Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

SmartMetropolis – Plataforma e Aplicações para Cidades Inteligentes

WP4 – Infraestrutura

Relatório de Utilização e Ocorrências da Instância de Nuvem do Projeto Smart Metropolis Ano 2 Trimestre 2

Natal-RN, Brasil
Julho 2017

Equipe Técnica

Docentes

Prof. Dr. Carlos Eduardo da Silva (Coordenador) - IMD/UFRN

Discentes

Inácia Fernandes da Costa Neta

Pesquisadores vinculados

Welkson Renny de Medeiros - PPGSW

Sumário

1 Introdução

O projeto Smart Metropolis, conduzido pelo Instituto Metrópole Digital (IMD) da Universidade Federal do Rio Grande do Norte (UFRN), tem como objetivo o desenvolvimento de soluções de tecnologia da informação e comunicação para Cidades Inteligentes e Humanas. O projeto é organizado em seis Pacotes de Trabalho (Work Packages - WP) temáticos, liderados cada por um coordenador. Cada WP possui um conjunto de objetivos a serem alcançados ao longo dos cinco anos de execução do projeto.

Este relatório está inserido no contexto do WP 4 - Infraestrutura Computacional, que no contexto do segundo ano de execução do projeto Smart Metropolis, tem os seguintes objetivos principais:

- Operação de Infraestrutura: Este objetivo engloba a operação e manutenção da instância Fiware do projeto, incluindo ferramentas para monitoramento da infraestrutura, e para gerenciamento de aplicações.
- Segurança da Informação e Controle de Acesso: Este objetivo engloba o estudo aprofundado dos mecanismos de segurança da plataforma Fiware com o fim de oferecer uma solução de controle de acesso que possa ser utilizada pelas aplicações que irão executar sobre a infraestrutura Fiware.

Neste contexto, este relatório corresponde ao entregável definido pela **Meta 1.1: Relatório de ocorrências e utilização da instância FIWARE do projeto**, que envolve o relato de consumo de recursos da infraestrutura computacional do projeto, assim como relatos de eventos que demandaram a atenção da equipe em relação a operação e manutenção da nuvem.

De maneira a atingir esta meta, foram realizados estudos em algumas ferramentas de monitoramento de uma nuvem OpenStack. Tal estudo permitiu à equipe entender o funcionamento de tais ferramentas, assim como definir uma ferramenta que será utilizada para o monitoramento do consumo de recursos da nuvem do projeto.

Dessa forma, este relatório traz o estudo de uma parte de monitoramento da nuvem, abrangendo, resumidamente:

- Relato do consumo de recursos da infraestrutura computacional do projeto
- Estudo do serviço Telemetry que faz bilhetagem para que se possa fazer cobrança ao usuário
- Estudo detalhado do projeto Monasca, que oferece um serviço de monitoramento.

Por conseguinte, houve o entendimento da arquitetura operada pelo OpenStack, facilitando quaisquer ações necessárias, além da definição do que precisa ser adicionado para melhoria do monitoramento desenvolvido.

Este relatório apresenta uma descrição do estudo realizado, além de incluir um resumo das estatísticas de utilização da nuvem do projeto Smart Metropolis, sendo organizado da seguinte maneira: A Seção 2 apresenta um relato sobre o consumo de recursos da nuvem. A Seção 3 descreve o serviço Telemetry detalhadamente, com suas categorias internas e todos os atributos. A Seção 4 apresenta um relato sobre práticas que podem ser adotadas para melhorar o desempenho da nuvem. A Seção 5 trata minuciosamente do projeto Monasca e seus atributos. A Seção 6 conclui este relatório.

2 Relato de Uso da Nuvem

Esta seção apresenta um relato do consumo de recursos na nuvem do projeto Smart Metrópolis. Iniciamos descrevendo assim como as opções de máquinas virtuais disponíveis para os usuários, seguido de um

relato de uso dividido por mês.

2.1 Recursos da nuvem

Um usuário da nuvem pode fazer a escolha do tamanho de máquina que deseja instalar, dependendo do tamanho de sua aplicaçao. Atualmente, a nuvem do projeto oferece diversas opções de tamanhos, com variações na quantidade de VCPUs, memória RAM, e espaço de armazenamento em disco, como detalhado na tabela 1.

Flavor Name	**VCPUs**	**RAM**	**Disco Root**	***Ephemeral Disk***
m1.xlarge	8	16GB	80GB	0GB
m1.large	4	8GB	60GB	0GB
m1.medium	2	4GB	20GB	0GB
m1.small	1	2GB	20GB	0GB
m1.tiny	1	512MB	1GB	0GB
m1.micro	1	64MB	0GB	0GB

Tabela 1: Tamanho de memórias e disco disponíveis

Além disso, a nuvem disponibiliza um número de imagens de sistemas operacionais que podem ser usadas para a criação de máquinas virtuais. A tabela 2 apresenta as imagens disponibilizadas até o momento. Caso o usuário deseje, ele pode utilizar uma nova imagem, bastando adicioná-la na nuvem através de sua interface de acesso Web.

Projeto	**Nome da Imagem**	**Tipo**	**Status**	**Público**	**Protegido**	**Formato**	**Tamanho**
admin	CentOS 6	Image	Active	Yes	No	QCOW2	744.3 MB
admin	CentOS 7	Image	Active	Yes	No	QCOW2	1.3 GB
admin	CirrOS	Image	Active	Yes	No	QCOW2	12.7 MB
admin	CoreOS 1185.3.0	Image	Active	Yes	No	QCOW2	246.6 MB
admin	Debian Jessie 8.6.2 x64	Image	Active	No	No	QCOW2	544.0 MB
admin	Fedora Atomic 25	Image	Active	Yes	No	QCOW2	499.8 MB
services	TestVM	Image	Active	Yes	No	QCOW2	21.5 MB
admin	Ubuntu Trusty 14	Image	Active	Yes	No	QCOW2	219.1 MB
admin	Ubuntu Xenial 16.04	Image	Active	Yes	No	QCOW2	299.6 MB

Tabela 2: Tipos de imagens disponíveis

2.2 Recursos em uso

Detalhado abaixo, são encontrados os relatórios gerais (de todos os projetos existentes na nuvem até esse período) gerados, separados por mês do trimestre de Maio a Julho de 2017:

1 Maio (01.05.2017 á 31.05.2017)

- **Instâncias ativas:** 10
- **RAM ativo:** 28GB
- **Tempo total de utilização das VCPUs do sistema (horas):** 13301.87

- **Tempo total de utilização do disco (GB/hora) do sistema:** 266037.37
- **Tempo total de utilização da RAM (MB/hora) do sistema:** 27242226.92

Nome do projeto	**VCPUs**	**Disco**	**RAM**	**Uso total do VCPU (horas)**	**Uso total do disco (GB - horas)**	**Uso total da memória (MB - horas)**
common	12	240GB	24GB	9937.54	198750.81	20352083.06
aulasBTI	1	20GB	2GB	744.00	14879.99	1523711.43
admin	1	20GB	2GB	1876.33	37526.57	3842721.00
WP-Aplicacao	0	0 Bytes	0 Bytes	744.00	14879.99	1523711.43

Tabela 3: Resumo do uso total - Maio 2017

2 Junho (01.06.2017 á 30.06.2017)

- **Instâncias ativas:**11
- **RAM ativo:** 30GB
- **Tempo total de utilização das VCPUs do sistema (horas):** 10799.89
- **Tempo total de utilização do disco (GB/hora) do sistema:** 215997.84
- **Tempo total de utilização da RAM (MB/hora) do sistema:** 22118179.27

Nome do projeto	**VCPUs**	**Disco**	**RAM**	**Uso total do VCPU (horas)**	**Uso total do disco (GB - horas)**	**Uso total da memória (MB - horas)**
common	12	240GB	24GB	8640.00	172799.93	17694713.17
aulasBTI	1	20GB	2GB	720.00	14399.99	1474559.43
admin	1	20GB	2GB	720.00	14399.99	1474559.43
WP-Aplicacao	1	20GB	2GB	719.90	14397.92	1474347.24

Tabela 4: Resumo do uso total - Junho 2017

3 Julho (01.07.2017 á 31.07.2017)

- **Instâncias ativas:** 16
- **RAM ativo:** 44GB
- **Tempo total de utilização das VCPUs do sistema (horas):** 13270.35
- **Tempo total de utilização do disco (GB/hora) do sistema:** 260700.37
- **Tempo total de utilização da RAM (MB/hora) do**
- **sistema:** 27177685.18

Esse é o retrato atual do uso dos recursos utilizados e disponíveis da nuvem. A partir dos relatórios acima pode ser notado o avanço do uso da nuvem durante esse trimestre, evidenciando uma particularidade que é a quantidade de instâncias que não está diretamente correlacionada a quantidade de VCPUs

Nome do projeto	VCPUs	Disco	RAM	Uso total do VCPU (horas)	Uso total do disco (GB - horas)	Uso total da memória (MB - horas)
common	14	260GB	28GB	9734.70	193866.18	19936673.77
aulasBTI	1	20GB	2GB	804.33	16086.52	1647259.97
admin	1	20GB	2GB	804.33	16086.52	1647259.97
Puppet/Ansible	5	80GB	10GB	1122.67	18574.63	2299231.49
WP-Aplicacao	1	20GB	2GB	804.33	16086.52	1647259.97

Tabela 5: Resumo do uso total - Julho 2017

presentes. Isso é explicado devido cada instância poder ter mais de uma VCPU posta (conforme explanado no relatório anterior).

Esses relatórios deixam o administrador atento ao que será necessário expandir ou modificar, para que não haja problemas de interrupção da estrutura. Considerando as tabelas expostas, seriam aproveitados os valores de disco

3 Telemetry

O serviço Telemetry foi projetado para suportar sistemas de cobrança para recursos da nuvem OpenStack. As etapas necessárias para cobrar o uso em um ambiente em nuvem são medição, classificação e cobrança. Atualmente, o projeto de telemetria fornece um conjunto de funcionalidades divididas em vários projetos. A Figure 1 apresenta uma visão geral dos principais serviços do Telemetry.

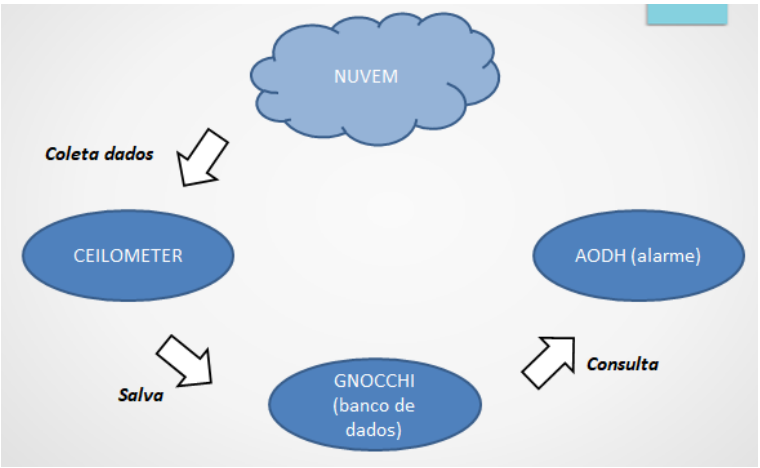

Figura 1: Serviços de acordo com suas funcionalidades

O Ceilometer é um serviço de coleta de dados, e objetiva coletar e transformar eficientemente os dados produzidos pelos serviços OpenStack. Esses destinam-se a ser usados para criar visões diferentes e ajudar a resolver vários casos de uso do Telemetry.

O Gnocchi é banco de dados de séries temporais e um serviço de listagem de recursos. Ele é um serviço de armazenamento de métricas que permite aos usuários capturar recursos do OpenStack e as métricas associadas a eles. Usando a agregação rolante definida por políticas de arquivamento (definidas pelo usuário), seu objetivo é fornecer um meio escalável de armazenar dados de curto e longo prazo e

fornecer uma visão estatística com base nos dados de entrada.

O Aodh é um serviço de alarme. Ele habilita a capacidade de acionar ações com base em regras definidas contra dados de amostra ou evento coletados pelo Ceilometer

Além desses, existe também o Panko, um serviço de armazenamento de eventos que permite aos usuários capturar a informação de estado dos recursos em um determinado momento. O objetivo é habilitar um meio escalável de armazenar dados de curto e longo prazos para casos de uso como auditoria e depuração do sistema.

A Figura 2 apresenta as várias ramificações do Telemetry, que foram utilizada para organização das subseções tratadas nessa seção: Subseção 5.3 descreve toda a parte da arquitetura suportada pelo sistema. A Subseção 3.2 apresenta um relato sobre a coleção de dados e suas finalidades diferentes. A Subseção 3.3 descreve como ocorre o processamento e tratamento dos dados. A Subseção 3.4 mostra como ocorrem os alarmes. A Subseção 3.5 mostra os diversos comandos existentes para o mais variados tipos de medições do sistema.

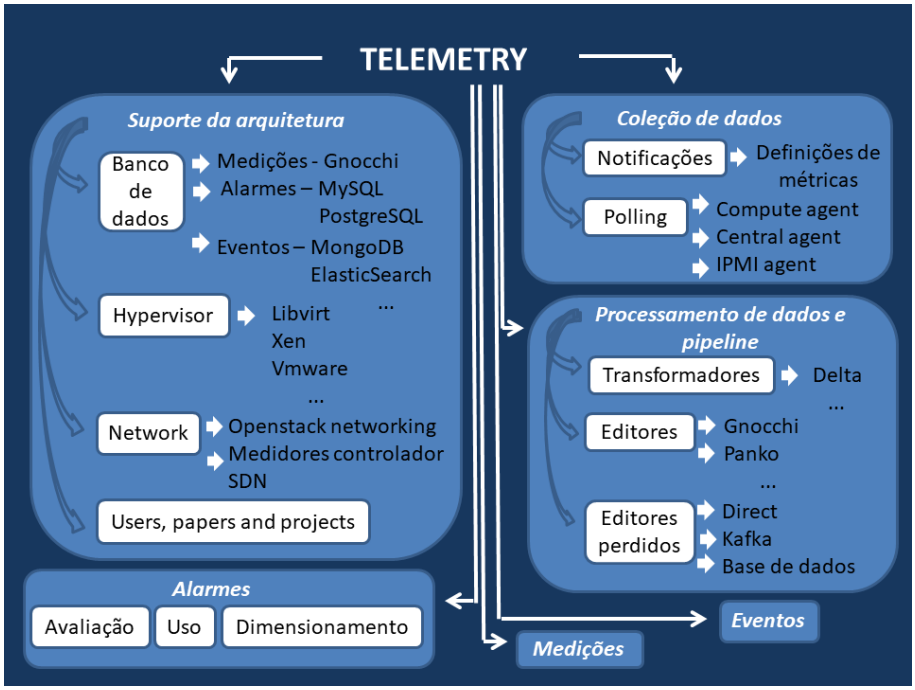

Figura 2: Visão geral do telemetry, com seções que serão abordadas no texto.

3.1 Suporte da Arquitetura

A arquitetura do sistema Telemetry é baseada em agentes. Vários módulos combinam suas responsabilidades para coletar dados, armazenar amostras em um banco de dados ou fornecer um serviço de API para lidar com pedidos recebidos. Os seguintes agentes estão disponíveis:

- Ceilometer-api (obsoleto na versão Ocata)
  Apresenta dados de medição agregados aos consumidores. As APIs de alarme, medidor e evento são agora manipuladas pelos serviços aodh, gnocchi e panko, respectivamente.

- Ceilometer-polling
  Pesquisas para diferentes tipos de dados do medidor, usando os plugins de pesquisa (pesquisadores) em interface única registrados em diferentes namespaces.

  - O "compute namespace"pesquisa o hypervisor local para adquirir dados de desempenho de instâncias locais.
  - O "central namespace"pesquisa as API públicas RESTful de outros serviços OpenStack, como o serviço Compute e de imagem.
  - O "ipmi namespace"identifica o nó local com suporte IPMI, para adquirir dados do sensor IPMI e informações do nível do datahost do Intel Node Manager.

  Os itens acima serão explanados mais a frente ainda neste documento.

- Ceilometer-agent-notification
  Consome mensagens AMQP de outros serviços OpenStack, normaliza mensagens e publica-as para destinos configurados.

- Ceilometer-collector (obsoleto em Ocata)
  Consome notificações AMQP dos agentes, envia esses dados para o armazenamento de dados apropriado. A arquitetura de Telemetria depende muito do serviço AMQP tanto para consumir notificações provenientes de serviços OpenStack como de comunicação interna.

O Telemetry utiliza diversos tipos de banco de dados para o armazenamento de eventos, amostras, definições de alarmes e alarmes. Cada um dos modelos de dados possui seu próprio serviço de armazenamento e cada um suporta vários back-ends, sendo esses tais mostrados na figura 3.

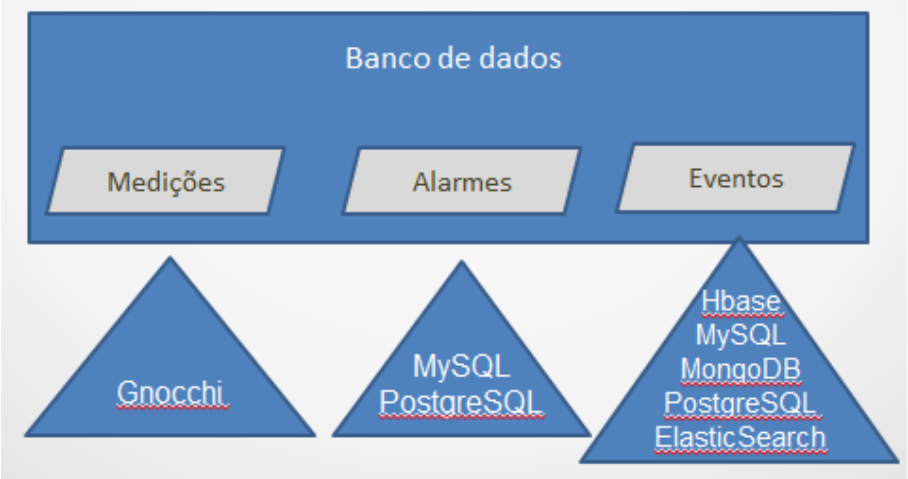

Figura 3: Organização da estrutura do banco de dados do Telemetry

Medições, geralmente representadas através de séries temporais são armazenadas no serviço Gnocchi. Alarmes, representando situações que ocorreram de acordo com determinada expressão lógica, podem ser armazenados em bases de dados relacionais, tais como MySQL ou PostgreSQL. Eventos que ocorram na nuvem podem ser armazenados em uma variedade de bases de dados.

O projeto telemetria suporta a coleta de informações sobre as máquinas virtuais que requer uma conexão próxima com o hypervisor que é executado nos hosts de computação. Para isso, ele suporta as seguintes tecnologias de virtualização:

- Via libvirt:
  - Máquina Virtual baseada em Kernel (KVM)

  - Emulador rápido (QEMU)
  - Linux Containers (LXC)
  - Linux de modo de usuário (UML)
- Hyper-V
- XEN
- VMware vSpheree

Assim como outros projetos OpenStack, o telemetria também utiliza o serviço de identidade para autenticar e autorizar usuários. Alternativamente, o Gnocchi pode ser configurado sem autenticação para evitar sobrecarga. O sistema usa dois papéis: admin e non-admin, as quais definem quantidade de dados retornados. A autorização ocorre antes de processar cada solicitação de API. A criação de definições de alarme também dependem muito do papel do usuário, que iniciou a ação. Configurações necessárias estão na documentação.

O telemetria também permite obter informações de rede, utilizando do serviço OpenStack Networking e de serviços de rede externa, como os listados abaixo:

- ***OpenStack Networking***
  - Medidores de rede básicos
  - Medidores Firewall-as-a-Service (FWaaS)
  - Medidores Load-Balancer-as-a-Service (LBaaS)
  - Medidores VPN-as-a-Service (VPNaaS)
- ***Medidores do controlador SDN***
  - OpenDaylight
  - OpenContrail

3.2 Data collection

O serviço Telemetry aproveita vários métodos para coletar amostras de dados que podem ser guardados na forma de eventos ou amostras no suporte de banco de dados (Figura 4). As amostras capturam uma medida numérica de um recurso. Todos os serviços OpenStack enviam notificações sobre as operações executadas ou o estado do sistema. Exemplo de uma notificação: Tempo de CPU de uma instância de VM.

A seguir alguns mecanismos de coleta de dados:

- Notificações
  Processando notificações de outros serviços OpenStack ao consumir mensagens do sistema de fila de mensagens configurado.
- Polling
  Recupera informações diretamente do hypervisor ou da máquina host usando SNMP, ou usando as APIs de outros serviços OpenStack.
- API RESTful (obsoleta no Ocata)
  Sobe amostras através da API RESTful.

Figura 4: Mecanismos utilizados para a coleta de dados

3.2.1 Notificações

Responsável por consumir notificações. Consome o barramento de mensagens e transforma notificações em eventos e amostras de medição. Responsável por todo o processamento de dados, como transformações e publicação. Após o processamento, os dados são enviados para qualquer alvo de editor suportado, como Gnocchi ou Panko (bancos de dados configurados).

O Serviço Telemetry apenas captura os eventos faturáveis ou criação de perfil a partir das notificações, portanto, nem todas as notificações enviadas dos diferentes serviços são utilizadas. Cada uma possui seu tipo. O agente de notificação filtra o tipo de evento.

A tabela a seguir apresenta um exemplo de tipo de evento de cada serviço do OpenStack que são transformados em amostras.

*Para maiores detalhes, deve-se consultar a documentação.

Serviço OpenStack	***Tipo de evento***
OpenStack Compute	compute.instance.*
Bare metal service	hardware.ipmi.*
OpenStack Image service	image.upload
OpenStack networking	network.exists
Orchestration module	orchestration.stack.creat.end
OpenStack block storage	volume.create.*

Tabela 6: Exemplo de tipo de evento dos serviços OpenStack

As notificações específicas do serviço Compute são importantes para administradores e usuários, permitindo que os administradores respondam rapidamente aos eventos.

- Definições De Medidor
  O serviço Telemetry coleta um subconjunto dos medidores, filtrando notificações de outros serviços. Diferentes configurações do medidor permitem que os operadores/administradores adicionem novas métricas ao projeto, atualizando o arquivo metros.yaml sem alterações de código adicionais, porém, este não deve ter removidas as definições de medidor existentes. Uma mensagem de notificação pode conter várias métricas. Usar * na definição do medidor captura todos os medidores e gera amostras, respectivamente.

3.2.2 Polling

Armazena uma imagem complexa da infra-estrutura e requer informações adicionais das fornecidas. Algumas informações não são emitidas diretamente, como o uso de recursos das instâncias VM e, para reunir esses dados, pesquisa a infra-estrutura, incluindo as APIs dos diferentes serviços e outros recursos, como os hypervisors (que requer uma interação mais próxima com os hosts de computação).

Existem três tipos de agentes que suportam o mecanismo de pesquisa: o agente de cálculo, o agente central e o agente de IPMI. São os mesmos agentes de pesquisa do ceilometer, mas carregam diferentes plug-ins de pesquisa (pesquisadores) de diferentes espaços para coletar dados.

- Compute agent
  - Responsável pela coleta de dados de uso de recursos de instâncias de VM em nós de computação individuais. Este mecanismo requer uma interação mais próxima com o hypervisor, portanto, um tipo de agente separado cumpre a coleta dos medidores relacionados, que é colocado nas máquinas host para recuperar essas informações localmente.
  - Uma instância de Compute agent deve ser instalada em cada nó de computação.
  - Não precisa de conexão direta de banco de dados. As amostras coletadas por este agente são enviadas via AMQP para o agente de notificação a ser processado.
  - Usa a API do hypervisor instalado nos hosts de computação. Portanto, os medidores suportados podem ser diferentes no caso de cada back-end de virtualização, pois cada ferramenta de inspeção fornece um conjunto diferente de medidores.
  - A lista de medidores coletados pode ser encontrada no OpenStack Compute.
- Agente Central
  - Pesquisa de APIs REST públicas para recuperar informações adicionais sobre os recursos que ainda não surgiram através de notificações e também para pesquisar recursos de hardware através de SNMP. Os seguintes serviços podem ser consultados com este agente:
    - OpenStack Networking
    - Armazenamento de objetos OpenStack
    - OpenStack Block Storage
    - Recursos de hardware via SNMP
    - Medidores de consumo de energia através do framework Kwapi (obsoleto em Newton)
  - Assim como o agente de computação, este componente também não precisa de uma conexão direta de banco de dados. As amostras são enviadas via AMQP para o agente de notificação.
- Agente IPMI
  - Coleta dados do sensor IPMI e dados do Intel Node Manager em nós de computação individuais dentro de uma implantação do OpenStack. Este agente requer um nó compatível com IPMI com o utilitário ipmitool instalado, que é comumente usado para controle IPMI em várias distribuições Linux.
  - Uma instância do agente IPMI pode ser instalada em cada nó de computação com suporte (sem suporte retorna vazio), exceto quando o nó é gerenciado pelo serviço de Bare Metal. Sugere-se instalação somente em um nó compatível por motivos de desempenho.

- A lista de medidores coletados pode ser encontrada no serviço de Bare metal.
- Não implantar o agente IPMI e o serviço Bare metal em um nó de computação pois causa amostras de sensores IPMI duplicados.

3.3 Processamento de dados e pipelines

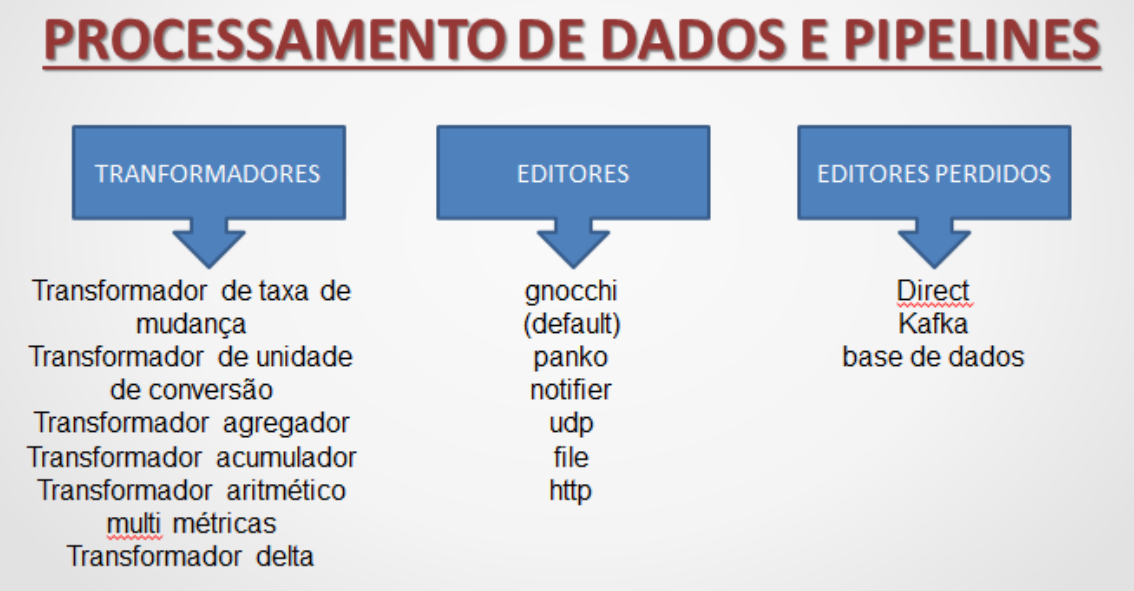

Figura 5: Recursos para processamento de dados: transformadores e editores

O mecanismo pelo qual os dados são processados denomina-se *pipeline*. Esses descrevem um acoplamento entre fontes de dados (amostras ou eventos) e os coletores correspondentes para transformação e publicação de dados. É tratado pelos agentes de notificação.

Um conjunto de manipuladores de notificação que emite alguns exemplos de dados para um conjunto de medidores e tipos de eventos correspondentes chama-se *fonte*.

Cada configuração de origem encapsula correspondência de nomes e mapeamento para uma ou mais *sinks* para publicação. Um *sink* é um consumidor de dados, fornecendo lógica para a transformação e publicação de dados emitidos por fontes relacionadas.

Sink descreve uma cadeia de manipuladores. A cadeia começa com transformadores e termina em editores. O primeiro transformador na cadeia são dados passados da fonte correspondente, leva algumas ações, como derivação da taxa de mudança, realização da conversão da unidade ou agregação, antes da publicação.

3.3.1 Transformadores

Definição de transformadores: Aceita a passagem do nome e dos parâmetros. A seção de parâmetros pode conter campos específicos do transformador, como campos de origem e de destino com diferentes subcampos em caso de taxa de alteração, que depende da implementação do transformador.

Transformadores suportados:

1. Transformador de taxa de mudança

Transformador que calcula a mudança de valor entre dois pontos de dados no tempo. Gera: Medidor cpu_util dos valores de amostra do contador de cpu, que representa o tempo acumulado de CPU em

nanosegundos. 2. Transformador de unidade de conversão
Aplica uma unidade de conversão. Leva o volume do medidor e o multiplica com a expressão da escala dada.

3. Transformador agregador
Resume as amostras recebidas até chegarem amostras suficientes ou um tempo limite foi alcançado. O volume da amostra criada é a soma dos volumes de amostras que entraram no transformador.

4. Transformador acumulador
Armazena as amostras até chegarem amostras suficientes e, em seguida, esvazia todas abaixo da linha de uma só vez

5. Transformador aritmético multi métricas
Permite realizar cálculos aritméticos sobre um ou mais medidores e / ou seus metadados. Por exemplo: Memory_util = 100 * memory.usage / memory Uma nova amostra é criada com as propriedades descritas na seção de destino da configuração do transformador. O volume da amostra é o resultado da expressão fornecida. O cálculo é realizado em amostras do mesmo recurso. O cálculo é limitado á métricas com o mesmo intervalo.

6. Transformador delta
Calcula a mudança entre dois pontos de dados de amostra de um recurso. Ele pode ser configurado para capturar apenas os deltas de crescimento positivo.

3.3.2 Editores

Existem vários métodos de transporte para transferir os dados coletados para um sistema externo. Os consumidores desses dados são amplamente diferentes, como sistemas de monitoramento, para os quais a perda de dados é aceitável e os sistemas de cobrança, que exigem um transporte de dados confiável.
O componente do editor permite salvar os dados no armazenamento persistente através do barramento de mensagens ou enviá-lo para um ou mais consumidores externos. Uma cadeia pode conter vários editores. Para resolver este problema, o multi-editor pode ser configurado para cada ponto de dados dentro do serviço, permitindo que o mesmo medidor ou evento técnico seja publicado várias vezes em vários destinos, cada um potencialmente usando um transporte diferente.

Os editores são especificados para cada pipeline: pipeline.yaml e event_pipeline.yaml.

- Gnocchi (padrão): Quando é habilitado, as informações de medição e recursos são empurradas para Gnocchi para armazenamento otimizado de séries temporais. Deve ser registrado no serviço de Identidade, pois o Ceilometer descobre o caminho exato através dele.
- Panko: Dados do evento no Ceilometer podem ser armazenados no panko Fornece uma interface HTTP REST para consultar os eventos do sistema no OpenStack.
- Udp: Emite dados de medição sobre UDP.
- Notificador: Emite dados sobre AMQP. Qualquer consumidor pode se inscrever no tópico publicado para processamento adicional. * Antes da Ocata, o coletor consumiria esta editora, mas já havia sido reprovado e, portanto, não era necessário.
  - Personalização:
    - - Per_meter_topic: O valor desse parâmetro é 1. Ele é usado para publicar as amostras na fila de tópicos adicionais além da fila padrão.

    * - Política: Usado para configurar o comportamento do caso, quando o editor não enviar as amostras, onde os possíveis valores predefinidos são:
        1. Padrão: Usado para espera e bloqueio até que as amostras tenham sido enviadas.
        2. Solta: Usado para deixar cair as amostras que não foram enviadas.
        3. Fila: Usado para criar uma fila na memória e tentar novamente enviar as amostras na fila no próximo período de publicação de amostras tema
    * Tópico: O nome do tópico da fila para publicar. A configuração deste irá substituir o tópico padrão definido pelas opções $metering_topic$ e $event_topic$. Esta opção pode ser usada para suportar vários consumidores.

- Arquivo: Registra a medição de dados em um arquivo.

  * Se um nome e local de arquivo não forem especificados, o editor de arquivos não registrará nenhum medidor, em vez disso, registra uma mensagem de aviso no arquivo de log configurado.

  As seguintes opções estão disponíveis para o editor de arquivos:

  1. Max_bytes: Quando > 0, isso causará uma rolagem. Quando o tamanho especificado está quase excedido, o arquivo é fechado e um novo arquivo é aberto silenciosamente para saída. Se = 0, a rolagem nunca ocorre.
  2. Backup_count: Se for diferente 0, uma extensão será anexada ao nome do arquivo antigo até que o valor especificado seja atingido. O arquivo que está escrito e contém os dados mais recentes é sempre aquele que é especificado sem extensões.

- HTTP: O serviço Telemetry suporta enviar amostras para um alvo HTTP externo. As amostras são enviadas sem qualquer modificação. Opções de configuração disponíveis:
  1. Tempo esgotado = O número de segundos antes da solicitação HTTP expirar.
  2. Max_retries = O nº de vezes para tentar novamente um pedido antes de falhar.
  3. Lote = Se for falso, o editor enviará cada amostra e evento individualmente, independentemente de o agente de notificação estar ou não configurado para processar em lotes.
  4. Agrupado = O número máximo de conexões abertas que o editor manterá. O aumento do valor pode melhorar o desempenho, mas também aumentará os requisitos de consumo de memória e soquete.

3.3.3 Editores Obsoletos

Editoras são obsoletas a partir de Ocata:

1. Direct
2. Kafka: Oferece opções semelhantes a editora de notificadores e envia dados de medição para um corretor kafka.
3. Base de dados: Foi substituída por editores Gnocchi e Panko.Quando o dispatcher do banco de dados está configurado como um armazenamento de dados, existe a opção de definir uma opção time_to_live (ttl) para amostras. Por padrão, o valor ttl para amostras é definido como -1, o que significa que eles são mantidos na base de dados para sempre. Cada amostra tem um carimbo de

hora e o valor ttl indica que uma amostra será excluída do banco de dados quando o número de segundos tiver decorrido desde que a leitura da amostra foi marcada.

* O nível de suporte difere dependendo da configuração de back end

3.4 Alarmes

Os alarmes fornecem o Monitoring-as-a-Service orientado a usuários para recursos em execução. Este tipo de monitoramento garante escalar ou fechar automaticamente um grupo de instânias através do serviço Orchestration, mas você também pode usar alarmes para fins gerais da saúde dos recursos da sua nuvem. Os alarmes seguem um modelo tri-state:

- ok: A regra que governa o alarme foi avaliada como False.
- alarm: A regra que governa o alarme foi avaliada como true.
- insufficient data: Não há suficientes pontos de dados disponíveis nos períodos de avaliação para determinar o estado do alarme.

3.4.1 Definições de alarme

As definições de alarme são regras que regem quando uma transição de estado deve ocorrer e as ações a serem tomadas. A natureza dessas regras depende do tipo de alarme. 1. Alarmes de regra de limite
Para alarmes convencionais, as transições de estado são regidas: - Um valor de limiar estático com um operador de comparação, como < ou >. - Uma seleção estatística para agregar os dados. - Uma janela de tempo deslizante para indicar o quão longe o passado recente pode ser visto.

* A partir de Ocata, o alarme de limiar é obsoleto uma vez que a API Ceilometer está obsoleta.

2. Alarmes de regras compostos
Permite aos usuários definição de alarme com múltiplas condições de disparo, usando uma combinação de relações de e e ou.

3. Alarmes de regra de combinação * Os alarmes combinados são obsoletos a partir de Newton para alarmes compostos. A funcionalidade de combinação de alarme é removida no Pike. O telemetry também suporta o conceito de um meta-alarme, que se agrega sobre o estado atual de um conjunto de alarmes básicos subjacentes combinados através de um operador lógico (e ou ou).

3.4.2 Dimensionamento de alarme

Define o conjunto de medidores correspondentes que se alimentem em uma avaliação de alarme. Os medidores são por instância de recurso, portanto, no caso mais simples, um alarme pode ser definido em um determinado medidor aplicado a todos os recursos visíveis para um determinado usuário. De um lado, alarmes dimensionados de forma restrita, onde essa seleção teria apenas um único alvo (identificado por ID de recurso). De outro, alarmes amplamente dimensionados, que identifica muitos recursos sobre os quais a estatística é agregada. Por exemplo, todas as instâncias iniciadas a partir de uma determinada imagem ou todas as instâncias com metadados de usuários correspondentes (o último é como o serviço Orchestration identifica grupos de escala automática).

3.4.3 Avaliação de alarme

Os alarmes são avaliados periodicamente, desativando uma vez a cada minuto.

1. Ações de alarme
   Qualquer transição de estado do alarme individual (para ok, alarme ou dados insuficientes) pode ter uma ou mais ações associadas, que enviam um sinal ao consumidor que a transição do estado ocorreu e fornece algum feedback, incluindo estados novos e anteriores, com alguns dados de razão que descrevem a disposição em relação ao limite, o número de pontos de dados envolvidos e os mais recentes. As transições de estado são detectadas pelo avaliador de alarme, enquanto o notificador de alarme efetua a ação de notificação real. - Webhooks: Tipo de notificação de fato usado por Telemetry alarmante e simplesmente envolve uma solicitação HTTP POST sendo enviada para um ponto final, com um corpo de solicitação contendo uma descrição da transição de estado codificada como um fragmento JSON.

   Ações de registo: Alternativa leve para webhooks, em que a transição de estado é registrada pelo notificador de alarme e é destinada principalmente a fins de teste.

2. Particionamento da carga de trabalho
   O processo de avaliaço de alarme usa o mesmo mecanismo para particionamento de carga de trabalho como agentes centrais e de compute. A biblioteca Tooz fornece a coordenação dentro dos grupos de instâncias de serviço. Para usar esta solução de particionamento de carga de trabalho, defina a opção de $avaliao_servio$ como padrão.

3.4.4 Usando alarmes

Criação de alarme: Atributos de alarme que podem ser configurados na criação, numa atualização:

- Estado: Estado inicial do alarme (padrão para dados insuficientes).
- Description: Descrição de texto livre do alarme (padrão para uma sinopse da regra de alarme).
- Enabled: True se a avaliação e a ação estiverem ativadas (padrão).
- Repeat-actions: True se as ações devem ser notificadas repetidamente enquanto permanece no estado alvo (padrão False).
- Ok-action: chamar quando o estado muda para ok.
- Insufficient-data-action: chamar quando o estado muda para dados insuficientes.
- Time-constraint: Restringe a avaliação a determinadas horas do dia ou dias da semana (expressa como cron com um fuso horário opcional).

3.4.5 Recuperação de alarmes

Quando o estado é dado como dados insuficientes ou os medidores ainda não foram coletados na janela de avaliação no passado recente (por exemplo, uma nova instância), ou, que a instância identificada não é visível para o usuário / projeto que possui o alarme, Ou simplesmente que um ciclo de avaliação de alarme não foi iniciado desde que o alarme foi criado (por padrão, os alarmes são avaliados uma vez por minuto). * A visibilidade dos alarmes depende da função e do projeto associados ao usuário que emite a

consulta: Os usuários de administração vêem todos os alarmes, independentemente do proprietário. Os usuários não gerenciados vêem apenas os alarmes associados ao projeto (de acordo com a segregação normal do projeto no OpenStack).

3.4.6 Atualização de alarme

Depois de estabelecido o limite do alarme, caso seja considerado baixo, pode ser atualizado, com a mudança valendo a partir do ciclo seguinte (padrão a cada minuto). A maioria dos atributos de alarme podem ser alterados dessa maneira. Com o tempo, o estado do alarme pode mudar frequentemente, especialmente se o limite for escolhido para estar próximo do valor de tendência da estatística. Pode seguir o histórico do alarme ao longo do seu ciclo de vida através da API de auditoria

3.4.7 Eliminação de alarme

Um alarme que não é mais necessário pode ser desativado para que ele não seja mais ativamente avaliado ou excluído permanentemente.

3.5 Medições

Esta seção traz um resumo sobre o formato e a origem dos medidores e também a lista de medidores disponíveis. Coleta medidores pesquisando os elementos de infra-estrutura e consumindo as notificações emitidas por outros serviços.

Existem várias metricas que são coletados por votação e pelo consumo. A origem para cada medidor está listada nas tabelas abaixo.

Tipo	Descrição
Cumulative	Aumento cumulativo ao longo do tempo (horas da instância)
Delta	Mudando ao longo do tempo (largura de banda)
Gauge	Itens discretos (IPs flutuantes, carregamentos de imagens) e valores flutuantes (E/S de disco)

Tabela 7: Origem dos medidores

Há possibilidade de armazenar metadata por amostras, que podem ser estendidos para OpenStack Compute e OpenStack Object Storage.

Dois métodos para adicionar informações de metadata ao OpenStack Compute:

1. Especificá-los quando inicia uma nova instância. As informações adicionais serão armazenadas com a amostra. O novo campo deve ser definido usando o prefixo metering.

2. Configurar as chaves na lista de chaves de metadata que gostaria de incluir no resource_metadata das amostras coletada relacionadas à instância.

Pode-se especificar cabeçalhos cujos valores serão armazenados juntamente com os dados de exemplo do OpenStack Object Storage. As informações adicionais também são armazenadas em resource_metadata. Para especificar o novo cabeçalho, define-se a opção metadata_headers na seção [filter: ceilometer] no proxy-server.conf sob a pasta swift. Tais dados adicionais servem, por exemplo, para distinguir usuários externos e internos. * As medições são agrupadas por serviços que são consultados pela Telemetria ou emite notificações que esse serviço consome.

Telemetry suporta armazenar notificações como eventos, porém, a lista de medidores ainda contém tipo de existência e outros itens relacionados ao evento. Maneira correta é configurando-o para usar a loja de eventos e desligar a coleção dos medidores relacionados ao evento.

3.5.1 OpenStack Compute

Exemplos de medidores coletados para OpenStack Compute: (Mitaka ou anterior) Existem medidores adicionados no Newton e adicionados e removidos no Ocata. Telemetry suporta a criação de novos medidores usando transformadores.

Nome	Tipo	Unidade	Recurso	Origem	Support	Nota
memory	Gauge	MB	instance ID	Notificações	Libvirt, Hyper-V	Volume de RAM alocado para a instância
disk.capacity	Gauge	B	instance ID	Pollster	Libvirt	A quantidade de disco ocupada pela instância na máquina host
vcpus	Gauge	vcpu	instance ID	Notificações		Nº de VCPUs alocadas para a instância

Tabela 8: Medidores para OpenStack Compute

Entre os medidores reunidos da libvirt e do Hyper-V, existem alguns que são gerados a partir de outros medidores. Exemplos de medidores criados usando o transformador rate_of_change da tabela acima é a seguinte:

cpu_util cpu.delta disk.read.requests.rate

* Para ativar o suporte libvirt memory.usage, precisa-se instalar libvirt version 1.1.1+, QEMU versão 1.5+, e também preparar um driver de balão adequado na imagem. É aplicável particularmente para Windows, a maioria das distribuições Linux já o construíram. A telemetria não é capaz de buscar amostras de memória.usagem sem o driver do balão da imagem.

* Para habilitar o libvirt disk.* support, quando executado no armazenamento compartilhado com respaldo RBD, precisa instalar o libvirt version 1.2.16+.

3.5.2 Bare Metal Service

Telemetry captura notificações que são emitidas pelo serviço de bare metal. A fonte das notificações são sensores IPMI que coletam dados da máquina host.

* Os dados do sensor não estão disponíveis no serviço de bare metal por padrão. Para habilitar e configurar este módulo, deve-se instalar serviço de bare metal.

3.5.3 Ipmi based meters

Outra maneira de reunir dados baseados em IPMI é usar sensores IPMI independentemente dos componentes do serviço de bare metal. As mesmas métricas do serviço de bare metal podem ser obtidas com

Nome	Tipo	Unidade	Recursos	Origem	Nota
hardware.ipmi.fan	Gauge	RPM	Sensor do ventilador	Notificações	Rodadas do ventilador por minuto (RPM)
hardware.ipmi.temperature	Gauge	C	Sensor de temperatura	Notificações	Leitura da temperatura
hardware.ipmi.current	Gauge	W	Sensor de corrente	Notificações	Leitura da corrente
hardware.ipmi.voltage	Gauge	V	Sensor de tensão	Notificações	Leitura da tensão

Tabela 9: Exemplo de medidores para o serviço de bare metal (Mitaka ou anterior)

a origem Pollster. Necessário implantação de ceilometer-agent-ipmi em cada nó compatível com IPMI para pesquisar dados de sensores locais.

* Para evitar a duplicação de dados de medição, não deve-se implantar o agente IPMI em nós gerenciados pelo serviço Bare metal e manter a opção $conduct.send_sensor_data$ configurada como False no arquivo de configuração ironic.conf. Além dos dados genéricos do sensor IPMI, medidores Intel Node Manager são gravados a partir da plataforma capaz. Alguns deles:

Nome	Tipo	Unidade	Recursos	Origem	Nota
hardware.ipmi.node.power	Gauge	W	host ID	Pollster	Potência atual do sistema
hardware.ipmi.node.temperature	Gauge	C	host ID	Pollster	Temperatura atua do sistema
hardware.ipmi.node.outlet_temperature	Gauge	C	host ID	Pollster	Temperatura de saída do sistema

Tabela 10: Exemplos de medidores para IPMI

3.5.4 SNMP based meters

Telemetry suporta a recolha de medidores de hospedagem genéricos baseados em SNMP. Para poder coletar esses dados, precisa executar snmpd em cada host alvo.

Alguns dos medidores disponíveis sobre as máquinas host usando o SNMP:

Nome	Tipo	Unidade	Recursos	Origem	Nota
hardware.cpu.util	Gauge	%	host ID	Pollster	Porcentagem de uso da CPU
hardware.disk.size.total	Gauge	KB	disk ID	Pollster	Tamanho total do disco
hardware.memory.used	Gauge	KB	host ID	Pollster	Tamanho da memória física usada

Tabela 11: Exemplos de medidores para SNMP

3.5.5 OpenStack Image Service

Os seguintes contadores são coletados para o serviço OpenStack Image:

Nome	Tipo	Unidade	Recursos	Origem	Nota
image.size	Gauge	image	image ID	Notificações, pollster	Tamanho da imagem carregada
image.update	Delta	image	image ID	Notificações	Número de atualizações na imagem
image.delete	Delta	image	image ID	Notificações	Número de exclusões de imagem

Tabela 12: Exemplo de medidores para OpenStack Image

3.5.6 OpenStack block storage

1. Adicionado em Mitaka ou anterior

Nome	Tipo	Unidade	Recursos	Origem	Nota
volume.size	Gauge	GB	volume ID	Notificações	Tamanho do volume
snapshot.size	Delta	GB	snapshot ID	Notificações	Tamanho do snapshot

Tabela 13: Exemplo de medidores para OpenStack block storage adicionados em Mitaka ou anterior

2. Removidos a partir de Ocata

Nome	Tipo	Unidade	Recursos	Origem	Nota
volume.create.(start|end)	Delta	volume	volume ID	Notificações	Criação de volume
volume.resize(start|end)	Delta	volume	volume ID	Notificações	Atualização do tamanho do volume
volume.delete(start|end)	Delta	volume	volume ID	Notificações	Eliminação do volume

Tabela 14: Exemplo de medidores para OpenStack block storage removidos a partir de Ocata

3.5.7 OpenStack object storage

Os seguintes contadores são coletados para OpenStack Object Storage:

Nome	Tipo	Unidade	Recursos	Origem	Nota
storage.objects.containers	Gauge	container	storage ID	Pollster	Número de containers
storage.containers.objects	Gauge	object	storage ID/container	Pollster	Número de objetos no container
storage.containers.objects.size	Gauge	B	storage ID/container	Pollster	Tamanho total dos objetos armazenados no container

Tabela 15: Exemplos de medidores para OpenStack object storage

3.5.8 Ceph object storage

(Mitaka ou anterior) Para reunir medidores da Ceph, precisa instalar e configurar o Ceph Object Gateway (radosgw). Deve-se habilitar o log de uso para obter os medidores relacionados, além de um usuário administrador com usuários, metadados e caps de uso configurados. Para acessar o Ceph da Telemetria, precisa-se especificar um grupo de serviços para o radosgw no arquivo de configuração ceilometer.conf juntamente com access_key e secret_key do usuário de administração mencionado acima.

Nome	Tipo	Unidade	Recursos	Origem	Nota
radosgw.objects.size	Gauge	B	storage ID	Pollster	Tamanho total dos objetos armazenados
radosgw.objects.containers	Gauge	container	storage ID	Pollster	Número de containers
radosgw.containers.objects	Gauge	object	storage ID/container	Pollster	Número de objetos no container

Tabela 16: Exemplos de medidores para ceph object storage

* As informações relacionadas ao uso podem não ser atualizadas logo após o upload ou download, porque o Ceph Object Gateway precisa de tempo para atualizar as propriedades de uso. Por exemplo, a configuração padrão precisa de aproximadamente 30 minutos para gerar os logs de uso.

3.5.9 OpenStack Identity

Medidores coletados:

Nome	Tipo	Unidade	Recursos	Origem	Nota
identity.authenticate.success	Delta	user	user ID	Notificações	Usuário autenticado com sucesso
identity.authenticate.pending	Delta	user	user ID	Notificações	Usuário com autenticação pendente
identity.authenticate.failure	Delta	user	user ID	Notificações	Usuário falhou ao autenticar

Tabela 17: Exemplos de medidores para OpenStack Identity

3.5.10 OpenStack Networking

Sendo os 3 últimos removidos em Ocata

Nome	Tipo	Unidade	Recursos	Origem	Nota
Largura de banda	Delta	B	label ID	Notificações	Bytes através do rótulo de medição I3
network.create	Delta	network	network ID	Notificações	Pedidos de criação para esta rede
ip.floating.create	Delta	ip	ip ID	Notificações	Pedidos de criação para este IP
router.create	Delta	router	router ID	Notificações	Pedidos de criação para este roteador

Tabela 18: Exemplos de medidores para OpenStack Networking

3.5.11 SDN controllers

Esses medidores estão disponíveis para switches baseados em OpenFlow. Para habilitar esses medidores, cada driver precisa ser configurado corretamente.

Nome	Tipo	Unidade	Recursos	Origem
switch.port.transmit.packets	Cumulative	packet	switch ID	Pacotes transmitidos por porta
switch.port.receive.bytes	Cumulative	B	switch ID	Bytes recebidos por porta
switch.port.transmit.bytes	Cumulative	B	switch ID	Bytes transmitidos por porta

Tabela 19: Exemplo de medidores para controladores SDN

3.5.12 LOAD-BALANCER-AS-A-SERVICE (LBAAS V1 e LBAAS V2)

Alguns medidores foram retirados em Ocata Os medidores acima são experimentais e podem gerar uma

Nome	Tipo	Unidade	Recursos	Origem
network.services.lb.pool	Gauge	pool	pool ID	Existência de LB pool
network.services.lb.vip (ou listener)	Gauge	Vip (listener)	Vip (listener) ID	Existência de LB VIP (listener)
network.services.lb.member	Gauge	member	member ID	Existência de LB member

Tabela 20: Exemplo de medidores para load-balancer-as-a-service (lbaas v1 e lbaas v2)

grande carga contra as API Neutron. Quando este suportar novas APIs, haverá aprimoramento dos medidores.

3.5.13 VPN-AS-A-SERVICE (VPNAAS)

Medidores foram removidos em Ocata

Nome	Tipo	Unidade	Recursos	Origem
network.services.vpn	Gauge	vpnservice	vpn ID	Existência de VPN
network.services.vpn.connections	Gauge	ipsec_site_connection	connection ID	Existência de conexão IPSec

Tabela 21: Exemplos de medidores para vpn-as-a-service(vpnaas)

Nome	Tipo	Unidade	Recursos	Origem	Nota
identity.authenticate.success	Gauge	firewall	firewall ID	Pollster	Existência de firewall
network.services.firewall.policy	Gauge	firewall_policy	firewall ID	Pollster	Existência de política de firewall

Tabela 22: Exemplo de medidores para firewall-as-a-service(fwaas)

3.5.14 FIREWALL-AS-A-SERVICE (FWAAS)

3.5.15 Orchestration service

Os seguintes contadores foram previamente coletados para o serviço Orchestration: (Removidos a partir de Ocata)

Nome	Tipo	Unidade	Recursos	Origem	Nota
stack.create	Delta	stack	stack ID	Notificações	Stack foi criada com sucesso
stack.update	Delta	stack	stack ID	Notificações	Stack foi atualizada com sucesso
stack.delete	Delta	stack	stack ID	Notificações	Stack foi deletada com sucesso

Tabela 23: Exemplos de medidores para Orchestration service

3.5.16 Serviço de processamento de dados para OpenStack

Contadores previamente coletados para o serviço de processamento de dados: (Removidos a partir de Ocata)

Nome	Tipo	Unidade	Recursos	Origem	Nota
cluster.create	Delta	cluster	cluster ID	Notificações	Cluster foi criado com sucesso
cluster.update	Delta	cluster	cluster ID	Notificações	Cluster foi atualizado com sucesso
cluster.delete	Delta	cluster	cluster ID	Notificações	Cluster foi deletado com sucesso

Tabela 24: Exemplo de medidores para processamento de dados

3.5.17 Key value store module

Contadores foram previamente coletados: (Removidos a partir de Newton)

Nome	Tipo	Unidade	Recursos	Origem	Nota
magnetodb.table.create	Gauge	table	table ID	Notificações	Tabela criada com sucesso
magnetodb.table.delete	Gauge	table	table ID	Notificações	Tabela deletada com sucesso
magneto.table.index.count	Gauge	index	table ID	Notificações	Nº de índices criados em uma tabela

Tabela 25: Exemplo de medidores para key value store module

3.6 Eventos

Um evento é o estado de um recurso em um ponto no tempo que pode ser descrito usando vários tipos de dados, incluindo dados não-numéricos, como o flavor de uma instância. Resumidamente, são qualquer ação feita no sistema. São emitidos todos os dados que alguém possa precisar e o consumidor filtra o que não interessa. Para simplificar o processo, as notificações são armazenadas e processadas no Ceilometer como Eventos.

- Estrutura de evento Os eventos são representados por cinco atributos-chave:
  1. event_type: Uma string que define o evento que ocorreu.
  2. Message_id: Um UUID para o evento.
  3. Generated: Um timestamp de quando o evento ocorreu no sistema.
  4. Traits: Um mapeamento plano de pares de valores-chave que descrevem o evento. Os traços do evento contêm a maioria dos detalhes do evento (podem ser strings, inteiros, flutuadores ou datetimes)
  5. Raw: Principalmente para auditoria, a mensagem de evento completo pode ser armazenada (não indexada) para avaliação futura.
- Indice de eventos
  A carga útil da notificação, que pode ser uma estrutura de dados JSON arbitrariamente complexa, é convertida em um conjunto plano de pares chave-valor. Essa conversão é especificada por um arquivo de configuração.* O formato do evento é destinado a processamento e consulta eficientes

4 Melhores práticas

4.1 Coleção de dados

1. Nem todos os dados são necessários, por isso, a configuração de quantidade de medidores é feita em pipeline.yaml.

* Nesse arquivo pode ser alterado o intervalo de pesquisa em base por métrica, pois são examinadas as APIs de serviço a cada 10 minutos. Se o intervalo de pesquisa for muito curto, provavelmente aumentará o estresse nas APIs de serviço.

* Em Ocata, precisa editar polling.yaml para definir quais metros gerar.

Expandir a configuração para ter maior controle sobre diferentes intervalos de medição.

2. Pode-se ajustar solicitações de polling, permitindo o suporte de jitter. Isso adiciona um atraso aleatório sobre como os agentes de pesquisa enviam solicitações às APIs do serviço.

3. Pode-se adicionar outros agentes centrais e de computação, quando necessário. Os agentes são projetados para dimensionar horizontalmente.

4.2 Armazenamento de dados

* A partir de Newton, o armazenamento de dados não é recomendado no ceilometer. Os dados de alarme, métrica e evento devem ser armazenados em aodh, gnocchi e panko, respectivamente.

Sobre ceilometer: 1. Deve-se evitar consultas abertas. Para melhor desempenho, usar intervalos de tempo razoáveis ou outras restrições de consulta para recuperar medições. * O número de itens retornados será restrito ao valor definido por default_api_return_limit no arquivo de configuração ceilometer.conf. O valor também pode ser definido por consulta passando a opção limite em solicitação.

2. Recomendado instalar a API atrás do mod_wsgi, pois fornece mais configurações para ajustar, como threads e processos no caso do WSGIDaemon.

3. O serviço de coleta não é um serviço de arquivamento. Deve-se definir um valor de Time to Live (TTL) para expirar dados e minimizar o tamanho do banco de dados. Para manter dados por mais tempo, pode armazená-los em um data warehouse fora da Telemetry.

4. É recomendado executar o MongoDB num nó separado otimizado para armazenamento rápido para um melhor desempenho e que este tenha muita memória.

5. Usar conjuntos de réplicas no MongoDB. Os conjuntos de réplicas fornecem alta disponibilidade através de failover automático. Se o seu nó principal falhar, o cluster permanece funcional alternando para um nó secundário

6. Usar sharding em MongoDB, armazenando registros de dados em várias máquinas e é a abordagem para atender às demandas de crescimento de dados.

5 Monasca

O Monasca é uma solução open-source que se integra ao OpenStack. Usa uma API REST para processamento, consulta métricas com alta velocidade e possui um mecanismo de alarme de transmissão e mecanismo de notificação. Fica fácil a visualização das subseções tratadas nessa seção: Subseção 5.1 descreve todas as condições de uso do sistema. A Subseção 5.2 apresenta um relato sobre as características principais. A Subseção 5.3 define os componentes do software, suas propriedades e seu relacionamento com outros elementos. A Subseção 5.4 mostra como são estabelecidos os eventos. A Subseção 5.5 traz a descrição de todas as condições de uso do sistema keystone. A Subseção 5.6 trata da gestão de logs A Subseção 5.7 diz o ambiente de desenvolvimento. A Subseção 5.8 aborda a publicação de métricas, encerrando essa seção.

5.1 Requisitos

1. Monitoramento da infraestrutura e os recursos do OpenStack
   Monitora a infraestrutura OpenStack (CRUD usando StackTach) e seus recursos (por exemplo, VM) para que outra ferramenta de monitoramento não seja necessária.

2. Can Scale
   Para centenas de milhares de servidores, métricas, alarmes

3. Criação de métricas e alarmes definidas pelo usuário que é administrador de sistema da nuvem ou um consumidor de nuvem e define métricas e alarmes em um grupo de recursos (por exemplo, agregados de host)

4. Multi-inquilino, multi-nuvem, modelos de implantação múltipla permitem processar todas essas várias métricas. Exemplo de implantação:
   - Nuvem privada única (por exemplo, 10 nós)
   - Múltiplas nuvens privadas
   - Nuvem privada e pública (híbrida)

5. Tem uma API extensível baseada em HTTP para métricas, criação de alarmes e coleta métrica.

6. Autentica com keystone ou com outros backends (por exemplo, CloudFoundry, Open LDAP, AWS)

7. Fornece um leque de backends plugáveis: vertica, influx DB, Cassandra, MySQL, etc.

8. Uso do Grafana como interface gráfica

5.2 Características - descrevendo recursos gerais

Uma solução de Monitoramento como Serviço (MONaaS), altamente compatível, escalável, confiável e tolerante a falhas, que se estende aos níveis de métricas do provedor de serviços de taxa de transferência de métricas. Processa milhares de métricas por segundo como também guarda dados por período superior a um ano sem perda enquanto processam consultas interativas. Usa Rest API para armazenar e consultar métricas e informações históricas. O http é o único protocolo utilizado permitindo melhor descrição dos dados através de dimensões. Essas dimensões são conjunto de pares (chave, valor) que definem métricas.As métricas são enviadas e autenticadas usando Keystone e armazenadas associadas a um ID de tenant. Possui um agente de monitoramento que suporta uma série de verificações de sistema e serviço integradas e do Nagios. O monitoramento de fonte aberta é baseado em open-source.

5.3 Arquitetura do sistema

Os vários itens apresentados na figura serão explicados a seguir:

- Agente de Monitoramento (monasca-agent): Consiste em vários subcomponentes e suporta métricas do sistema, como a utilização de CPU e memória disponível, plugins Nagios, statsd e muitas verificações para serviços como o MySQL, RabbitMQ, e muitos outros.

- API de monitoramento (monasca-api): Uma API RESTful bem definida para o monitoramento principalmente dos seguintes conceitos e áreas:
  - Métricas: armazena e pesquisa enormes quantidades de métricas em tempo real.
  - Estatísticas: estatísticas de consulta para métricas.
  - Definições de alarme: cria, atualiza, pesquisa e exclui as definições de alarme.
  - Alarmes: consulta e exclui o histórico de alarmes.
    * Gramática expressiva simples para criar alarmes compostos feitos por subexpressões de alarme e operadores lógicos.

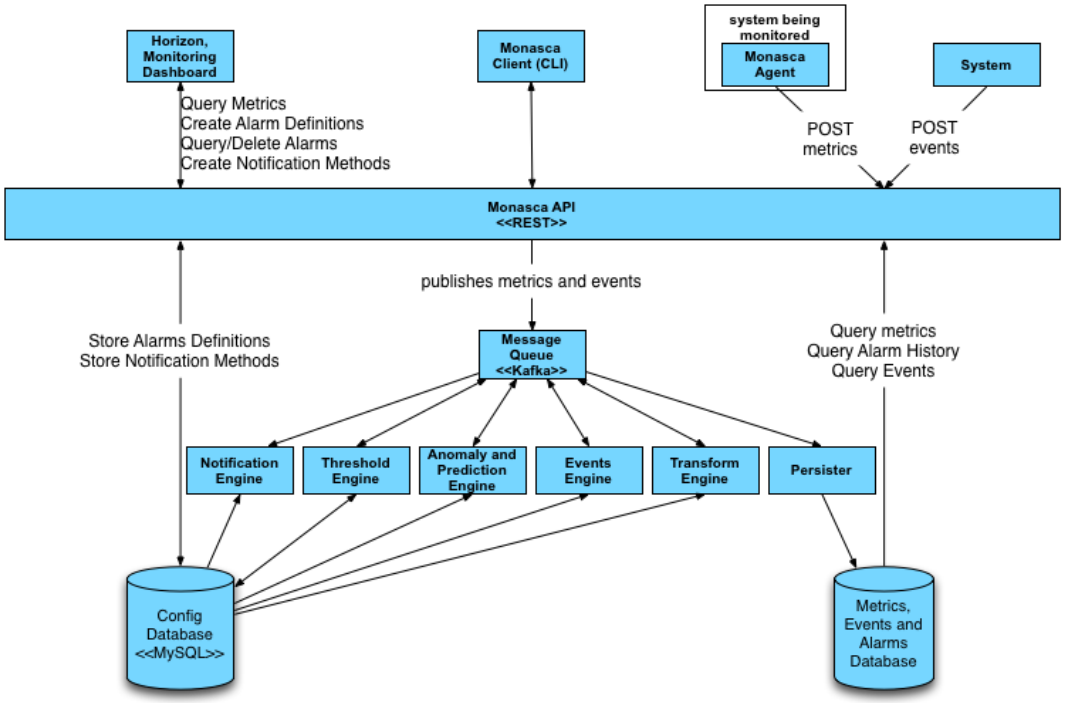

Figura 6: Arquitetura do Monasca

  - ∗ As alarms severities podem ser associadas a alarmes.
  - ∗ O histórico completo de transição do estado do alarme é armazenado permitindo a posterior análise de causa raiz (RCA) ou análises avançadas.
- Métodos de notificação: cria e exclui métodos de notificação e associa-os a alarmes, como e-mail. Pode notificar usuários diretamente via e-mail quando ocorrem transições de estado de alarme.

- Persister (monasca-persister): Consome métricas e transições de estado de alarme do MessageQ e as armazena no banco de dados de métricas e alarmes.
- Mecanismo de transformação e agregação (monasca-transform): Transforma nomes e valores métricos, como cálculos derivativos delta ou baseados em tempo e cria novas métricas que são publicadas no Message Queue.
- Motor de Previsão e Anomalia: Avalia a previsão e as anomalias e gera métricas previstas, bem como a probabilidade de anomalia e os escores desta.
- Threshold Engine (monasca-thresh): calcula limiares em métricas e publica alarmes para o MessageQ quando excedido. Com base em Apache Storm, um sistema de computação em tempo real distribuído livre e aberto.
- Mecanismo de Notificação (notificação monasca): Consome mensagens de transição de estado de alarme do MessageQ e envia notificações para alarmes como emails.
- Mecanismo de análise (monasca-analytics): Consome transições e métricas de estado de alarme do MessageQ e detecta anomalia e agrupamento/correlação de alarmes.

- Message Queue: recebe métricas publicadas da API de Monitoramento e mensagens de transição de estado de alarme do Threshold Engine que são consumidas por outros componentes, como o Persister. Também é usado para publicar e consumir outros eventos no sistema. Atualmente, um MessageQ baseado em Kafka é suportado. A Kafka é uma fila de mensagens de alto desempenho, distribuída, tolerante a falhas e escalável com durabilidade incorporada.

- Banco de Dados de Métricas e Alarmes: armazena principalmente métricas e o histórico de estado do alarme. Atualmente, o Vertica e o InfluxDB são suportados.

- Configuração de Database: armazena muita configuração e outras informações no sistema. Atualmente, o MySQL é suportado.

- Cliente de Monitoramento (python-monascaclient): Um cliente e biblioteca de linha de comando Python que comunica e controla a API de Monitoramento. Possui uma biblioteca "monascaclient"semelhante aos outros clientes OpenStack, que podem ser usados para criar rapidamente recursos adicionais. A biblioteca do Cliente de Monitoramento é usada pela UI de monitoramento, editor de ceilômetro e outros componentes.

- Monitorando UI: um painel do Horizon para visualizar a saúde geral e o status de uma nuvem OpenStack.

- Editor de ceilômetro: um complemento de vários publisher para Ceilometer, não mostrado, que converte e publica amostras para a API de Monitoramento.

5.3.1 Banco de dados de métricas e alarmes

Um banco de dados de análise de alto desempenho que pode armazenar enormes quantidades de métricas e alarmes em tempo real e também oferecer suporte a consultas interativas. O esquema SQL que é usado pela Vertica é o seguinte:

- MonMetrics.Measurements: armazena as medidas reais que são enviadas.

- MonMetrics.DefinitionDimensions

- MonMetrics.Definitions

- MonMetric.Dimensions

5.3.2 Messages Queue

Fila de mensagens distribuídas, performantes, escaláveis e HA para distribuição de métricas, alarmes e eventos no sistema de monitoramento. Atualmente, com base em Kafka.

- Mensagens
  Existem várias mensagens que são publicadas e consumidas por vários componentes em Monasca através do MessageQ.

5.4 Eventos

Os fluxos de eventos podem ser definidos pela filtragem e agrupamento de eventos usando campos no evento. Os manipuladores podem ser associados a definições de fluxo que disparam quando ocorre o gatilho do fluxo.

Exemplo de eventos OpenStack são: criação de instância e exclusão de instância. A ação específica pode seguir esses eventos. Por exemplo, se uma instância for excluída, todos os alarmes relacionados a esta instância devem ser excluídos também.

5.4.1 API de eventos

- Eventos
- Transformadores
- Definição de fluxo

5.4.2 Motor de Transformação

Consome eventos de Kafka, transforma-os e publica em Kafka.

5.4.3 Motor de eventos

Consume eventos transformados da Kafka e usa o pipeline Winchester para processá-los.

5.4.4 Distiller

Não é necessária nenhuma alteração.

5.4.5 Winchester

Adiciona suporte para multi-tenancy, atualiza as pipelines e adiciona e exclui definições de pipeline em tempo de execução. Fornecer definições de pipeline em métodos, não arquivos de yaml. Winchester atualmente lê as informações de configuração do pipeline dos arquivos yaml no horário de inicialização. Manipulador de pipeline publica eventos de notificação de modo que o mecanismo de notificação possa consumi-los.

5.4.6 Threshold Engine

Atualização gera eventos de transição de estado de alarme mais gerais.

5.4.7 MySQL

Inicializa esquemas de Winchester, transformações Monasca e esquemas de encanamento.

5.5 Requisitos keystone

Monasca baseia-se na chave para a execução e existem requisitos sobre quais configurações de chave deve existir, os quais: O ponto final para o api deve ser registrado em keystone como o serviço 'monasca'; O api deve ter um token de administrador para usar na verificação dos tokens de keystone que recebe; Para cada projeto que use Monasca, dois usuários devem existir - um no papel 'monasca-agent' e será usado

pelo agente da monasca em máquinas e o outro não deve estar nessa função e pode ser usado para fazer login na UI, usando a CLI ou para consultas diretas contra a API.

5.6 Logging

5.6.1 Gestão de logs - lado do cliente

1. Monasca Log Agent - Logstash

   Monitora um ou mais arquivos de log, adiciona informações de meta (por exemplo, dimensões), autentica com o KeyStone e envia os logs (em um volume) para a API do Monasca Log.

2. Monasca Log Agent - Beaver

   Monitora um arquivo, adiciona informações de meta (por exemplo, dimensões), autentica com KeyStone e envia os logs (em um volume) para a API do Log Monasca.

5.6.2 Gerenciamento de Log - Lado do Servidor - Consumindo Logs

1. Monasca Log API Consome logs dos agentes, autoriza-os e publica-os para Kafka.
2. Monasca Log Transformer Consome logs de Kafka, transforma-os e os publica em Kafka.
3. Monasca Log Persister Consome logs da Kafka, prepara-os para o armazenamento a granel e os armazena na Elasticsearch.
4. Monasca Log Metrics Consome registros de Kafka, cria métricas para logs com severidade CRÍTICA, ERROR, ADVERTÊNCIA e publica-os para Kafka.
5. Monasca Log Storage Todos os logs são armazenados no Elasticsearch.

5.6.3 Gerenciamento de Log - Lado do Servidor - Visualizando Logs

- Servidor Monasca Kibana Autorização com KeyStone e visualização de logs (armazenados em elasticsearch). Tecnologia de base: Kibana

5.6.4 Gerenciamento de logs - Fluxo de dados

O diagrama visualiza a integração de logs no pipeline de processamento da Monasca. Há indicação de atalhos para primeiro passo.

5.7 Ambiente de desenvolvimento e tecnologias

O DevStack é o principal ambiente de desenvolvimento para o OpenStack: instala o Serviço Monasca, o Agente, o Painel de Monitoramento Horizon e Grafana O Monasca usa uma série de tecnologias de terceiros, aqui destacadas: Apache kafka, POSTgres e InfluxDB.

5.8 Sequência pós métrica

Esta seção descreve a seqüência de operações envolvida na publicação de uma métrica para a API Monasca.

1. Uma métrica é postada na API da Monasca.

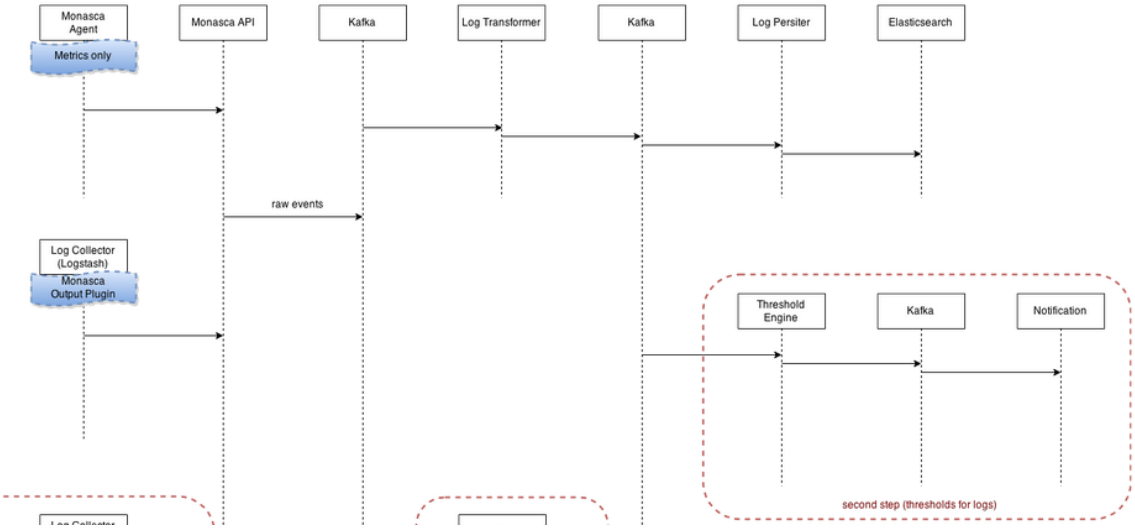

Figura 7: Fluxo de dados para monasca

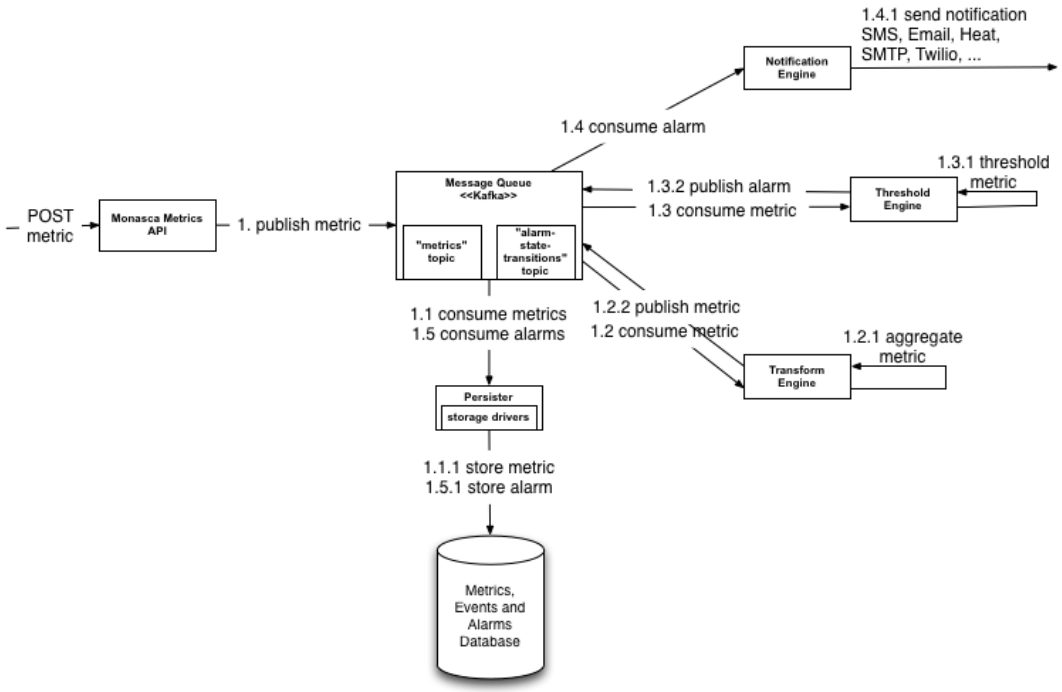

Figura 8: Operações para publicação de métrica para monasca

2. A API Monasca autentica e valida a solicitação e publica a métrica para o Message Queue.
3. O Persister consome a métrica da Fila de Mensagens e armazena na Metrics Store.
4. O Transform Engine consome as métricas da Message Queue, executa transformações e operações de agregação em métricas e publica métricas que cria de volta para Message Queue.
5. O Threshold Engine consome métricas da Message Queue e avalia alarmes. Se uma alteração de estado ocorrer em um alarme, um "evento de estado de alarme-estado"é publicado para o Message Queue.

6. O mecanismo de notificação consome eventos de transição de estado de alarme da fila de mensagens, avalia se eles possuem um método de notificação associado e envia a notificação apropriada, como email.

7. O Persister consome o "evento de transição de estado de alarme"da fila de mensagens e o armazena na loja de histórico de estado de alarme.

6 Considerações Finais

O referido documento traz o relato do monitoramento do uso dos recursos utilizados juntos a nuvem OpenStack para o trimestre maio-julho/2017. Foi descrito especificadamente o serviço Telemetry, seus segmentos, objetivos e frutos. Além disso, foi apresentado também o estudo especificado do Monasca, um "aplicativo"associado ao Telemetry para solução de problemas (como a lentidão).

Uma dificuldade encontrada foi a mudança brusca na documentação do OpenStack, atrasando consideravelmente a pesquisa. Como sugestões de atividades para serem realizadas no próximo trimestre podem ser citadas:

- Aprofundamento do estudo do Telemetry e seus serviços

- Instalação de ferramentas para monitoramento dos recursos, em sequência:
    1. Gnocchi (medições - banco de dados)
    2. MySQL (Alarmes)
    3. ElasticSearch (Eventos)