OpenStack Metering Using Ceilometer

Ruslan Kiyanchuk on July 3, 2013

The objective of the **OpenStack project** is to produce an Open Source Cloud Computing platform that will meet the needs of public and private clouds by being simple to implement and massively scalable. Since OpenStack provides infrastructure as a service (IaaS) to end clients, it's necessary to be able to meter its performance and utilization for billing, benchmarking, scalability, and statistics purposes.

Several projects to **meter OpenStack** infrastructure are available:

- **Zabbix** is a general-purpose enterprise-class open source distributed monitoring solution for networks and applications that can be customized for use with OpenStack.
- **Synaps** is an AWS CloudWatch-compatible cloud monitoring system that collects metric data, provides statistics data, and monitors and notifies based on user-defined alarms.
- **Healthmon** by HP aims to deliver "Cloud Resource Monitor," an extensible service to the OpenStack Cloud Operating System by providing a monitoring service for Cloud Resources and Infrastructure.
- **StackTach** is a debugging and monitoring utility for OpenStack that can work with multiple datacenters, including multi-cell deployments.
- The **Ceilometer** project is the infrastructure to collect measurements within OpenStack so that only one agent is needed to collect the same data. Its primary targets are monitoring and metering, but the framework is expandable to collect usage for other needs.

Among these, **Ceilometer** is actively developed and the most promising, well-suited for the purpose of metering OpenStack infrastructure. It graduated the incubation process and is now a part of OpenStack. In meteorology, a ceilometer is a device that uses a laser or other light source to determine the height of a cloud base. Thus, the Ceilometer project is a framework for monitoring and metering the OpenStack cloud and is also expandable to suit other needs.

Architecture of OpenStack Ceilometer

The primary purposes of the Ceilometer project are the following [1]:

- The efficient collection of metering data in terms of CPU and network costs.
- Collecting data by monitoring notifications sent from services or by polling the infrastructure.
- Configuring the type of collected data to meet various operating requirements.
- Accessing and inserting the metering data through the REST API.
- Expanding the framework to collect custom usage data by additional plugins.
- Producing signed and non-repudiable metering messages.

In order to fulfill these requirements, the following architecture has been implemented in the **Grizzly** release:

An API server provides access to metering data in the database via a REST API.

A central agent polls utilization statistics for other resources not tied to instances or compute nodes. There may be only one instance of the central agent running for the infrastructure.

A compute agent polls metering data and instances statistics from the compute node (primarily the hypervisor). Compute agents must run on each compute node that needs to be monitored.

A collector monitors the message queues (for notifications sent by the infrastructure and for metering data coming from the agents). Notification messages are processed, turned into metering messages, signed, and sent back out onto the message bus using the appropriate topic. The collector may run on one or more management servers.

A data store is a database capable of handling concurrent writes (from one or more collector instances) and reads (from the API server). The collector, central agent, and API may run on any node.

These services communicate using the standard OpenStack messaging bus. Only the collector and API server have access to the data store. The supported databases are MongoDB, SQL-based databases compatible with SQLAlchemy, and HBase; however, Ceilometer developers recommend MongoDB due to its processing of concurrent read/writes. In addition, only the MongoDB backend has been thoroughly tested and deployed on a production scale. A dedicated host for storing the Ceilometer database is recommended, as it can generate lots of writes [2]. Production scale metering is estimated to have 386 writes per second and 33,360,480 events a day, which would require 239 Gb of volume for storing statistics per month.

Integration of related projects into Ceilometer

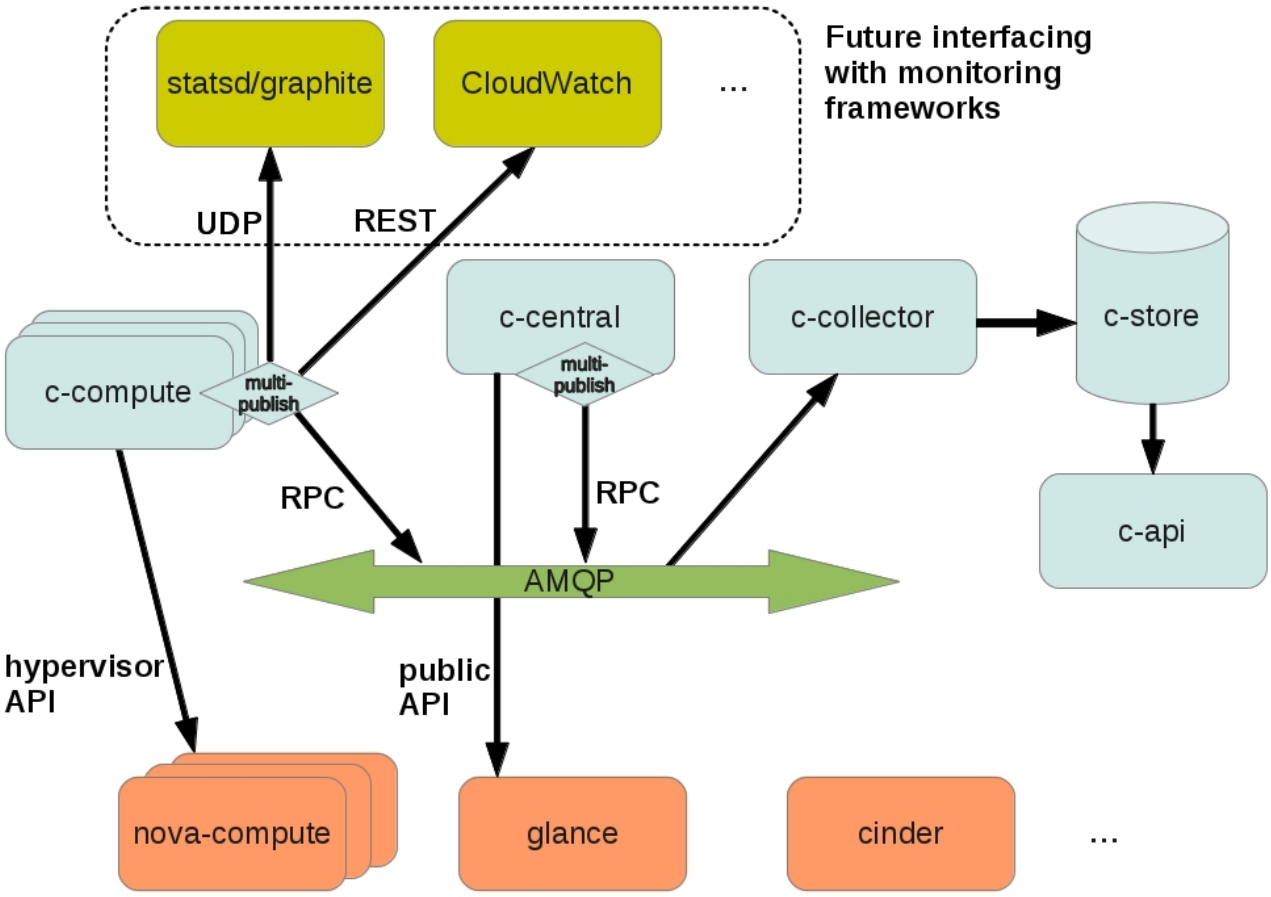
With the growth of OpenStack to production level, more missing features are needed for its successful usage as a cloud provider: billing system, usage statistics, autoscaling, benchmarking, and troubleshooting tools. After several projects were started to fulfill these needs it became clear that great part of their monitoring implementation has a common functionality.

In order to avoid fragmentation and functionality duplication, related projects will be integrated to provide a unified monitoring and metering API for other services. Because they had the same goal for metering, the Healthmon project was integrated into Ceilometer, but their data model and collection mechanisms were different [4]. A blueprint for Healthmon and Ceilometer integration has

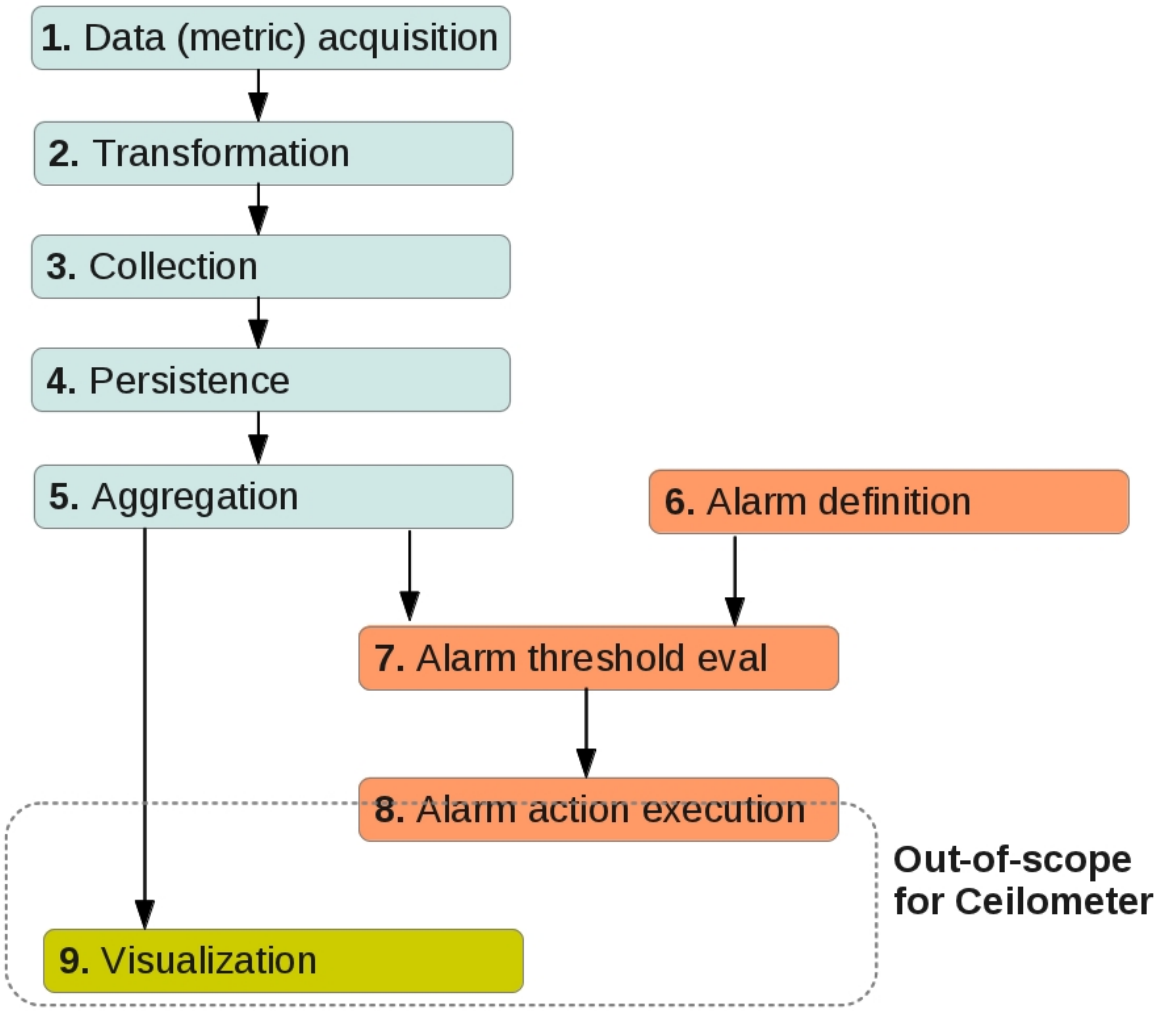
been created and approved for the OpenStack Havana release. The Synaps and StackTach projects had some unique functionality and are being integrated into Ceilometer as additional features. The main reason for Ceilometer's survival and integration of other projects is not the overwhelming features list implemented, but rather its good modularity. Most other metering projects would have implemented limited metering and some additional specific functionality. Ceilometer, however, will provide a comprehensive metering service and API to the obtained data to build any other feature, whether it's billing, autoscaling, or performance monitoring.

The cloud applications orchestrator project, OpenStack Heat, is also going to build its metric and alarm backend on top of Ceilometer API, which will allow implementation of such features as autoscaling [3]. This process includes introducing the alarm API and the ability to post metering samples via REST requests to Ceilometer and also rework Heat to make the metric logic pluggable.

Integration will extend the Ceilometer API with additional features and plugins and result in the following modifications in its architecture [5]:



Most of the integration and additional features are planned for the OpenStack **Havana** release. The primary roadmap for Ceilometer is to cover most of the metering and monitoring functionality and provide the possibility for other services (CLI, GUI, visualization, alarm action execution, etc.), to be built around the Ceilometer API.



Measurements in Ceilometer

Three type of meters are defined in ceilometer:

- **Cumulative:** Increasing over time (for example, instance hours)
- **Gauge:** Discrete items (for example, floating IPs or image uploads) and fluctuating values (such as disk I/O)
- **Delta:** Changing over time (for example, bandwidth)

Each meter is collected from one or more samples (gathered from the messaging queue or polled by agents), which are represented by counter objects. Each counter has the following fields:

– **counter_name**
The string for meter id. As a convention, the '.' separator is used to go from least to most discriminant word (for example, disk.ephemeral.size).

– **counter_type**
One of the described counter types (cumulative, gauge, delta).

– **counter_volume**
The amount of measured data (CPU ticks, bytes transmitted through network, provisioning time, etc.). This field is irrelevant for gauge-type counters and should be set to some default value in this case

(usually: 1).

– **counter_unit**
The description of the counter unit of measurement. Whenever a volume is to be measured, SI-approved units and their approved abbreviations will be used. Information units should be expressed in bits ('b') or bytes ('B'). When the measurement does not represent a volume, the unit description should always precisely describe what is measured (instances, images, floating IPs, etc.).

– **resource_id**
The identifier of the resource being measured (UUID for instance, network, image, etc.).

– **project_id**
The project (tenant) ID the resource belongs to.

– **user_id**
The User ID the resource belongs to.

– **resource_metadata**
Some additional metadata taken from the metering notification payload.

A full list of currently provided metrics may be found in the OpenStack Ceilometer documentation [6].

Ceilometer Features

Due to active development of Ceilometer and its integration with other projects, many additional features are planned for the OpenStack Havana release as blueprints. The upcoming and already implemented functionality is described below [7].

Post metering samples via the Ceilometer REST API

Implementation of this blueprint allows posting of metering data using Ceilometer REST API v2. A list of counters to be posted should be defined in JSON format and sent as a POST request to url `http://<metering_host>:8777/v2/meters/<meter_id>` (counter name corresponds to meter id). For example:

```
[
    {
        "counter_name": "instance",
        "counter_type": "gauge",
        "counter_unit": "instance",
        "counter_volume": "1",
        "resource_id": "bd9431c1-8d69-4ad3-803a-8d4a6b89fd36",
        "project_id": "35b17138-b364-4e6a-a131-8f3099c5be68",
        "user_id": "efd87807-12d2-4b38-9c70-5f5c2ac427ff",
        "resource_metadata": {
            "name1": "value1",
            "name2": "value2"
```

```
            }
        }
]
```

This enables the custom agents to post metering data to Ceilometer with minimum effort.

Alarms API in Ceilometer

Alarms will allow monitoring of a meter's state and notifications once some threshold value has been reached. This feature will enable numerous capabilities to be based on Ceilometer, like autoscaling, troubleshooting, and any other actions of infrastructure. The corresponding Alarm API blueprint is approved with high priority and planned for the Havana release.

Extending Ceilometer API

Ceilometer API will be extended to provide advanced functionality needed for billing engines, such as:

- Maximum usage of a resource that lasted more than 1 hour;
- Use of a resource over a period of time, listing changes by increment of X volume over a period of Y time;
- Providing additional statistics (Deviation, Median, Variation, Distribution, Slope, etc.).

The corresponding blueprint is approved and planned for the Havana-2 release.

Metering Quantum Bandwidth

Ceilometer will be extended for computing network bandwidth with Quantum. The Meter Quantum bandwidth blueprint is approved with medium priority for the Havana-3 release.

Monitoring Physical Devices

Ceilometer will monitor physical devices in the OpenStack environment, including physical servers running Glance, Cinder, Quantum, Swift, Nova compute node, and Nova controller and network devices used in the OpenStack environment (switches, firewalls). The Monitoring Physical Devices blueprint is approved for the Havana-2 release and its delivery is already in the process of code review.

Extending Ceilometer

Ceilometer was designed to be easy to extend and configure, so it can be tuned for each installation. A plugin system based on setuptools entry points provides the ability to add new monitors in the collector or subagents for polling.

Two kinds of plugins are expected: pollsters and listeners. Listeners process notifications generated by OpenStack components and put into a message queue to construct corresponding counter objects. Pollsters are for custom polling of infrastructure for specific meters, notifications for which

are not put in a message queue by OpenStack components. All plugins are configured in the `setup.cfg` file as `[entry_points]` section. For example, to enable custom plugins located in the `ceilometer/plugins` directory and defined as `MyCustomListener` and `MyCustomPollster` classes, the `setup.cfg` file should be customized as follows:

```
[entry_points]
ceilometer.collector =
    custom_listener = ceilometer.plugins:MyCustomListener
        ...
ceilometer.poll.central =
    custom_pollster = ceilometer.plugins:MyCustomPollster
        ...
```

The purpose of pollster plugins is to retrieve needed metering data from the infrastructure and construct a counter object out of it. Plugins for the central agent are defined in the `ceilometer.poll.central` namespace of `setup.cfg` entry points, while for compute agents they are the in `ceilometer.poll.compute` namespace. Listener plugins are loaded from the `ceilometer.collector` plugin.

The heart of the system is the collector, which monitors the message bus for data being provided by the pollsters as well as notification messages from other OpenStack components such as Nova, Glance, Quantum, and Swift.

A typical listener plugin must have several methods for accepting certain notifications from the message queue and generating counters out of them. The `get_event_types()` function should return a list of strings containing the event types the plugin is interested in. These events will be passed to the plugin each time a corresponding notification arrives.
The `notification_to_metadata()` function is responsible for processing the notification payload and constructing metadata that will be included into metering messages and accessible via Ceilometer API. The `process_notification()` function defines the logic of constructing the counter using data from specific notifications. This method can also return an empty list if no useful metering data has been found in the current notification. The counters are created by the `ceilometer.counter.Counter()` constructor, which accepts the required counter fields value (see Measurements). The meters provided by Ceilometer are implemented as plugins as well and may be used as a reference for creating additional plugins.

Summary

Ceilometer is a promising project designed to provide comprehensive metering and monitoring capabilities by fulfilling the functionality required to implement numerous features necessary for OpenStack production use. Even though Ceilometer already has stories of deployment (CloudWatch, AT&T, Dreamhost [5]), many changes and additional features will be developed by October 2013. Therefore, the project should be much better suited for production use after the Havana release, when most significant blueprints will be implemented.

References

[1] Ceilometer Developer Documentation

[2] Ceilometer: Choosing database backend

[3] Blueprint: Move to Ceilometer as Heat CloudWatch metric/alarm back-end

[4] Ceilometer and Healthmon integration

[5] Ceilometer Graduation

[6] OpenStack Ceilometer Documentation: Measurements

[7] Ceilometer blueprints approved for Havana release

ceilometer openstack

13 comments

13 Responses

1. Ranjit Nayak

It seems like Ceilometer is headed in multiple directions. It is performing Metering and Monitoring functions.

These are 2 distinct capabilities which may share common elements but are very distinct.

Monitoring function is related to crossing thresholds and raising the necessary alarms when the thresholds are crossed. It is also a realtime function and the user needs to take some kind of action only when the thresholds are crossed. The monitoring function could go down and still raise an alarm if the threshold has been crossed.

Metering on the other hand is a related to collecting usage data. It does not have to raise an alarm but if it goes down there is a loss of information. The data collected does not have to be acted up immediately.

July 5, 2013 15:04
Reply

- Ruslan Kiyanchuk

Thank you for the comment. Indeed, up until Havana Ceilometer has been going to provide only metering functions and you are right about differences between metering and monitoring.

However as the result of integrating Synaps into Ceilometer alarming API for monitoring purposes was decided to be implemented as well. The reason is that

Synaps itself had to implement metering first in order to provide any monitoring. Those functions are different but are very tight together. It would most likely be an overhead to implement metering and monitoring as separate independent services and that is the main reason why Synaps had agreed for integration. As mentioned in [5] (Ceilometer Graduation) “Alarm definition and threshold evaluation are also logical for Ceilometer to provide in the future, in terms of addressing the Heat requirement.”

Also, could you please give a detailed description of how an alarm may still be raised once the monitoring service went down, as was mentioned in your comment?

July 7, 2013 02:26
Reply

- Ranjit Nayak

  Ruslan,

  A monitoring service reads or polls and then has to store only the last measurement. If the service then goes down, I’m assuming since the last reading is persisted in a database, when it comes back up, it looks at the last reading and takes the next measurement. The change in these two should indicate a cross of threshold.

  July 11, 2013 15:05
  Reply

2. Peter Phaal

Polling the hypervisors for metrics doesn’t scale very well – particularly if you want low latency metrics for performance optimization. Host sFlow agents are available for most hypervisors – the agents efficiently push hypervisor and VM metrics to the collector. Support for the sFlow standard in physical and virtual switches provides end to end network visibility as well.

There is more to monitoring than just generating alerts based on thresholds and metering for billing. A comprehensive real-time view of performance lays the foundation for automated load placement, load balancing, multi-tenant performance isolation etc. as well as supporting alerting and metering.

July 10, 2013 08:21
Reply

- Ruslan Kiyanchuk

  Thank you for mentioning sFlow, there is a post with its overview.

  While being robust monitoring tool for network and servers, sFlow doesn’t cover many other metrics like Swift, Glance, Cinder statistics and kwapi (see Ceilometer measurements) and is also not so easily extendable. I couldn’t find cases of

deployment OpenStack with sFlow monitoring as well.

As for comprehensiveness of monitoring in Ceilometer, there were some clarifications in the previous comment. Primary goal of Ceilometer is metering, however alarming functionality is very cognant to already implemented features and is logical to be also provided — alarming was introduced as the result of integrating Synaps into Ceilometer.

> *"A comprehensive real-time view of performance lays the foundation for automated load placement, load balancing, multi-tenant performance isolation etc. as well as supporting alerting and metering."*
That is true, but such single tool would be overwhelmed and unmaintainable in case of supporting all those features at once. Modularity helps to keep things simple.

July 11, 2013 02:10
Reply

Peter Phaal

I don't see sFlow and Ceilometer as competing technologies – you could use them together, using Ceilometer to receive standard metrics from sFlow agents and poll for custom metrics. This hybrid approach increases scaleability and allows the incorporation of metrics from a wide range of sources. If you are familiar with using Host sFlow with Graphite or Ganglia, they blend standard sFlow metrics with custom metrics in a similar fashion.

The sFlow data can be sent to performance analysis, capacity planning and security tools running alongside Ceilometer. Sharing metrics reduces overhead and allows a wide array of requirements to be addressed by complementary tools.

You are correct, the focus of sFlow is efficient export of standard sets of metrics – it isn't intended for exporting arbitrary named values. However, I believe there is tremendous value in agreeing on a core set of metrics for each service: standardized metrics allow instrumentation to be efficiently embedded, and standardized metrics allow applications to do deeper analysis, rather than simply recording named values.

As community agreement develops on the core metrics for addition services within OpenStack, then please suggest them on the sFlow.org mailing list.

Finally, sFlow doesn't impose any particular collection architecture. As I mentioned, you could send the data Ceilometer, but for performance optimization applications there is great value in maintaining a global view and it is possible for an sFlow collector to maintain real-time visibility across all the hypervisors, VMs and switches – even for a large OpenStack installation.

July 12, 2013 09:33
Reply

Linus

What's the main changes between architecture 1 and 2?

July 31, 2013 02:45
Reply

Ruslan Kiyanchuk

The main change is the following:
multi-publisher implementation that enables agents to send metrics not only to collector but to various services via REST, UDP, RPC protocols.

Also the alarm API has been implemented even though it is not shown on architecture diagram.

August 14, 2013 00:59

Jacob Bushman

I think sflow would be a perfect compliment with Ceilometer.

I am working with a large Openstack deployment currently at Bluehost.

We have found it very difficult to scale the existing messaging system in Openstack and so we haven't adopted Ceilometer yet. Although we think it is a great project.

We are currently expanding our solution for monitoring and metering Openstack using sflow.

August 9, 2013 00:38
Reply

3. @TheSandyWalsh

StackTach is actively under development and rolled out in some of the largest OpenStack deployments. It has the most comprehensive reporting and monitoring available for OpenStack currently.

That said, we are in the (slow) process of migrating its functionality to Ceilometer.

September 13, 2013 11:20
Reply

4. Ram

Ceilometer page shows/lists down 6 major purpose, has all of them implemented. Has the code made public, could youpls let us know.
Wouldnt be a major change for org to move away from their custom built code to Ceilometer, upon release.

November 4, 2013 01:15
Reply

5. William

Since the Havana release, how do things look in Ceilometer? Have the release features that were mentioned above been implemented? Would you still recommend Ceilometer for metering and monitoring?

Thanks.

February 11, 2014 07:07
Reply

Continuing the Discussion

1. Dell Open Source Ecosystem Digest #23. Issue Highlight: "Stackalytics: Who's growing the OpenStack Pie" | ServerGround.net

   [...] Mirantis: "OpenStack Metering Using Ceilometer" [...]

   July 5, 201305:40

Post a comment

Some HTML is OK

Name *(required)*

Email *(required, but never shared)*

Web

Post comment

or, reply to this post via trackback.

Raleigh, NC.
OpenStack Bootcamp, Aug 19-21

Paris, France.
OpenStack Bootcamp, Aug 20-22

Dallas, TX.
OpenStack Bootcamp, Aug 26-29

Seattle, WA.
OpenStack Bootcamp, Sep 16-19

New York, NY.
OpenStack Bootcamp, Sep 16-19

Chicago, IL.
OpenStack Bootcamp, Sep 23-26

Munich, Germany.
OpenStack Bootcamp, Oct 7-10

Paris, France - OPENSTACK SUMMIT.
OpenStack Bootcamp, Oct 29-31

MOST POPULAR BLOG POSTS

- Cloud Prizefight: OpenStack vs. VMware
- OpenStack Networking Tutorial: Single-host FlatDHCPManager
- Configuring Floating IP addresses for Networking in OpenStack Public and Private Clouds
- OpenStack Networking – FlatManager and FlatDHCPManager
- Openstack Networking for Scalability and Multi-tenancy with VlanManager
- An OpenStack guy takes CloudStack for a test drive

BLOG ARCHIVE

August 2014

July 2014

June 2014

May 2014

April 2014

March 2014

- *Earlier*

LEADERSHIP
INVESTORS
BOARD OF DIRECTORS
PARTNERS
PRESS CENTER
PRIVACY POLICY

Mirantis Inc.
Pure Play OpenStack.

615 National Avenue, Suite 100 Mountain View, CA 94043
+1-650-963-9828

loading