `Workflow +1` vote.

Reviews Prioritisation

Monasca project uses *Review-Priority* field in Gerrit to emphasize prioritized code changes.

Every developer can propose the changes which should be prioritized in weekly team meeting or in the mailing list. Any core reviewer, preferably from a different company, can confirm such proposed change by setting *Review-Priority* +1.

Prioritized changes can be listed in this dashboard.

Project Team Lead Duties

All common PTL duties are enumerated in the PTL guide.

2.1.2 Database Migrations

Monasca uses Alembic migrations to set up its configuration database. If you need to change the configuration databases schema, you need to create a migration to adjust the database accordingly, as follows:

```
cd monasca_api/db/
alembic revision
```

This will create a new skeleton revision for you to edit. You will find existing revisions to use for inspiration in the `/monasca_api/db/alembic/versions/` directory.

Measurement data stored in a Time Series database (such as InfluxDB) would be migrated to a new version using standard practice for a given TSDB.

2.1.3 Codebase documentation

Following section contains codebase documenation generated with, a little bit of assistance, sphinx.ext.autodoc.

Modules

CHAPTER

THREE

FOR OPERATORS

3.1 Administrating

3.1.1 Administration guide

Schema Setup

For setting up the Monasca configuration database, we provide `monasca_db`, an Alembic based database migration tool. Historically, the schema for the configuration database was created by a SQL script. This SQL was changed a couple of times, so `monasca_db` comes with a mechanism to detect the SQL script revision being used to create it and stamp the database with the matching Alembic revision.

Setting up a new database

If you are deploying Monasca from scratch, database setup is quite straightforward:

1. Create a database and configure access credentials with `ALL PRIVILEGES` permission level on it in the Monasca API configuration files `[database]` section.
2. Run schema migrations: `monasca_db upgrade`. It will run all migrations up to and including the most recent one (`head`) unless a revision to migrate to is explicitly specified.

Upgrading Existing Database from Legacy Schema

If you have been running an older version of Monasca, you can attempt to identify and stamp its database schema:

```
monasca_db stamp --from-fingerprint
```

This command will generate a unique fingerprint for the database schema in question and match that fingerprint with an in-code map of fingerprints to database schema revisions. This should work for all official (shipped as part of the `monasca-api` repository) schema scripts. If you used a custom third-party schema script to set up the database, it may not be listed and youll get an error message similar to this one (the fingerprint hash will vary):

```
Schema fingerprint 3d45493070e3b8e6fc492d2369e51423ca4cc1ac does not match␣
↪any known legacy revision.
```

If this happens to you, please create a Storyboard story against the openstack/monasca-api project. Provide the following alongside the story:

1. A copy of or pointer to the schema SQL script being used to set up the database.
2. The fingerprint shown in the error message.

3. The output of `monasca_db fingerprint --raw`.

Time Series Databases Setup

Enabling InfluxDB Time Series Index in existing deployments

If enabling TSI on an existing InfluxDB install please follow the instructions for migrating existing data here: https://docs.influxdata.com/influxdb/v1.7/administration/upgrading/#upgrading-influxdb-1-3-1-4-no-tsi-preview-to-1-7-x-tsi-enabled

Database Per Tenant

It is envisaged that separate database per tenant will be the default behaviour in a future release of Monasca. Not only would it make queries faster for tenants, it would also allow administrators to define retention policy per tenancy. To enable this, set *influxdb.db_per_tenant* to *True* in *monasca-{api,persister}* config (it defaults to *False* at the moment if not set).

To migrate existing data to database per tenant, refer to README.rst under the following URL which also contains the Python script to facilitate migration: https://opendev.org/openstack/monasca-persister/src/branch/master/monasca_persister/tools/db-per-tenant/

3.2 Glossary

3.2.1 Glossary

3.3 Installation

3.3.1 Installation

3.4 User

3.4.1 User guide

3.5 Configuration

- *Sample Config Files*

3.5.1 Command Line Interface

monasca (python-monascaclient)

This is the main command line interface for working with the Monasca services, including retrieving metrics from storage.

See the https://docs.openstack.org/python-monascaclient/latest/ for details.

monasca_db

CLI for Monasca database management.

```
usage: api [-h] [--config-dir DIR] [--config-file PATH] [--version]
           {fingerprint,detect-revision,stamp,upgrade,version} ...
```

monasca-status

CLI for checking the status of Monasca.

Use the command *monasca-status upgrade check* to check the readiness of the system for an upgrade.

Return Codes

Return code	Description
0	All upgrade readiness checks passed successfully and there is nothing to do.
1	At least one check encountered an issue and requires further investigation. This is considered a warning but the upgrade may be OK.
2	There was an upgrade status check failure that needs to be investigated. This should be considered something that stops an upgrade.
255	An unexpected error occurred.

History

Introduced in the Stein cycle as part of the OpenStack Community wide goal. https://governance.openstack.org/tc/goals/stein/upgrade-checkers.html

3.5.2 Samples

The following sections show sample configuration files for monasca-api and related utilities. These are generated from the code (apart from the samples for logging and paster) and reflect the current state of code in the monasca-api repository.

Sample Configuration For Application

This sample configuration can also be viewed in monasca-api.conf.sample.

```
[DEFAULT]
```

Sample Configuration For Logging

This sample configuration can also be viewed in api-logging.conf.

```
[loggers]
keys = root, sqlalchemy, kafka, kafkalib

[handlers]
keys = console, file

[formatters]
keys = context

[logger_root]
level = DEBUG
handlers = console, file

[logger_sqlalchemy]
```

(continues on next page)

(continued from previous page)

```
qualname = sqlalchemy.engine
# "level = INFO" logs SQL queries.
# "level = DEBUG" logs SQL queries and results.
# "level = WARN" logs neither.  (Recommended for production systems.)
level = DEBUG
handlers = console, file
propagate=0

[logger_kafka]
qualname = kafka
level = DEBUG
handlers = console, file
propagate = 0

[logger_kafkalib]
qualname = monasca_common.kafka_lib
level = INFO
handlers = console, file
propagate = 0

[handler_console]
class = logging.StreamHandler
args = (sys.stderr,)
level = DEBUG
formatter = context

[handler_file]
class = logging.handlers.RotatingFileHandler
level = DEBUG
formatter = context
# store up to 5*100MB of logs
args = ('/var/log/monasca/api/monasca-api.log', 'a', 104857600, 5)

[formatter_context]
class = oslo_log.formatters.ContextFormatter
```

Sample Configuration For Paster

This sample configuration can also be viewed in api-config.ini.

```
[DEFAULT]
name = monasca_api

[pipeline:main]
pipeline = request_id auth api

[app:api]
paste.app_factory = monasca_api.api.server:launch

```

(continues on next page)

(continued from previous page)

```
[filter:auth]
paste.filter_factory = monasca_api.healthcheck.keystone_protocol:filter_
↪factory

[filter:request_id]
paste.filter_factory = oslo_middleware.request_id:RequestId.factory

[server:main]
use = egg:gunicorn#main
host = 127.0.0.1
port = 8070
workers = 9
worker-connections = 2000
worker-class = eventlet
timeout = 30
backlog = 2048
keepalive = 2
proc_name = monasca-api
loglevel = DEBUG
```