Telemetry root source
https://wiki.openstack.org/wiki/Telemetry

Module Objectives

- Understand the telemetry program objectives
- Identify the telemetry architecture components
- Define the best practices for deploying and configuring telemetry services

Telemetry Objectives

Telemetry Program Mission

"To reliably collect data on the utilization of the physical and virtual resources comprising deployed clouds, persist these data for subsequent retrieval and analysis, and trigger actions when defined criteria are met."

Telemetry is an umbrella project with a broad goal divided amongst multiple smaller projects (talked about in the next slide)

Source:
https://wiki.openstack.org/wiki/Telemetry

Billing Process

- Billing process consists of 3 steps
  - Metering:
    - Collect usage data
  - Rating:
    - Transform usage data into billable items and calculate costs
  - Billing:
    - Create invoice and collect payment from clients
- Telemetry only provides Metering aspect of the billing process

Objectives of Telemetry Program

- "Metering"
  - Single source for rating engines to transform events into billable items
- "Multi-publisher"
  - Standard way to collect metrics regardless of collection purpose
  - Ex) Monitoring, debugging, and graphing tools
- "Alarming"
  - Tool to watch variations in key values and triggering action
  - Ex) Provide auto-scaling capabilities with Heat

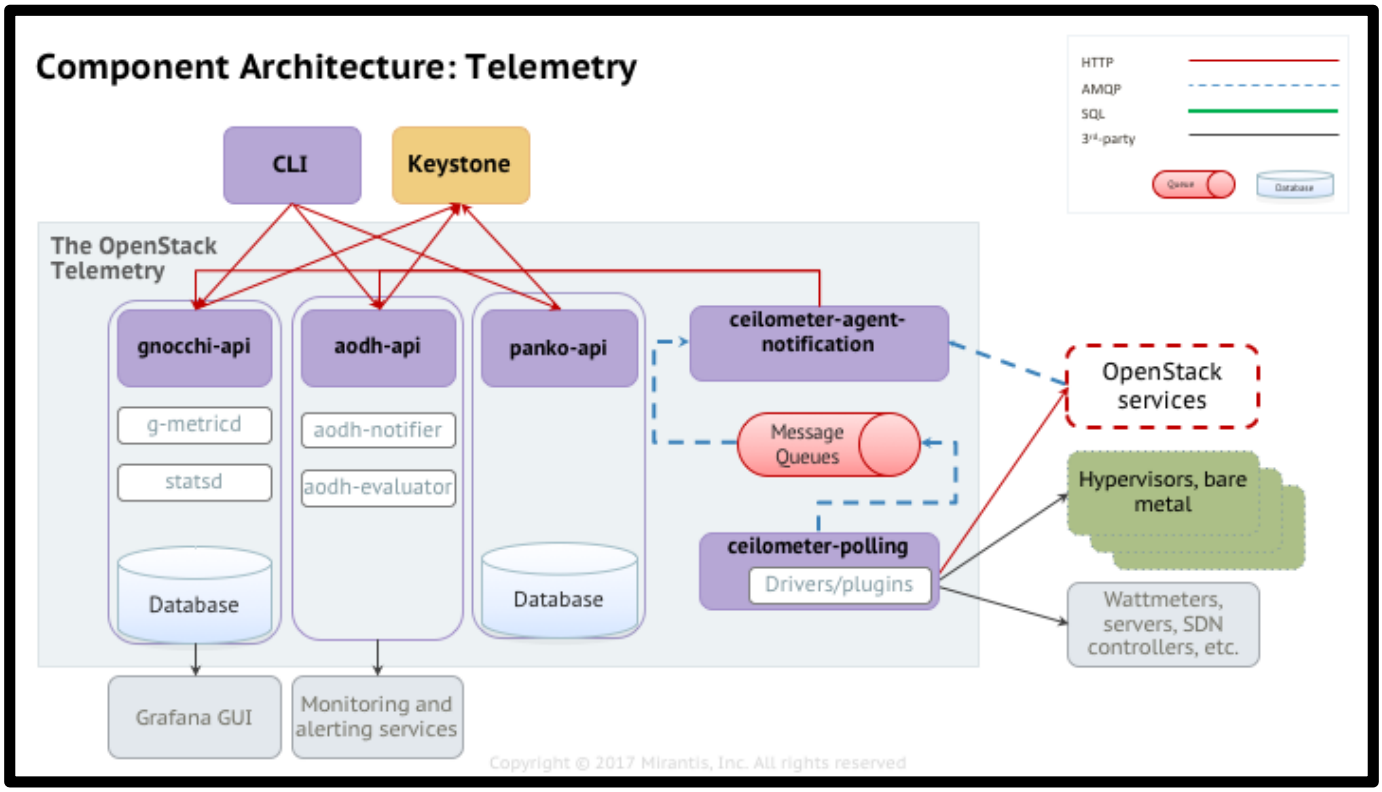
Telemetry Programs and Services

- Ceilometer
  - Data collection service
- Gnocchi
  - Large scale multi-tenant time series DB for storing and retrieving metrics information
- Aodh
  - Alarming service

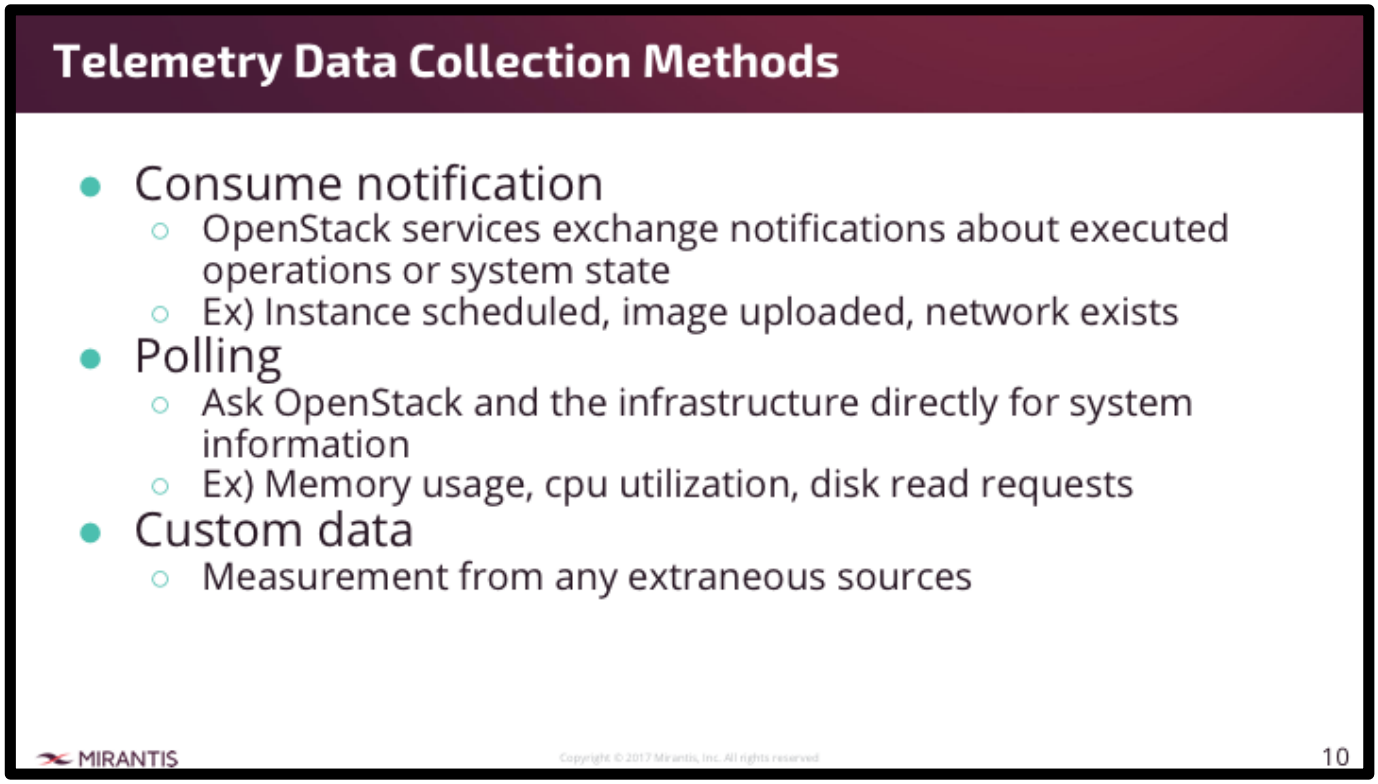
Panko is here too, but due to severe lack of documentation we won't mention it.

Telemetry Key Features

- Collect metering data
- Configure and transform data to be stored
- Record history of state changes as events
- Present collected data through REST API
- Create alarms and set triggers based on thresholds



https://docs.openstack.org/ceilometer/latest/admin/telemetry-system-architecture.html



“There are several types of data collection methods in telemetry”

All OpenStack services emit several notifications about the various types of events that happen in the system
during normal operation. These notifications can be consumed and interpreted into billable items.

There are much more resources to the cloud which cannot be observed from notifications alone. Polling the
OpenStack services and the infrastructure (like Hypervisor and server sensors) can gather additional information
About our cloud.

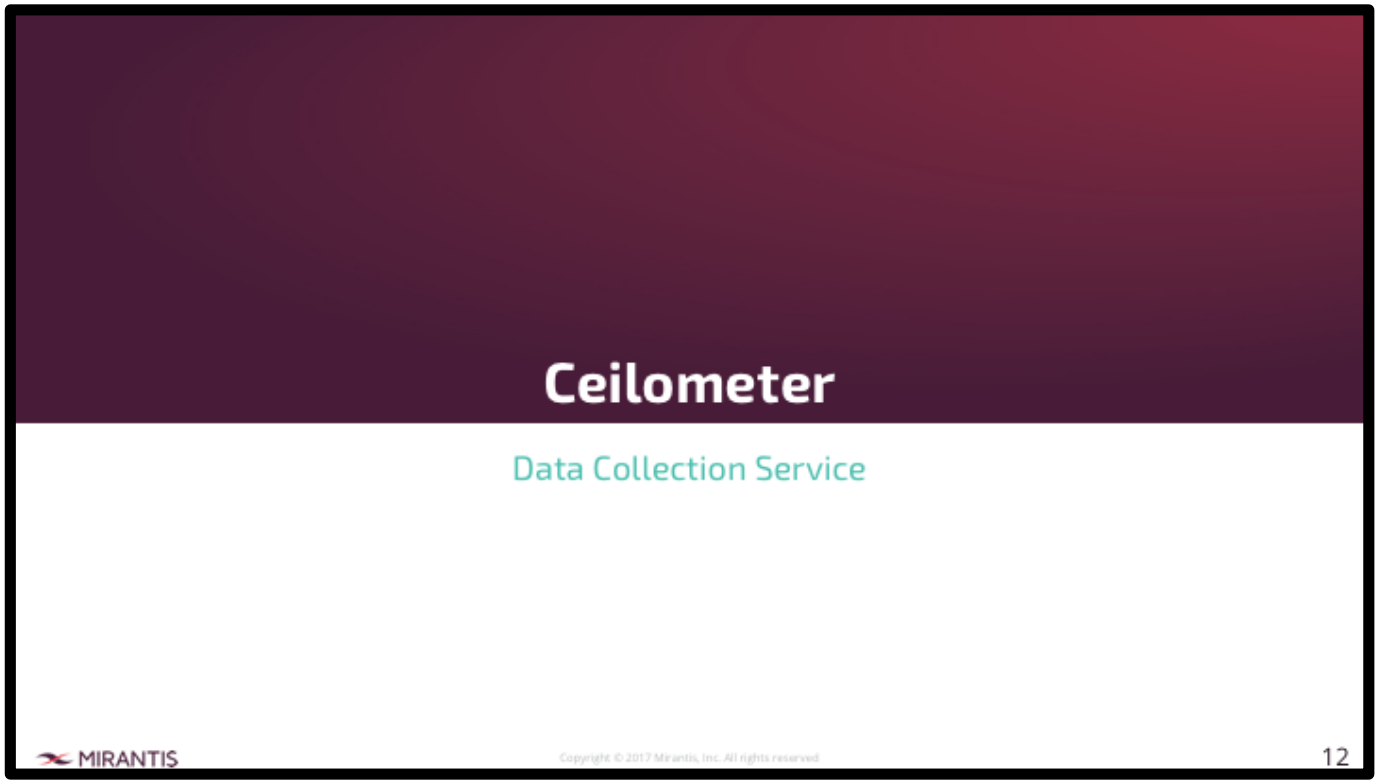
Lastly, you may have measurements outside the scope of OpenStack which may be brought into Telemetry program for
Archival purposes.

Source:
https://docs.openstack.org/ceilometer/pike/admin/telemetry-data-collection.html#telemetry-data-collection

Telemetry Data Retrieval Methods

- Telemetry API v2 (Deprecated)
  - $ ceilometer meter-list
  - Retrieve list of all meters available
- Gnocchi API
  - Query metrics & its data points

There is a telemetry API which provides access to meter database - for example the command ‘ceilometer meter-list’.
However since Ocata this has been deprecated in favor of using Gnocchi time series database



Let’s talk in-depth about each of the telemetry projects

Ceilometer

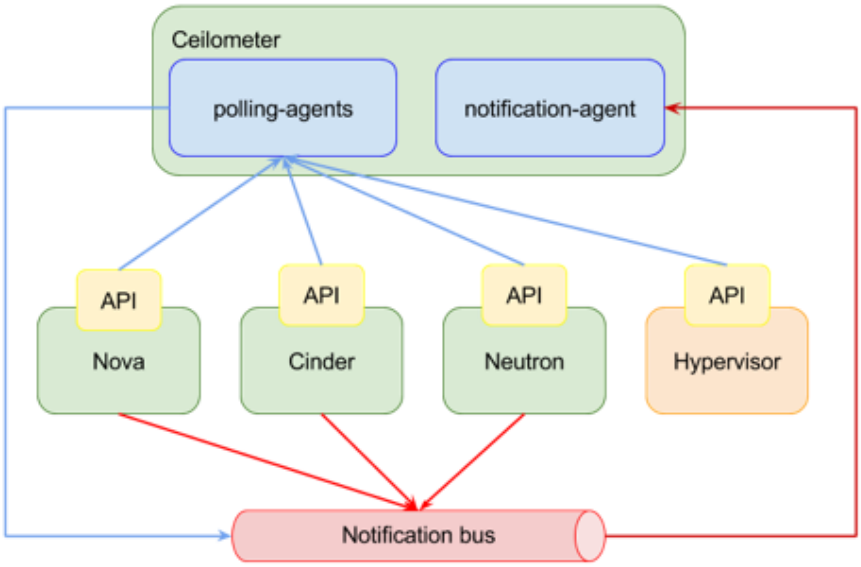
- Introduced in 2012
- Goal is to
  - Provide infrastructure to collect all OpenStack data
  - Transform events to billable objects called “meters”

In the early days of Ceilometer, OpenStack simply required a project which can provide
The infrastructure to collect events and provide metering data for billing purposes.

Eventually, OpenStack saw the need for additional features such as alarming functionalities.
All event related features were added in the Ceilometer project, but quickly became unscalable.

Source
https://docs.openstack.org/ceilometer/latest/contributor/overview.html
https://docs.openstack.org/releasenotes/ceilometer/ocata.html

Ceilometer Architecture: Data Collection



In Ceilometer there are two main methods of collecting data - Polling agents and notification agent.

Polling agents consist of multiple type of agents for its respective resources they are polling.
These agents can speak to various services API including OpenStack, Hypervisor, and baremetal depending on

The agent.

Notification agent consumes messages emitted by OpenStack services. It also has another job
Which is to transform and publish all the collected data (including data from polling agents).

Diagram shows the agents and their relationship to the rest of the cloud. Only go over data collection
Portion of the architecture and don't talk too much about publishing.

Source:
https://docs.openstack.org/ceilometer/latest/contributor/architecture.html

Ceilometer Data Collection Agents

- Data collection by agents that poll services
  - Compute agent
  - Central agent
  - IPMI agent
- Data collection by agent that listens to messages
  - Notification agent

Ceilometer Compute Agent

- Runs on each compute node
- Collect resource utilization data
  - VMs running on the host
  - Host itself
- Supported Hypervisors
  - Libvirt (KVM, QEMU, UML, LXC)
  - Xen
  - HyperV
  - VMware vSphere
- Samples sent to notification agent via AMQP

Ceilometer Central Agent

- Polling OpenStack services outside of compute
  - Neutron, Swift, Cinder, etc.
  - Existence of floating IP, number of objects, size of volume, etc.
- "Catch-all" agent
  - Hardware resources via SNMP
  - Measure CPU load, RAM, network traffic
- Samples sent to notification agent via AMQP

Ceilometer IPMI Agent

- Intelligent Platform Management Interface
  - Interface for monitoring hardware components
- Runs on each compute node
  - Note: Do not deploy IPMI agent on a node running Bare metal as a Service such as Ironic OpenStack program
- Collect sensor data from local node
  - CPU, RAM, I/O, Fan RPM, etc. utilization
- Requires IPMI capable host with ipmitool installed

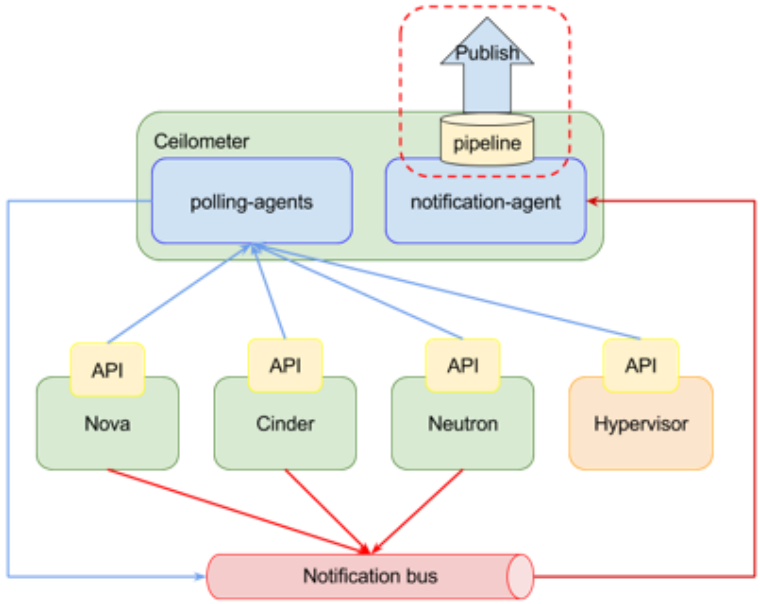
Source:
https://docs.openstack.org/ceilometer/pike/admin/telemetry-data-collection.html#telemetry-data-collection

Ceilometer Notification Agent

- Consume OpenStack notifications and transform messages into events and samples
  - Events from queue are filtered by event type to only include ones pertaining to billable items
  - Network create, image upload, volume exists, etc.
- Transform and publish data to backend

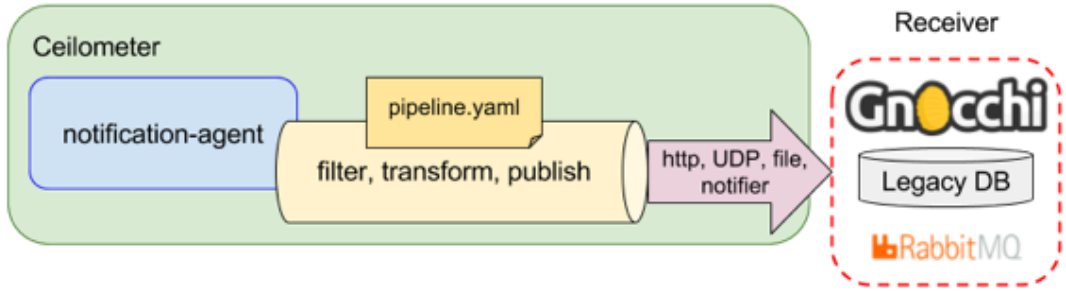
Source:
https://docs.openstack.org/ceilometer/pike/admin/telemetry-data-collection.html#telemetry-data-collection

Ceilometer Architecture: Processing Data



In the next segment of Ceilometer Architecture we will discuss the dotted red outline which looks at
How Ceilometer processes data.

Ceilometer Architecture: Processing Data



When we zoom-in to the highlighted section from the previous slide, we can see the details of the pipeline
And publishing mechanism. You can define your filters, transfomers, and publishers in pipeline.yaml and
Define the receiving endpoint which by default in Ocata will be Gnocchi

Ceilometer Pipeline

- Pipeline is the mechanism by which data is processed
- Defines coupling between:
  - **Source**: producer of data
  - **Sinks**: consumer of data providing logic for *transformation* and *publication*
- Stored in a configuration file: /etc/ceilometer/**pipeline.yaml**

Ceilometer processes data through a mechanism called Pipeline.

Pipeline is defined by the cloud admin through a configuration file called pipeline.yaml.
The configuration layout defines coupling between the source of the data and its sink - the consumer of data
Providing logic for transformation and publication.

Ceilometer pipeline.yaml Example

```
sources:
  - name: cpu_meter
    interval: 600
    meters:
      - "cpu"
    sinks:
      - cpu_delta_sink
      - another_sink
```

```
sinks:
  - name: cpu_delta_sink
    transformers:
      - name: "delta"
        parameters:
          target:
            name: "cpu.delta"
            growth_only: True
    publishers:
      - notifier://
```


Talk about the source & sink relationship

Ceilometer Basic Terminology

- Meters
  - Measure of a particular aspect of resource usage or ongoing performance
    - Ex) Instance exists, CPU utilization
- Sample
  - An individual numeric data point associated with a particular meter
  - This is the unit of storage
- Transformer
  - Section of pipeline which manipulates data before publishing

Ceilometer Basic Terminology

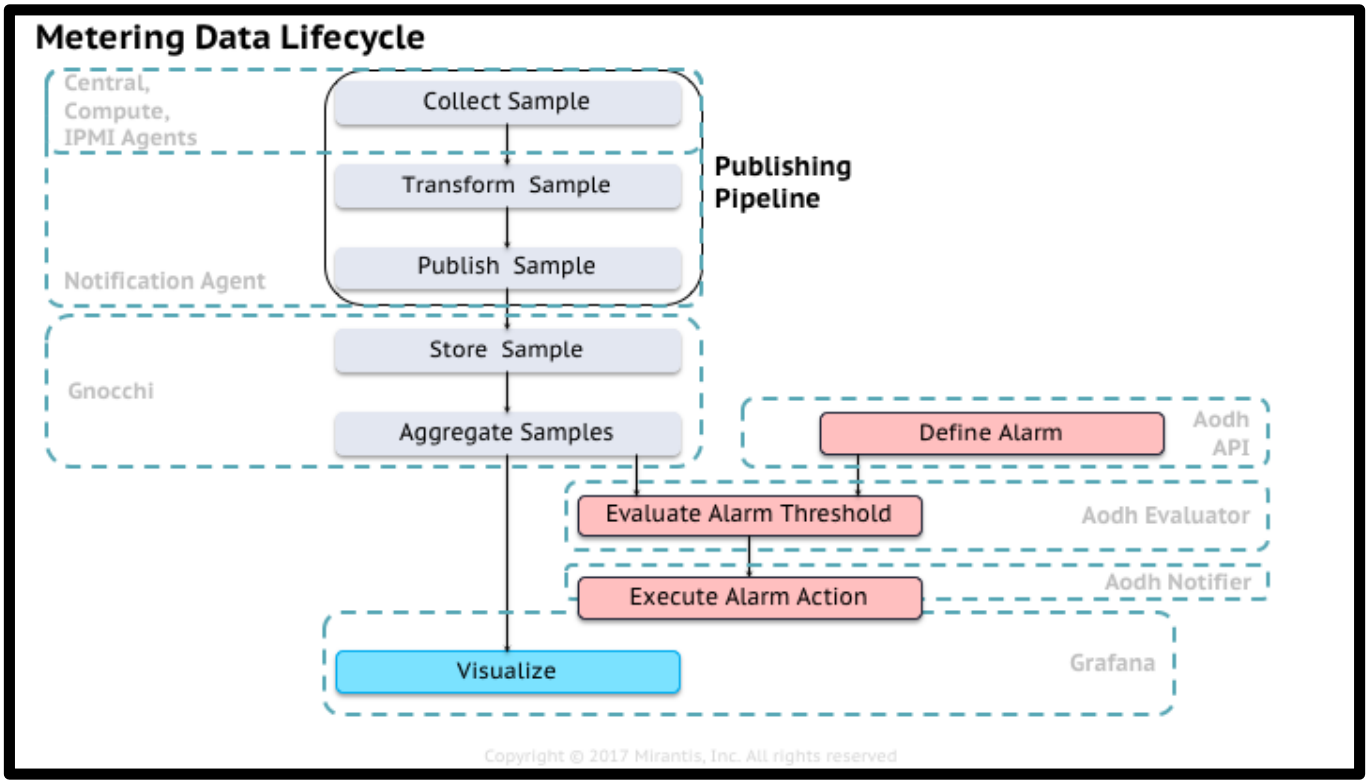
- Statistics
  - Aggregation of samples over a time period
    - Ex) Maximum CPU utilization over past 6 hours
- Events
  - The state of an object in an OpenStack service at a point in time
    - Ex) Instance down, image created

Ceilometer Meter Types

- Cumulative
  - Increasing over time
  - Ex) Instance hours
- Delta
  - Change since previous datapoint
  - Ex) Bandwidth, CPU delta, volume resize
- Gauge
  - Discrete items and fluctuating values
  - Ex) Floating IP, image uploads, memory usage

Telemetry uses 3 measurement types to encompass OpenStack's resources and its usage.

Source:
https://docs.openstack.org/ceilometer/pike/admin/telemetry-measurements.html



So far we discussed how meters are collected, transformed, and published. We also discussed

The responsibilities of various Ceilometer components. In the next section we will talk about Gnocchi, time-series database component
Which will store our samples and index resources.

This slide is based on:

http://www.mirantis.com/blog/openstack-metering-using-ceilometer/
http://openstack.redhat.com/CeilometerQuickStart
http://docs.openstack.org/developer/ceilometer/architecture.html
http://docs.openstack.org/developer/ceilometer/configuration.html#pipelines

What Instructor can talk about:

Ceilometer Agents (Compute, Central, Notification) collect, transform and publish metering data.

This chain of steps and their details are configured by something called Publishing Pipeline in Ceilometer. Pipeline configuration is stored in a separate configuration file, called pipeline.yaml, by default.

Ceilometer Collector listens for the published data and stores it in database.
Ceilometer API allows to query for the stored metering data and can also aggregate it into statistics.
Ceilometer Alarm Evaluator can use the statistics to compare to the alarm threshold

and make a decision if alarm state has changed. Ceilometer Alarm Evaluator will call the alarm action specified by user when defining an alarm.

In production environments the metering data and statistics will be also visualized, for example in form of graphs by Graphite, but that is out of the scope for Ceilometer for now. Also, alarm may require more complex actions than the ones provided by Ceilometer, for example manual intervention, that is also out-of-scope for Ceilometer and needs to be handled separately.

Don’t get too mumbo jumbo with Gnocchi. Introduce basic architecture and why it’s good.

Gnocchi

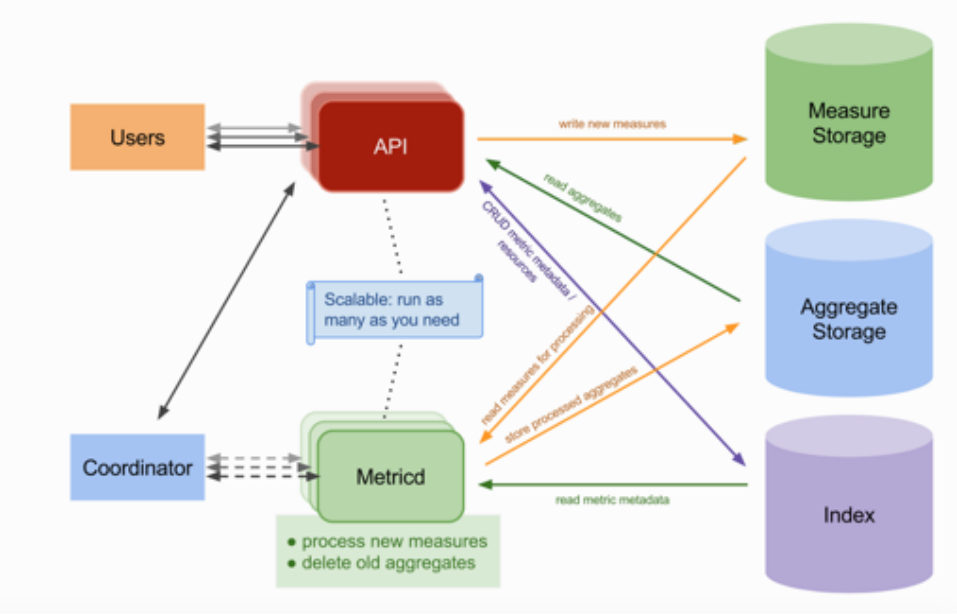
- Spin-off from Ceilometer in 2014
- Time series database designed to store **cloud** metrics at scale
- REST API to create and manipulate data
- Not an *OpenStack* program
  - Similar to RabbitMQ, MySQL

Gnocchi was started as a project in 2014 when Ceilometer encountered performance issues while using its
Standard database to store metrics.

The word cloud here is highlighted because Gnocchi was founded to specifically address the need to
Cleverly organize the vast amount of multiple types of data generated by the cloud.

Source:
http://gnocchi.xyz/index.html
https://julien.danjou.info/blog/2015/openstack-gnocchi-first-release
https://julien.danjou.info/blog/2014/openstack-ceilometer-the-gnocchi-experiment
(Not just any blog, this one is from former Ceilometer PTL and Gnocchi starter)

Gnocchi Architecture



Gnocchi API

- Two authentication methods
  - Basic HTTP Authentication header
  - Keystone Authentication
    - Each request to Gnocchi must contain valid Keystone token

MIRANTIS

Source:
http://gnocchi.xyz/install.html#configuring-authentication

Gnocchi Backend Types

- Incoming Storage
  - Stores new measurements sent to metrics
  - By default same location as aggregate storage (below)
- Aggregate Storage
  - Stores aggregate of metrics
  - Timestamp & values computed according to policy definition
- Indexer
  - Stores index of all resources, archive policies, and metrics
  - Link resources with metrics

Gnocchi Backend Drivers

- Incoming & Aggregate Storage drivers
  - File (default): May be enough depending on your infrastructure size
  - Ceph (preferred): More scalable and consistent than regular files thus preferred
  - Swift
  - S3
  - Redis
- Index driver
  - PostgreSQL (preferred)
  - MySQL (5.6.4+)

Source:
http://gnocchi.xyz/architecture.html#back-ends

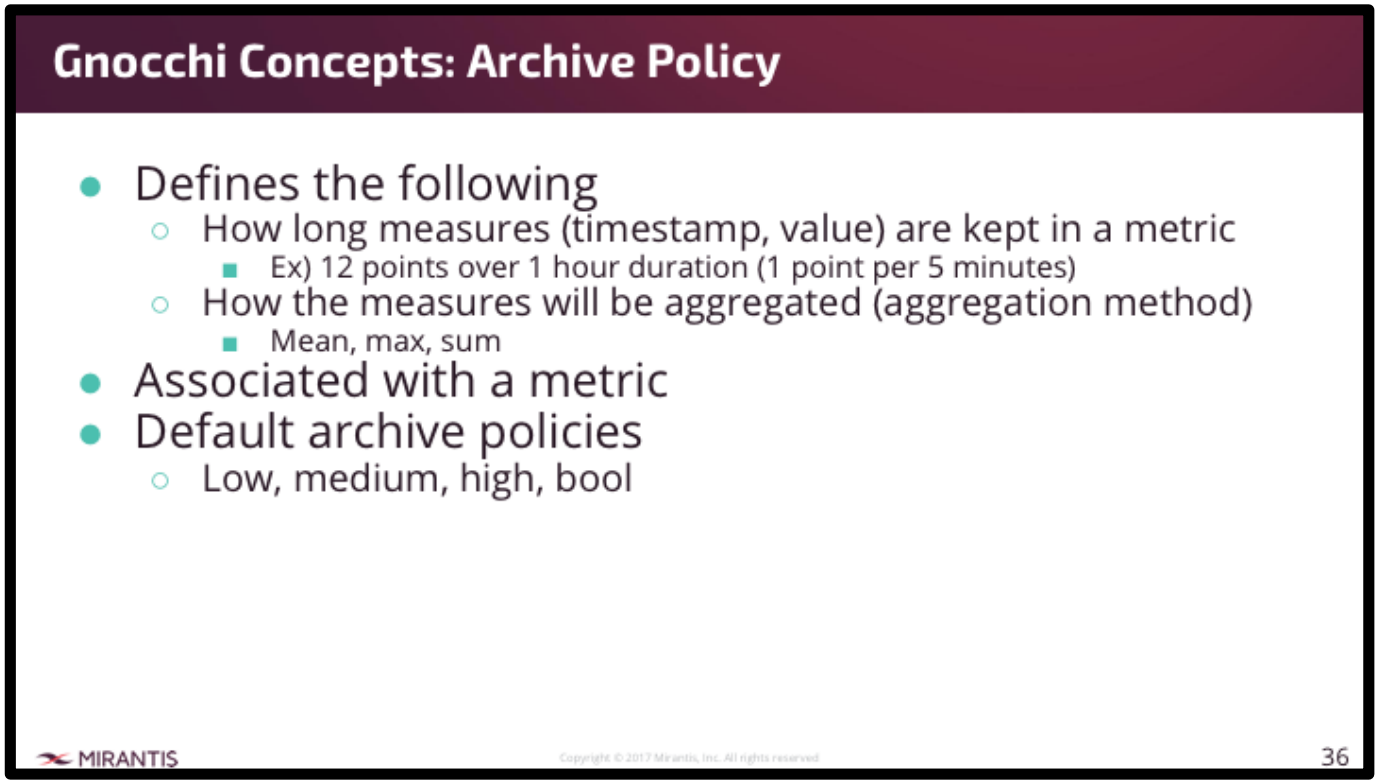
Gnocchi Concepts Overview

- Metrics Object
- Archive Policy
  - Metric aggregation
  - Aggregation methods
- Resources

Gnocchi does not store all metrics it gathers. Instead, it computes “metric aggregation” defined by the user for specific set of metrics.

Gnocchi Concepts: Metrics Object

- Represents a any thing that can be measured
  - CPU Usage of server
  - Room temperature
  - Number of bytes on an interface
- Includes the following properties
  - UUID
  - Archive policy



- low
  - 5 minutes granularity over 30 days
  - aggregation methods used: default_aggregation_methods
  - maximum estimated size per metric: 406 KiB
- medium
  - 1 minute granularity over 7 days
  - 1 hour granularity over 365 days
  - aggregation methods used: default_aggregation_methods
  - maximum estimated size per metric: 887 KiB
- high
  - 1 second granularity over 1 hour
  - 1 minute granularity over 1 week
  - 1 hour granularity over 1 year
  - aggregation methods used: default_aggregation_methods

        - maximum estimated size per metric: 1 057 KiB
- bool
        - 1 second granularity over 1 year
        - aggregation methods used: *last*
        - maximum optimistic size per metric: 1 539 KiB
        - maximum pessimistic size per metric: 277 172 KiB

Source: http://gnocchi.xyz/operating.html#how-to-define-archive-policies

Gnocchi Concepts: Resources

- An entity representing anything in your infrastructure
- Each resource has a type
  - Generic
  - Specialized types such as OpenStack entities
    - Ex) Instance ID, image, flavor, host
- Indexed by ID, user_id, project_id

Gnocchi Grafana Support

- Database can be viewed on GUI using Grafana
- Proxy Mode
  - Browser has access to Grafana, which has access to Gnocchi
  - Provide Keystone authtoken for authentication
- Direct Mode
  - Browser has direct access to Grafana, Gnocchi, and Keystone
  - Provide project, user, pass, Keystone URL and Grafana will ask Keystone for token

Aodh

Alarming Service

Aodh

- Spin-off from Ceilometer alarm service in Liberty
- Set and trigger alarms based on collected meters
- Alarms have three states
  - Ok
    - The alarm rule is evaluated as false
  - Alarm
    - The alarm rule is evaluated as true
  - Insufficient data
    - Not enough data points available to determine the state

Source: https://docs.openstack.org/aodh/latest/contributor/architecture.html#high-level-architecture

Aodh Alarm Evaluator

- aodh-evaluator
  - Periodically retrieve list of alarms and metering data
  - Evaluate data with alarm to determine when to fire alarms
  - Supports multiple types of alarms
    - Threshold rules
      - Gnocchi_resources_threshold_rule
      - Gnocchi_aggregation_by_resource_rule
      - Gnocchi_aggregation_by_resource_threshold_rule
    - Composite rules
      - Multiple alarms using combination of 'and' and 'or' relations

Aodh Alarm Notifier

- Listens to the notifier message queue for messages on when alarm state changes
- Perform action specified by the user for that alarm
  - HTTP callback
    - HTTP POST request to an endpoint containing description of alarm state change
  - Log
    - Alternative to HTTP callback, alarm state change is simply logged - primarily used for testing purposes