Monasca
Monitoring software

B. van Walstijn, R. Stauttener, R. Bolding
Amsterdam University of Applied Sciences
January 23, 2019

-Overview

Monasca is a open-source multi-tenant, highly scalable, performant, fault-tolerant monitoring-as-a-service solution that integrates with OpenStack. It uses a REST API for high-speed metrics processing and querying and has a streaming alarm engine and notification engine.[1]

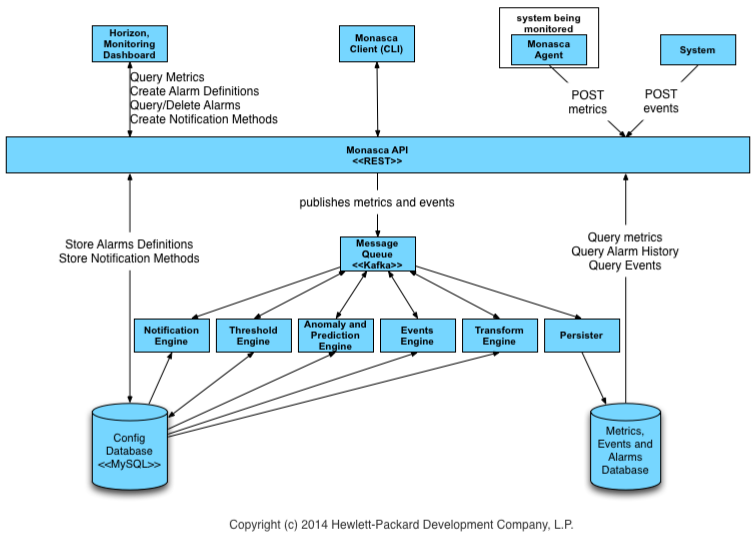


Figure 1: Monasca architecture

In this case the monasca monitoring software will be used to monitor the energy and system metrics of a computer cluster. This contains the power usage, CPU-usage and memory usage of every computer in the cluster. The client will be the controlling node in the system and the agents are the computer in the cluster. The monasca API will be set up using docker while the agents will be using the *monasca-agent* library available for installation from the Python Package Index (PyPI).

-Monasca agent

The Monasca Agent is the component of the Monasca monitoring system that collects metrics from the system it is running on and sends them to the Monasca API. The collector runs based on a configurable interval and

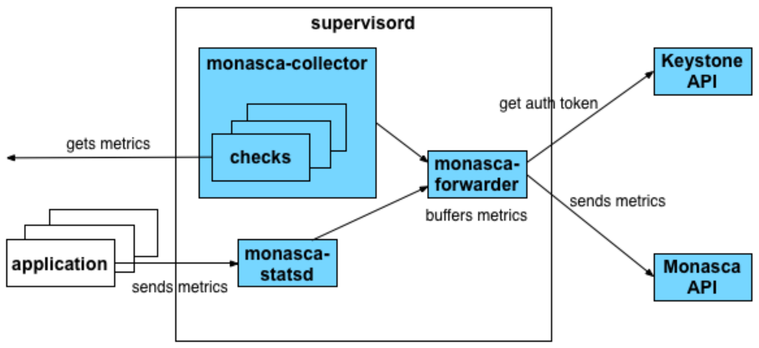


Figure 2: Monasca agent architecture

collects system metrics such as cpu or disk utilization as well as any metrics from additional configured plugins such as mySQL or Kafka. The forwarder takes the metrics from the collector and statsd daemon and forwards them on to the Monasca-API. Once sent to the Monasca-API, the metrics continue through the Monasca pipeline and end up in the Metrics Database. Once sent to the Monasca-API, the metrics continue through the Monasca pipeline and end up in the Metrics Database. In this case we only have to use the forwarder. Because the energy metrics will be gathered from measurements from the *INA219 current sensor* and the system metrics we need will be gathered by using the *psutil* library the metrics can go straight to the forwarder. *psutil* (process and system utilities) is a cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors) in Python.

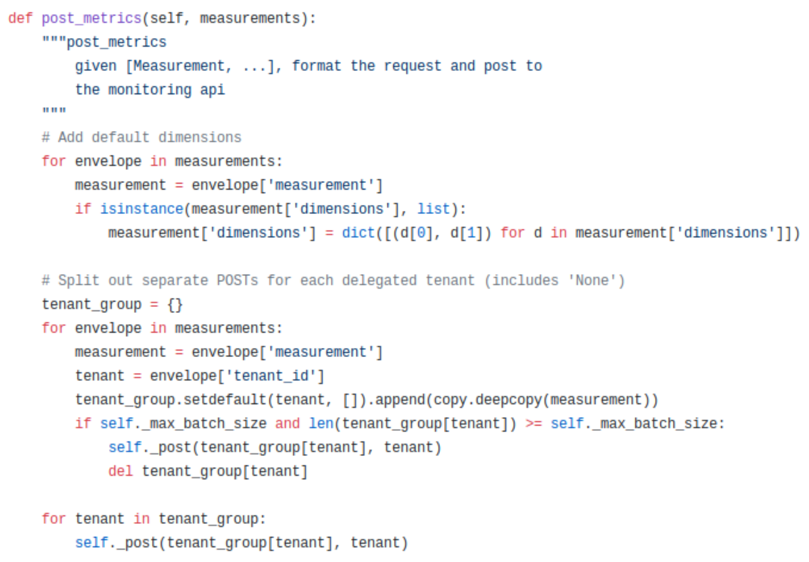
The agent is manually configured by an agent.yaml file. In the agant.yaml file will show information about:

- **Username** a required parameter that specifies the username needed to login to Keystone to get a token
- **Password** a required parameter that specifies the password needed to login to Keystone to get a token
- **monasca_url** specifies the url of the monasca api for retrieving tokens
- **keystone_url** specifies the url of the keystone api for retrieving tokens
- **dimensions** A comma separated list of key:value pairs to include as dimensions in all submitted metrics, like service
- **service** Service this node is associated with, added as a dimension

The configuration is really important, this decides wheter your metrics will be send correctly to the monasca API or not!

Forwarder

As described earlier, for this project we only have to use the forwarder of the monasca agent to send our metrics to the monasca API. In the forwarder component is a function called *post_metrics*. This function will be used to send your metrics to the monasca API (if your configuration file is complete).



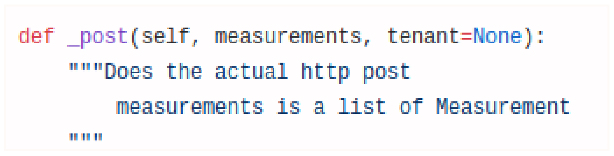
```
def post_metrics(self, measurements):
    """post_metrics
        given [Measurement, ...], format the request and post to
        the monitoring api
    """
    # Add default dimensions
    for envelope in measurements:
        measurement = envelope['measurement']
        if isinstance(measurement['dimensions'], list):
            measurement['dimensions'] = dict([(d[0], d[1]) for d in measurement['dimensions']])

    # Split out separate POSTs for each delegated tenant (includes 'None')
    tenant_group = {}
    for envelope in measurements:
        measurement = envelope['measurement']
        tenant = envelope['tenant_id']
        tenant_group.setdefault(tenant, []).append(copy.deepcopy(measurement))
        if self._max_batch_size and len(tenant_group[tenant]) >= self._max_batch_size:
            self._post(tenant_group[tenant], tenant)
            del tenant_group[tenant]

    for tenant in tenant_group:
        self._post(tenant_group[tenant], tenant)
```

Figure 3: The post_metrics function from the monasca-agent library

When using this function, the first argument would be *self* and the second argument would be your measurements/metrics. If the monasca API is up and running, the configuration file is complete and the function being used correctly, it will send your metrics to the monasca API. The library will handle the rest. As described in the last lines of the function, it will call other functions from the library, like *_post*



```
def _post(self, measurements, tenant=None):
    """Does the actual http post
        measurements is a list of Measurement
    """
```

Figure 4: The first bit of the _post function from the monasca-agent library

Installation

To install the monasca-agent library from the Python Package Index (PyPi), pip is required. to install *pip* (Ubuntu or Debian based install):

$ **sudo apt-get install python-pip**
$ **sudo pip install –upgrade pip**

The monasca-agent library will not run on Windows or Mac OS. The library is tested thoroughly on Ubuntu and should work under most flavors of Linux. To install the monasca-agent library:

```
$ sudo pip install monasca-agent
```

-Monasca-Docker

Monasca has a *monasca-docker* repository. This repository contains resources for building and deploying a full Monasca stack in Docker, which is basically everything the client needs. For monasca-docker you'll need a Docker environment.

- Docker 1.13 or later, 17.04.0-ce or later recommended
- docker-compose 1.9.0 or later
- At least 4 GiB of memory available to Docker
- At least 2 CPUs (cores)

For more information about installing Docker, go to: **https://docs.docker.com/install/**
For more information about installing docker-compose, go to: **https://github.com/Yelp/docker-compose/blob/master/docs/install.md**

Quickstart

First clone the monasca-docker repository the directory you wish to put it in. then you run *docker-compose*

```
$ git clone https://github.com/monasca/monasca-docker
$ cd monasca-docker
$ docker-compose up
```

The following services should be exposed to your host machine:

- Keystone on ports 5000 and 35357
- Monasca-API on port 8070
- Grafana on port 3000.
  Username: **mini-mon**
  Password: **password**

Keystone

Keystone takes care of everything that has to do with the authentication. If you want your agent to access the monasca-API, it should have the right authentication-settings in it's configuration file. All the keystone authentication-settings can be found and changed in the **monasca-docker/keystone/keystone-bootstrap.sh** file.

By default:

- Keystone username: admin
- Keystone password: s3cr3t
- Keystone project name: admin
- Keystone admin URL: http://localhost:35357
- Keystone public URL: http://localhost:5000

For more information about keystone in the monasca-docker repository and all the default authentication-settings, go to: **https://github.com/monasca/monasca-docker/tree/master/keystone**

References

1. Hewlett-Packard Development Company, L.P. 2015 [Accessed on Dec 2018]. [Online]. Available: `https://wiki.openstack.org/wiki/Monasca`
2. Giampaolo Rodola. Oct 2018 [Accessed on Nov 2018]. [Online]. Available: `https://pypi.org/project/psutil/`
3. Hewlett Packard Enterprise Development LP. 2018 [Accessed on Dec 2018]. [Online]. Available: `https://github.com/openstack/monasca-agent/blob/master/docs/Agent.md`
4. Hewlett-Packard Enterprise Development LP. 2017 [Accessed on Dec 2018]. [Online]. Available: `http://monasca.io/docs/deploy-docker.html`
5. Denis Poisson. Nov 2017 [Accessed on Jan 2019]. [Online]. Available: `https://github.com/monasca/monasca-docker/tree/master/keystone`