Pablo Brunetti dos Santos

CloudMetric: Um arcabouço para criação e monitoramento de métricas personalizadas em nuvens computacionais

Vitória, ES

27 de Setembro de 2019

Pablo Brunetti dos Santos

CloudMetric: Um arcabouço para criação e monitoramento de métricas personalizadas em nuvens computacionais

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Orientador: Prof. Dr. Rodolfo da Silva Villaça

Vitória, ES

27 de Setembro de 2019

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

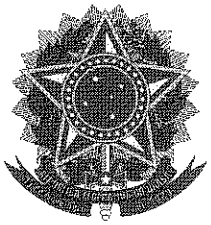
---

Santos, Pablo Brunetti dos, 1990-
S237c CloudMetric: Um arcabouço para criação e monitoramento de métricas personalizadas em nuvens computacionais / Pablo Brunetti dos Santos. - 2019.
77 f. : il.

Orientador: Rodolfo da Silva Villaça.
Dissertação (Mestrado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Computação em nuvem. I. Villaça, Rodolfo da Silva. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004



CLOUDMETRIC: UM ARCABOUÇO PARA CRIAÇÃO E MONITORAMENTO DE MÉTRICAS PERSONALIZADAS EM NUVENS COMPUTACIONAIS

Pablo Brunetti dos Santos

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 27 de setembro de 2019:

Prof. Dr. Rodolfo da Silva Villaca
Orientador(a)

Prof. Dr. Magnos Martinello
Membro Interno

Prof. Dr. Alexnan Bartholomeu Liberato
Membro Externo, Participação remota

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória-ES, 27 de setembro de 2019.

Agradecimentos

Agradeço aos meus pais por toda a dedicação que tiveram comigo durante ao mestrado.

Agradeço à CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pela concessão da bolsa.

Agradeço por ter recebido financiamento parcial do projeto Horizon 2020 (União Européia) sob no. 688941 (FUTEBOL), assim como do MCTI por meio da RNP e do CTIC para a execução deste mestrado.

Agradeço ao professor orientador e à equipe do NERDS (Núcleo de Estudos em Redes Definidas por Software) pela colaboração.

Resumo

As nuvens computacionais necessitam de medição e monitoramento constante para garantir o funcionamento adequado das diferentes aplicações ali hospedadas. Questões como: “Como monitorar?”, “O quê monitorar?” e “Como integrar métricas de desempenho das aplicações à nuvem?” são aspectos relevantes que precisam ser investigados. Contudo algumas plataformas de nuvens não oferecem um serviço de monitoramento de recursos que possa ser personalizado e meios de criar métricas especializadas para diferentes recursos e aplicações. Toda a criação e monitoramento de métricas é realizado em forma de *scripts* e de modo descentralizado, ou seja, não existe a possibilidade de integração entre múltiplas nuvens OpenStack, por exemplo. Portanto, esse trabalho propõe um arcabouço destinado a criação e monitoramento de métricas personalizadas em nuvens computacionais. Para validar o CloudMetric, foi construído um protótipo baseado na plataforma de computação em nuvem OpenStack, que permite o monitoramento de métricas criadas e gerenciadas pelo módulo Ceilometer, e a criação e monitoramento de métricas personalizadas realizada pelo próprio CloudMetric. O monitoramento é realizado através da consulta de métricas armazenadas no banco de dados temporal *Gnocchi*, visualizando as métricas no construtor de visualizador de *dashboards Grafana* e exportando em um arquivo no formato json. Através desse arcabouço foi demonstrado que o CloudMetric é capaz de realizar a criação de métricas de uma maneira padronizada, sem utilização da *APIs* de programação nativa do OpenStack, de maneira rápida, simples e robusta. São apresentadas demonstrações das funcionalidades do CloudMetric em 2 Casos de Teste: controle de trajetória de um robô em um espaço inteligente e monitoramento de métricas de um servidor *web*.

Palavras-chaves: Monitoramento, OpenStack, Nuvens Computacionais.

Abstract

The computational clouds need constant measurement and monitoring to ensure the proper functioning of the different applications hosted there. Questions such as “ How to monitor? ”, “ What to monitor? ” And “ How to integrate application performance metrics with the cloud? ” Are relevant aspects that need to be investigated. However some cloud platforms do not offer a customizable resource monitoring service and means of creating specialized metrics for different resources and applications. All metrics creation and monitoring is performed in a script and decentralized manner, ie there is no possibility of integration between multiple OpenStack clouds, for example. Therefore, this paper proposes a framework for the creation and monitoring of custom metrics in computational clouds. To validate CloudMetric, a prototype based on the OpenStack cloud computing platform was built, which allows monitoring of metrics created and managed by the Ceilometer module and the creation and monitoring of custom metrics by CloudMetric itself. Monitoring is performed by querying metrics stored in the Gnocchi temporal database, viewing the metrics in the Grafana dashboard viewer builder and exporting to a file in json format. Through this framework it was demonstrated that CloudMetric was able to perform metric creation in a standardized manner without using the OpenStack native programming APIs with fast, simple and robust monitoring. A demonstration of CloudMetric’s functionality is presented in 2 test cases: a robot’s path control in an intelligent space and web server metric monitoring. In addition, tutorial documentation allows the use of this framework in future computational cloud-based research projects.

Keywords: Monitoring, OpenStack, Computational Clouds.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
AWS	Amazon Web Services
AMQP	Advanced Message Queuing Protocol
CLI	Command-Line Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
JSON	JavaScript Object Notation
FUTEBOL	Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory
GCP	Google Cloud Platform
IaaS	Infrastructure as a Service
IP	Internet Protocol
JSON	JavaScript Object Notation
MVT	Modelo, Visão e Template
NFV	Network Functions Virtualization
PaaS	Platform as a Service
QoE	Quality of Experience
RAM	Random Access Memory
REST	Representational State Transfer
SaaS	Software as a Service
SDK	Software development kit
SASL	Simple Authentication and Security Layer
SLA	Service Level Agreement
SSH	Secure Shell

TLS	Transport Layer Security
VM	Virtual Machine
VPN	Virtual Private Network
WEB	World Wide Web

Sumário

1 Introdução

Ambientes de Computação em Nuvem têm sido amplamente disponibilizados por grandes, pequenos e médios provedores de serviço. Um dos ambientes mais populares é o IaaS (*Infrastructure as a Service*), onde recursos de computação são disponibilizados para que os usuários hospedem suas aplicações nessas infraestruturas compartilhadas. No âmbito acadêmico, vários ambientes experimentais têm sido implantados usando conceitos e recursos de nuvens computacionais. Dentre esses ambientes destaca-se o FUTEBOL (*Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory*) [1], Necos (*Novel Enablers for Cloud Slicing*) [2] e FIBRE (*Future Internet Brazilian Environment for Experimentation*) [3]

Da mesma forma que as infraestruturas de redes e de servidores tradicionais requerem constante medição e monitoramento de redes e serviços para garantir o correto dimensionamento dos recursos disponíveis, as nuvens computacionais também têm essa necessidade, com o agravante de que a diversidade de diferentes níveis de serviço que precisam ser atendidos é bem maior, o que torna a tarefa bastante desafiadora (FANIYI; BAHSOON, 2015).

Subsidiar os orquestradores de recurso nas nuvens computacionais com informações adequadas, de tal forma a garantir a correta alocação dos recursos para os usuários e aplicações e garantir um nível de Qualidade de Experiência (QoE) para os usuários da nuvem computacional continua sendo um tema de grande relevância, tanto no mercado, quanto na academia. “Como monitorar?”, “O quê monitorar?” e “Como integrar métricas de desempenho das aplicações em nuvem?” são perguntas relevantes nesse contexto e que precisam ser investigadas para serem respondidas adequadamente.

Uma das tecnologias de Computação em Nuvem muito utilizada é o OpenStack (FOUNDATION, 2019c). O OpenStack é uma plataforma projetada para ambientes de IaaS, possuindo projetos de código aberto permitindo a contribuição de toda a comunidade na revisão dos códigos ou criação de novas funcionalidades. O OpenStack possui diversos módulos como o Horizon (*Dashboard*), Keystone (serviço de identidade), Glance (Imagens), Neutron (configuração da rede), AODH (serviço de alarme), Tacker (orquestração de NFV (*network functions virtualization*)), Heat (orquestração de aplicações na nuvem), Ceilometer(monitoramento), entre outros (OPENSTACK, 2019e). Dentre esses módulos destaca-se o Ceilometer, usado para a coleta de dados de todos os componentes do OpenStack.

[1] http://www.ict-futebol.org.br/
[2] http://www.h2020-necos.eu/
[3] https://www.fibre.org.br/

O Ceilometer monitora todos os recursos disponíveis, tais como recursos de cada instância de máquinas virtuais (Utilização de CPU (*Central Process Unit*), uso de memória RAM (*Random Access Memory*), métricas relacionados ao disco rígido), recursos de Rede, além de recursos relativos aos seus próprios módulos, como por exemplo recursos de identidade no Keystone. Além disso fornece todas as medidas Utiliza-se o banco de dados temporal Gnocchi para armazenar as medições das métricas. O Gnocchi se integra a diversos módulos do OpenStack (AODH, Tacker, Heat) e é o banco de dados temporal recomendado (FOUNDATION, 2019b) pela Fundação OpenStack.

Para efetuar o monitoramento no OpenStack é necessário utilizar meios complexos e dependente da aprendizagem de comandos e APIs (*applications programming interface*).

O OpenStack não oferece nativamente uma interface gráfica em seu *Dashboard* (FOUNDATION, 2019d), para configurar ações de monitoramento de visualização dos recursos disponíveis. Devido a esse fato, para efetuar o monitoramento de qualquer métrica no OpenStack é necessário possuir um conhecimento da documentação e efetuar uma sequência de comandos no terminal. Outra dificuldade encontrada é na visualização dessas métricas. Para visualizar as métricas de forma nativa no OpenStack é preciso realizar algumas configurações para integrar o visualizador de métricas Grafana, ou criar sua própria aplicação de visualização para ler as medidas via interface Rest API do Gnocchi. O Grafana é uma plataforma *open source* que permite consultar, visualizar, alertar e interpretar métricas armazenadas no banco de dados do Gnocchi.

O OpenStack também não oferece nativamente uma interface gráfica para a configuração de monitoramento e visualização de métricas de recursos externos, como por exemplo medidas realizadas em nível de aplicações que estão instaladas nas máquinas virtuais dos usuários. Para criar métricas personalizadas e adicioná-las ao monitoramento do OpenStack é necessário ter conhecimento de toda a documentação *Rest API* do Gnocchi e criar *scripts* para essa finalidade, que na maioria das vezes não possuem um padrão, provocando retrabalho e gerando muitos erros na criação destes *scripts*.

Nesse contexto este trabalho apresenta o CloudMetric, um arcabouço destinado à criação e monitoramento de métricas personalizadas múltiplas nuvens OpenStack. Por meio da utilização do CloudMetric o monitoramento de diversas nuvens OpenStack pode ser concentrado em um único serviço, com espaços de monitoramento personalizados por usuário, interface gráfica amigável, responsiva e acessada por qualquer navegador *web (World Wide Web)* atual.

1.1 Objetivos

O objetivo geral desse trabalho é a criação de um arcabouço capaz de monitorar simultaneamente métricas nativamente existente em nuvens OpenStack e facilitar a criação

e o monitoramento de métricas personalizadas criadas pelos usuários.

São objetivos específicos deste trabalho:

- Propor uma arquitetura que habilite a criação e o monitoramento de métricas personalizadas por usuários devidamente cadastrados e autorizados pelo sistema;
- Elicitar os requisitos para a integração das múltiplas nuvens OpenStack com o arcabouço proposto;
- Desenvolver um padrão de entrada de dados para todas as métricas personalizadas criadas pelos usuários.
- Desenvolver uma interface *web* amigável, permitindo que os usuários do CloudMetric visualizem, criem e exportem métricas de múltiplas nuvens de maneira rápida, simples e robusta.

1.2 Estrutura do Texto

Este trabalho se encontra organizado da seguinte maneira:

- O Capítulo 2 apresenta trabalhos relacionados ao arcabouço CloudMetric e aponta as principais contribuições do CloudMetric perante o estado da arte do monitoramento de ambientes de nuvem;
- O Capítulo 3 apresenta uma revisão dos principais conceitos e ferramentas empregadas na criação e desenvolvimento do CloudMetric;
- O Capítulo 4 detalha o modelo de arquitetura conceitual para a construção do arcabouço;
- O Capítulo 5 traz de forma detalhada o levantamento de requisitos, diagrama de classes e o desenvolvimento do arcabouço;
- O Capítulo 6 apresenta os resultados da avaliação do CloudMetric e os Caso de Teste implementados.
- O Capítulo 7 traz a conclusão, resumindo as principais contribuições do CloudMetric e aponta sugestões de trabalhos futuros.

2 Trabalhos Relacionados

Este capítulo apresenta os trabalhos relacionados que contextualizam o desenvolvimento deste trabalho, destacando as principais contribuições desta dissertação. Serão apresentados trabalhos que abordam arquitetura com múltiplas nuvens OpenStack, criação de métricas personalizadas a níveis de usuários e integração com módulos da nuvem. Além disso serão mostrados trabalhos que realizaram a validação através de protótipos.

2.1 Arquitetura de um monitor de desempenho de rede para serviços de aplicação em múltiplas nuvens

No trabalho (Young-min Kim et al., 2013) foi criado uma arquitetura de um monitor de desempenho de rede para serviços de aplicação em múltiplas nuvens. Esta arquitetura inclui integração com agentes externos, agentes internos, armazenamento central das métricas em uma base de dados (DMBS) e possibilidade de consumo das métricas das múltiplas nuvens em tempo real através do padrão de comunicação *publish/subscribe*. O foco do trabalho foi na integração de agentes externos fornecedores de métricas de medição e desempenho *DipZoom* (Rabinovich et al., 2006) e *PlanetLab* (CHUN et al., 2003) e na utilização do agente interno *CloudCmp* (LI et al., 2010) para o fornecimento de métricas internas (CPU, Memória) nas múltiplas nuvens monitoradas.

Embora este trabalho tenha características semelhantes ao CloudMetric, a arquitetura proposta possibilita somente a integração aos agentes externos *DipZoom* (Rabinovich et al., 2006) e *PlanetLab* e impossibilita a integração das métricas internas com outros módulos agregados às nuvens computacionais monitoradas devido ao fato de utilizar o *CloudCmp* como agente interno das múltiplas nuvens monitoradas.

2.2 Projeto de um agente baseado em um arcabouço de monitoramento para nuvens federadas

O trabalho (Aversa; Tasquier, 2016) propõe uma arquitetura multicamadas, em que cada camada visa monitorar diferentes aspectos da infraestrutura em múltiplas nuvens, a partir da detecção de condições críticas e compondo diferentes níveis de monitoramento para verificar o Contrato de Nível de Serviço (SLA) federado. Essa arquitetura é composta por 4 camadas: *Broker*, Implantação de aplicações, serviço de monitoramento e violação de SLA.

A camada mais baixa no serviço de monitoramento é representada por *Probes*, que representam agentes instalados pelo usuário. A sua função é detectar violações das métricas de desempenho do recurso realizando a comparação dos recursos obtidos das máquinas virtuais com parâmetros do SLA dos recursos monitorados, enviando um alerta sempre que algum parâmetro estiver fora da faixa contratada. O monitoramento utilizado nesse serviço de monitoramento é baseado em agentes de infraestruturas de nuvem heterogêneas (Aversa; Tasquier; Venticinque, 2013). Esse modelo de monitoramento permite a inclusão de novas métricas através de desenvolvimento de módulos de criação de métricas personalizadas dos agentes.

Este trabalho se difere do CloudMetric pois não utiliza módulos de monitoramento nativos das nuvens monitoradas, impossibilitando que as métricas personalizadas se integrem com os serviços nativos da nuvem.

2.3 *CloudProcMon*: uma estrutura de monitoramento de nuvem não intrusiva

O monitoramento na nuvem coleta métricas de monitoramento das infraestruturas físicas e virtuais da nuvem. A coleta dessas métricas podem ser realizadas de forma intrusiva através da inserção de agentes de monitoramento nas máquinas virtuais da nuvem ou não intrusiva através da coleta dos dados no sistema operacional hospedeiro (nó de Computação em Nuvem) ou do componente da nuvem (controlador da nuvem). Em *CloudProcMon* (Syed et al., 2018) é criado uma estrutura de monitoramento que coleta dados de monitoramento do sistema operacional hospedeiro de maneira não invasiva e realiza a vinculação desses dados ao controlador de nuvem para uso no monitoramento. Os dados de monitoramento foram coletados vinculadas ao painel de monitoramento no nó do controlador de nuvem.

Este trabalho se assemelha ao CloudMetric por apresentar um protótipo criado para o OpenStack. Entretanto esse protótipo não funciona para múltiplas nuvens e criação de métricas personalizadas.

2.4 Uma estrutura genérica e extensível para monitorar o consumo de energia de nuvens OpenStack

Em (Rossigneux et al., 2014) foi apresentada a arquitetura de uma estrutura genérica e flexível, denominada *KiloWattAPI* (KWAPI), que faz interface com o OpenStack para fornecer informações sobre o consumo de energia. A estrutura suporta vários dispositivos medidores de energia, vários formatos de medição e minimiza a sobrecarga da

comunicação. Está estrutura é integrada ao Ceilometer. O trabalho possui a limitação de não se integrar em múltiplas nuvens e fornecer suporte a uma quantidade limitada de métricas personalizadas.

Na Tabela 1 é apresentada uma comparação entre as principais características do trabalhos apresentados. Através dessa tabela é visto que o CloudMetric se diferencia dos 4 trabalhos apresentados, pois agrega as características de monitorar múltiplas nuvens OpenStack, realizar a criação de métricas personalizadas em qualquer aplicação, se integrar com módulos da nuvem OpenStack e ter um protótipo construído validando todo o trabalho realizado.

Tabela 1 – Comparativo entre os trabalhos

Trabalho	**Múltiplas Nuvens OpenStack**	**Métricas Personalizadas**	**Integrado com módulos da nuvem**	**Protótipo**
CloudMetric	Sim	Sim	Sim	Sim
2.1	Sim	Limitado a métricas fornecidas por 2 agentes externos	Não	Limitado a visualização dos resultados em consulta no banco de dados
2.2	Sim	Sim	Não	Não
2.3	Não	Não	Não	Sim
2.4	Não	Limitado	Sim	Somente gráficos

3 Conceitos Fundamentais

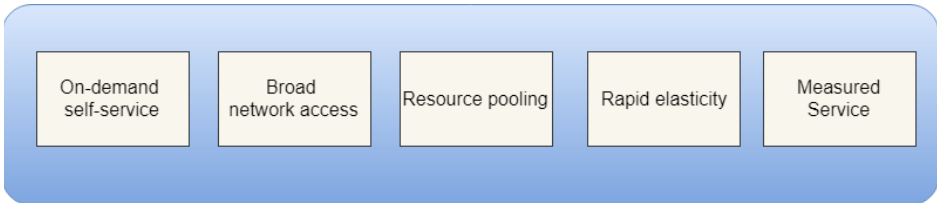
Este capítulo tem como objetivo apresentar os conceitos fundamentais utilizado no desenvolvimento do CloudMetric. Nele serão vistos conceitos relacionados à Computação em Nuvem, à plataforma IaaS OpenStack e ao protocolo AMQP.

3.1 Computação em Nuvem

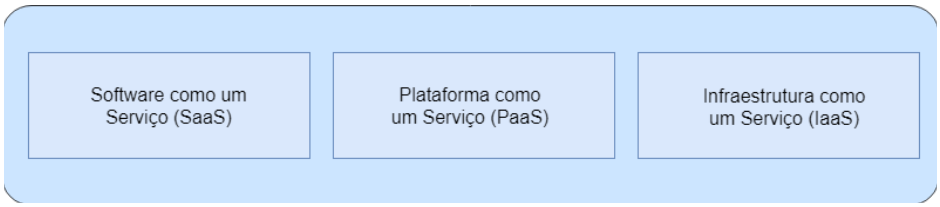
Segundo (BADGER et al., 2012), Computação em Nuvem pode ser definido como um modelo para habilitar acesso à rede de forma ubíqua, conveniente e sob demanda para recursos de computação configurável (redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com o mínimo gerenciamento de esforço ou provedor de serviços. Sendo assim, para que um serviço possa ser caracterizado como Computação em Nuvem, é essencial que ele seja estruturado tal como apresentado na Figura 1.

As características essenciais de um serviço de nuvem são:

- *On-demand self-service*: Um usuário pode provisionar unilateralmente recursos de computação sem exigir interação humana com cada provedor de serviços;
- *Broad network access*: Os recursos estão disponíveis na rede e o acesso é realizado por meio de mecanismos que promovam o uso de plataformas heterogêneas (celulares, *tablets*, laptop e computadores pessoais);
- *Resource pooling*: Os recursos de computação do provedor são agrupados para atender a vários usuários, usando um modelo *multi-tenant* com diferentes recursos físicos e virtuais, alocando os recursos de acordo com a demanda do usuário;
- *Rapid elasticity*: Os recursos podem ser provisionados e liberados de forma elástica, na maioria dos casos de forma automática, mensurando os recursos de acordo com a demanda dos usuários;
- *Measured Service*: Os sistemas em nuvem controlam e otimizam automaticamente o uso de recursos, aproveitando um serviço de medição e algum nível de abstração. O uso desses recursos podem ser monitorados, controlados e reportados, fornecendo transparência para o fornecedor e o usuário que utiliza o serviço.



(a) Características Essenciais



(b) Modelos de Serviço

(c) Modelos de Implantação

Figura 1 – Modelo de Nuvem proposto por (BADGER et al., 2012)

Além disso existem três modelos de serviço de Computação em Nuvem (BADGER et al., 2012):

- `Software como um Serviço (SaaS)`: O Software como um Serviço fornece um produto completo executado e gerenciado pelo provedor de serviço, sem o gerenciamento do usuário na infraestrutura da nuvem, incluindo redes, servidores e sistemas operacionais;
- `Plataforma como um Serviço (PaaS)`: Com as Plataformas como serviço, as organizações não precisam gerenciar a infraestrutura subjacente (geralmente hardware e sistemas operacionais), e podem focar na implantação e no gerenciamento das aplicações, fazendo o uso de linguagens de programação, bibliotecas, serviços e configurações suportados pelo fornecedor do serviço;
- `Infraestrutura como um Serviço (IaaS)`: A Infraestrutura como Serviço fornece o mais alto nível de flexibilidade e de controle de gerenciamento dos seus recursos para o consumidor, como por exemplo provisionar recursos de computação (processamento, armazenamento, rede).

De acordo com (BADGER et al., 2012) a infraestrutura de nuvem pode ser implantada em quatro modelos:

- `Nuvem Privada`: Provisionada para uso exclusivo de uma única organização. Pode ser gerenciada e mantida pela própria organização ou por terceiros;
- `Nuvem Comunitária`: Provisionada para uso específico de uma comunidade que possui interesses em comum. Pode ser gerenciada e mantida pela própria organização ou por terceiros;
- `Nuvem Pública`: A infraestrutura de nuvem é provisionada para uso aberto ao público geral. Pode ser gerenciada, mantida e operada por uma organização comercial, acadêmica ou governamental;
- `Nuvem Híbrida`: A infraestrutura de nuvem é uma composição de duas ou mais infraestruturas de nuvem distintas(privada, comunitária ou pública) quer permanecem como entidades exclusivas, mas unidas por tecnologia padronizada que permite a portabilidade de dados e aplicações.

3.2 OpenStack

O OpenStack é um sistema operacional de nuvem que controla grandes conjuntos de recursos de computação, armazenamento, e recursos de rede dentro de um *datacenter*, todos gerenciados e provisionados através de APIs (*Applications Programming Interface*) com mecanismos comuns de autenticação, como representado na Figura 2. Além da funcionalidade padrão de uma Infraestrutura como Serviço, possui componentes adicionais que fornecem orquestração, gerenciamento de falhas e serviços, dentre outros, que garantem a alta disponibilidade das aplicações dos usuários (FOUNDATION, 2019c).

A sua plataforma possui código aberto, a qual pessoas ou empresas podem colaborar com o projeto, acessar o código fonte e propor melhorias na plataforma. Devido a essas características grandes empresas utilizam essa plataforma, podendo citar como exemplo o WalMart, T-Mobile, Santander, Nike (FOUNDATION, 2017).

OpenStack é composto por diversos projetos de código aberto (OPENSTACK, 2019e). Existem 6 (seis) serviços básicos e estáveis que abrangem computação, rede, armazenamento, identidade e imagens, além de diversos projetos opcionais em diversos estágios de desenvolvimento. Esses serviços básicos constituem a infraestrutura que permite aos projetos terem acesso a painéis, orquestração, provisionamento de *hardware*, sistema de mensageria, *containers* e governança. Os seis módulos básicos são:

- `Nova`: Responsável pelo gerenciamento total dos recursos computacionais do OpenStack, incluindo programações, criações e exclusões;
- `Neutron`: Responsável por gerenciar os recursos de rede e prover conectividade a outros serviços do OpenStack;

- `Keystone`: Responsável por autenticar, autorizar e fornecer o catálogo de *endpoints* para todos os serviços do OpenStack;
- `Glance`: Serviço responsável pela descoberta, registro e recuperação de imagens de máquinas virtuais;
- `Cinder`: Responsáve por oferecer armazenamento de blocos persistentes acessível *(block storage)*;
- `Swift`: Responsável pelo armazenamento de objetos altamente tolerante a falhas. Armazena e recupera objetos de dados não estruturados.

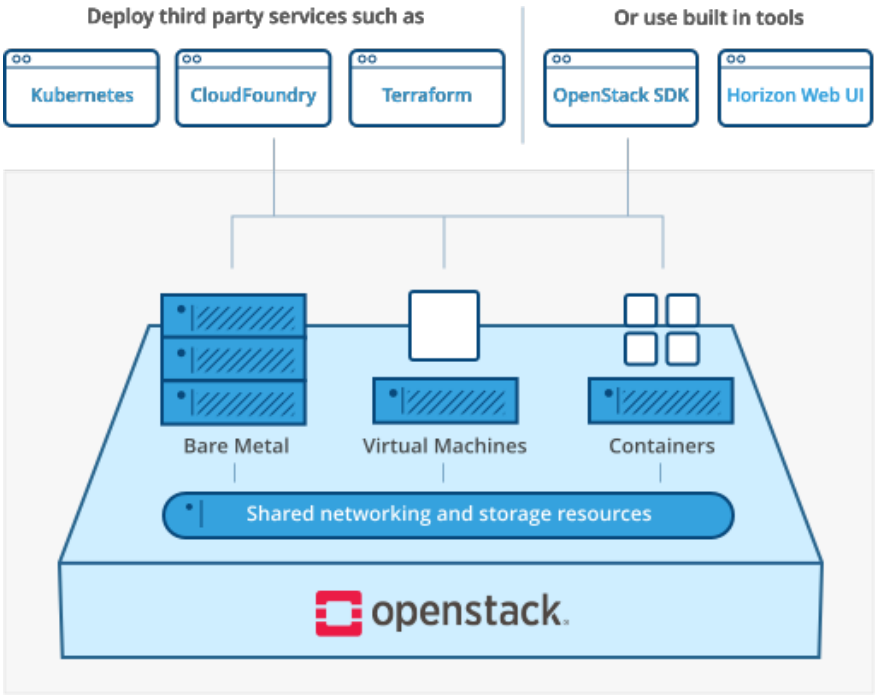


Figura 2 – Estrutura do OpenStack. Fonte: (OPENSTACK, 2019d)

Além dos módulos básicos o OpenStack possui diversos outros projetos que proveem serviços de telemetria, interface com o usuário, e outros. Na Figura 3 é apresentado todos os serviços básicos, os serviços de telemetria, a comunicação e interação entre os módulos. Podemos destacar os módulos Horizon, que oferece uma *Dashboard* de comunicação acessível via *Web* que possibilita a administração e provisionamento de recursos de Computação em Nuvem e o Ceilometer, módulo responsável por prover as medições de recursos da nuvem.

Considerando os módulos básicos e os opcionais que usualmente são implantados, um cenário de implantação comum é o uso de um nó dedicado aos serviços de gestão da nuvem e um ou mais nós dedicados a prover recursos de computação. Esse esquema de implantação é exemplificado na Figura 4.

Para a compreensão das funcionalidades fornecidas pelo OpenStack, é necessário conhecer alguns conceitos básicos. Projetos são unidades organizacionais na nuvem às

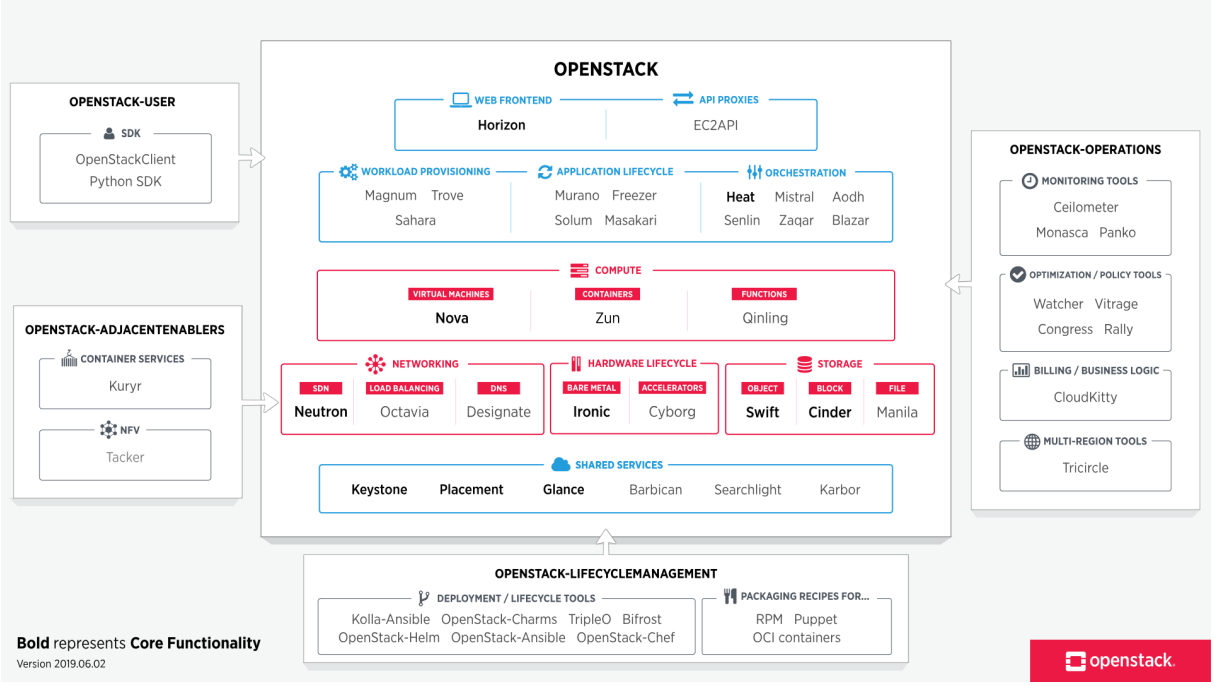


Figura 3 – Mapa de serviços do OpenStack. Fonte: (OPENSTACK, 2019d)

quais os usuários são atribuídos. Cada usuário membro do projeto tem um papel, ou seja, um conjunto de direitos e privilégios que definem quais ações ele pode executar. Os usuários podem ser membros de um ou mais projetos, possuindo um papel em cada um deles (OPENSTACK, 2019b).

O OpenStack, se comunica internamente entre seus módulos a partir de APIs (OPENSTACK, 2019c). Para cada módulo, temos uma referência de API. Essas APIs são utilizadas na interações entre os módulos, tanto de forma visual por meio do módulo Horizon, como por *scripts* criados por administradores. Nem todos os módulos apresentam uma interface visual no Horizon, como por exemplo o módulo Ceilometer, que precisa de *scripts* para executar qualquer funcionalidade.

O OpenStack mantém usuários para suas APIs, todos desenvolvidos na linguagem Python. Como exemplo podemos citar o OpenStackClient, que é a interface CLI (*comand-line interface*) para todos os serviços do OpenStack e o OpenstackSdk, o SDK (*software development kit*) oficial do OpenStack.

Para utilizar o OpenStackClient ou o SDK é necessário fornecer previamente os dados de identificação do usuário e do projeto. Esses dados são fornecidos através de um arquivo *OpenStack rc file*, ou openrc.sh. Esse *script* pode ser baixado diretamente da *Dashboard* do usuário no projeto a qual faz parte. No código 3.1, temos um exemplo básico de um arquivo openrc.sh, pertencente ao projeto admin e os parâmetros que precisam ser preenchidos para ser realizada a autenticação do usuário.

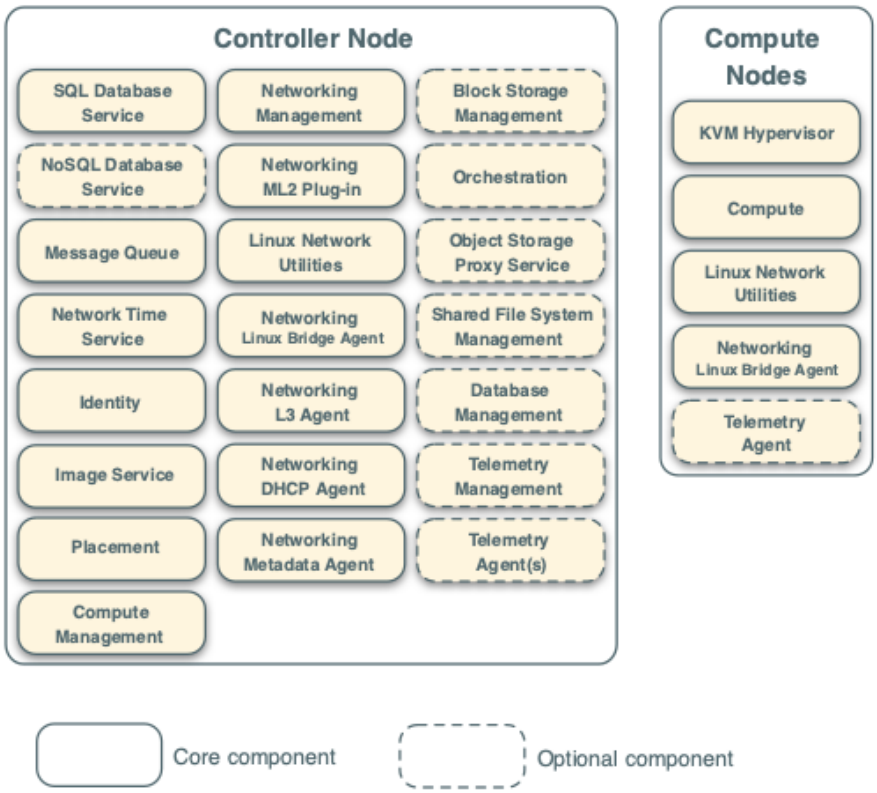


Figura 4 – Modelo de Implantação do OpenStack. Fonte: (OPENSTACK, 2019f)

Listagem 3.1 – Modelo de arquivo rc file. Fonte:(OPENSTACK, 2019a)

```
1 export OS_PROJECT_DOMAIN_NAME=default
2 export OS_USER_DOMAIN_NAME=default
3 export OS_PROJECT_NAME=admin
4 export OS_USERNAME=admin
5 export OS_PASSWORD=ADMIN_PASS
6 export OS_AUTH_URL=http://controller:35357/v3
7 export OS_IDENTITY_API_VERSION=3
8 export OS_IMAGE_API_VERSION=2
```

Após carregar essas variáveis de ambiente é possível executar os comandos da API/SDK do OpenStack.

3.3 Ceilometer

O Ceilometer é um serviço de telemetria a qual coleta dados de vários serviços do OpenStack (FOUNDATION, 2019a).

O Ceilometer oferece cinco serviços centrais, conforme apresentado na Figura 5. Cada um dos serviços do Ceilometer é projetado para escalar horizontalmente. Processos separados dedicados à execução de fluxo de tarefas chamados de `Workers` e nós podem ser adicionados dependendo da carga esperada.

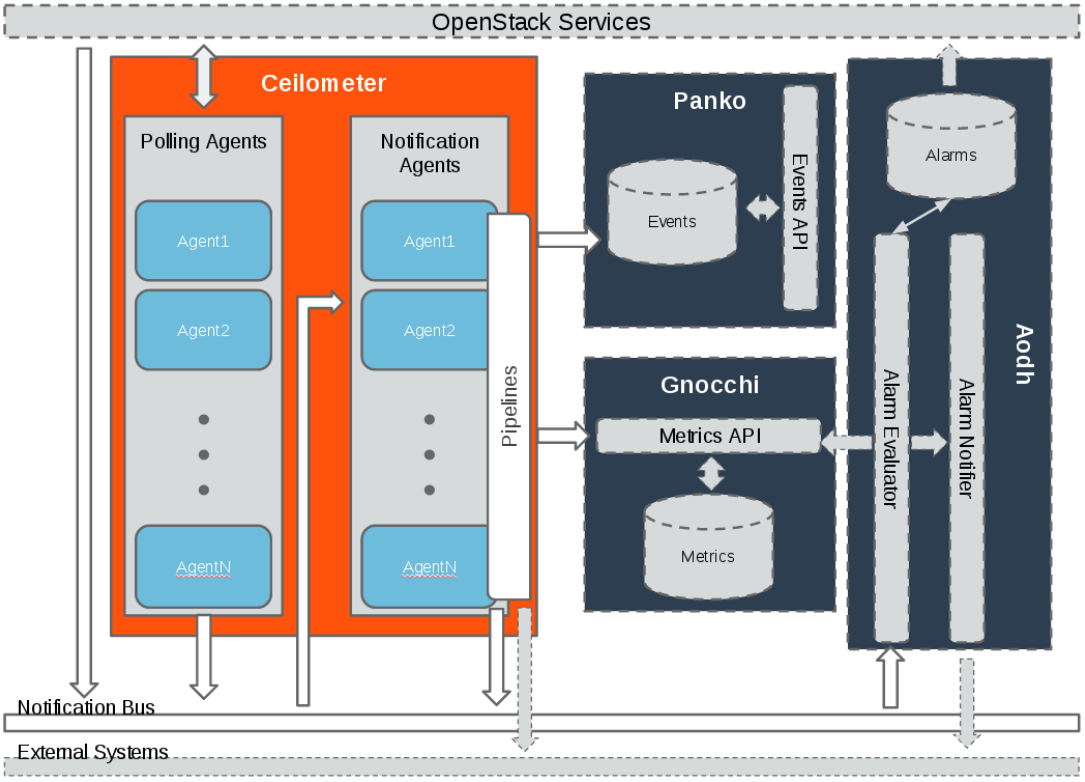


Figura 5 – Arquitetura do Ceilometer. Fonte:(FOUNDATION, 2019a)

- `Agente de Polling`: *Daemon* projetada para efetuar o *polling* de serviços (*compute*, volume,*network*,*image*,*object storage)* do OpenStack;
- `Agente de Notificação`: *Daemon* projetado para ouvir notificações no software de enfileiramento de mensagens (*broker*) do OpenStack, convertê-las em eventos e amostras, e aplicar ações.

Todos os dados são normalizados, coletados pelo Ceilometer e enviados para o banco de dados temporal *Gnocchi*.O *Gnocchi* captura os dados de medição em um formato de série temporal, otimizando o armazenamento e as consultas.

O Ceilometer possui em sua arquitetura o serviço de alarme chamado Aodh, que possui a função de enviar alertas quando as regras definidas pelo usuário são violadas e o Panko, serviço responsável pelo armazenamento de eventos projetado para capturar dados orientados a documentos, como logs e ações de eventos do sistema.

3.4 Gnocchi

Gnocchi é um banco de dados temporal *multi-tenant* de métricas e recursos (GNOCCHI, 2019). É projetado para armazenar métricas em grande escala e possui propriedades de alta disponibilidade, garantindo que a telemetria esteja sempre ativa e recuperada

de forma rápida. Consiste de diversos serviços: um *HTTP Rest API*, um *daemon* opcional compatível com *statsD* e um *daemon* de processamento chamado metricd, conforme apresentado na Figura 6.

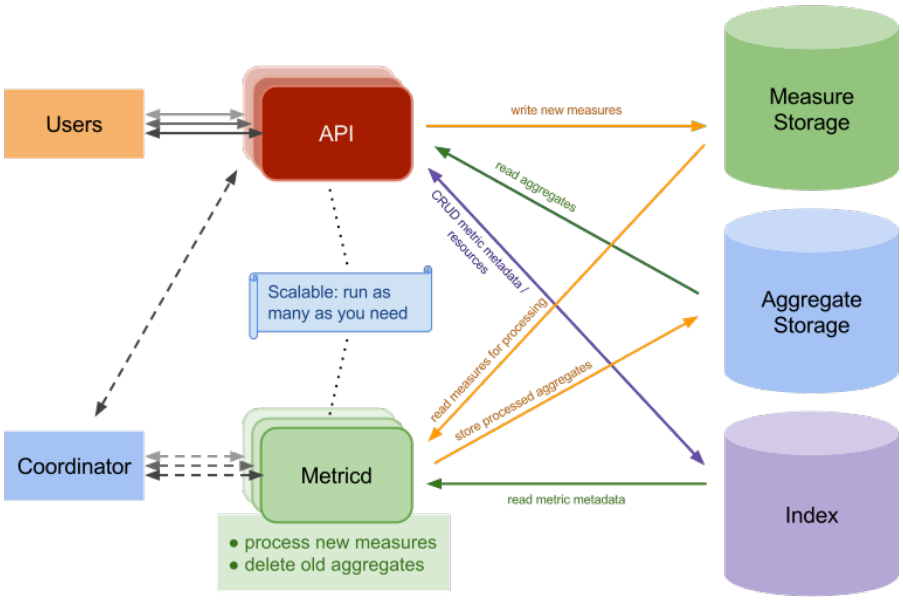


Figura 6 – Arquitetura de serviços do Gnocchi. Fonte:(GNOCCHI, 2019)

Todos os serviços são sem estado e escalados horizontalmente, ou seja, se a carga de trabalho for incrementada, novos serviços serão criados para processar os fluxos de novas requisições. O Gnocchi é composto de três componentes: *incoming measure storage*, *aggregated metric storage* e *indexer driver*.

O *incoming measure storage* e o *aggregated metric storage* podem utilizar diferentes sistemas de armazenamento (File, Ceph, OpenStack Swiftm Amazon S3 e Redis) enquanto que o *indexer driver* suporta os banco de dados PostgreSQL e MySQL para indexar os recursos e métricas manipulados pelo Gnocchi. O indexador é responsável por armazenar o índice de todos os recursos, arquivos de política e métricas, juntamente com suas definições, tipos e propriedades. O indexador também é responsável por vincular recursos a métricas e relacionar diferentes recursos.

Para criar uma métrica no Gnocchi é necessário agrupa-la em uma determinada lista de recursos denominada "Tipo de Recurso". Dentro do tipo de recurso, temos os recursos que são definidos como uma entidade, que representa um recurso específico na infraestrutura a qual se associa a métrica. Por fim a métrica é uma entidade que armazena agregados (tupla de dados gerada a partir de várias medidas, de acordo com a definição do arquivo de política, composta de *timestamp* e valor) e podem relacionar um recurso a uma métrica. A forma como uma métrica armazena esses agregados é definido pelo arquivo de política.

No arquivo de política é determinado por quanto tempo as agregações serão mantidas em uma métrica e como serão agregados. As agregações são calculadas durante um período de tempo chamado de granularidade e mantidas por um período de tempo definido no arquivo de política. Por padrão existem três tipos de arquivos de política:

- Baixo: armazena as métricas com granularidade de 5 minutos em 30 dias;
- Médio: armazena as métricas com granularidade de 1 minuto em 7 dias e 1 hora nos últimos 365 dias;
- Alto: armazena as métricas com granularidade de 1 segundo na última hora, 1 minuto em 7 dias e 1 hora nos últimos 365 dias.

Todos os arquivos de política armazena média, mínimo, desvio padrão, soma, máximo e quantidade.

O acesso ao Gnocchi pode ser realizado através de um cliente python chamado gnocchiclient e por um *Rest API* do Gnocchi.

3.5 AMQP

De acordo com (DIZDAREVI et al., 2019) AMQP (*Advanced Message Queuing Protocol*) é um protocolo aberto que segue o paradigma *publish-subscribe*, projetado para permitir a interoperabilidade entre uma ampla gama de aplicações e sistemas. Esse recurso de interoperabilidade do AMQP é significativo, pois permite que diferentes plataformas, implementadas em diferentes linguagens de programação troquem mensagens.

O AMQP foi implementado em duas versões muito diferentes, o AMQP 0.9.1 e AMQP 1.0, cada uma com um paradigma de mensagens completamente diferente. O AMQP 0.9.1 implementa o paradigma de *publish-subscribe* que possuem os componentes do *broker*: *exchange* e as filas de mensagens.

As *exchanges* representam uma parte do *broker* que é usado para direcionar as mensagens recebidas dos *publishers*. A publicação de mensagens em uma entidade do *exchange* é a primeira etapa do processo e, depois disso, as mensagens são roteadas para uma ou mais filas. Uma mensagem permanecerá na fila até ser recebida por um assinante. Esse processo de roteamento, que realmente liga *exchanges* e filas, depende das chamadas *bindings*, que são regras e condições predefinidas para a distribuição de mensagens.

A versão mais recente do protocolo AMQP, AMQP 1.0, por outro lado, não está vinculada a nenhum mecanismo de mensagens específico. Enquanto as versões mais antigas do protocolo usavam especificamente a abordagem de *publish-subscribe* com arquitetura que consiste de *exchanges* e filas de mensagens, as novas implementações do AMQP seguem

um paradigma *peer-to-peer* e podem ser usadas sem um *broker*. O AMQP usa o TCP para transporte confiável e, além disso, fornece três níveis diferentes de QoS. Por fim, o protocolo AMQP fornece mecanismos de segurança complementares, para proteção de dados usando o protocolo TLS (*Transport Layer Security*) para criptografia e para autenticação usando SASL (*Simple Authentication and Security Layer*).

3.5.1 RabbitMQ

O RabbitMQ é um software de enfileiramento de mensagens chamado de *message broker* ou gerenciador de filas. Ele é um software em que filas podem ser definidas e aplicativos podem se conectar a essas filas e transferir mensagens para elas. As filas de mensagens permitem a comunicação assíncrona, o que significa que outros aplicativos que estão produzindo e consumindo mensagens interagem com a fila, em vez de se comunicarem diretamente uns com os outros.

Uma mensagem pode incluir qualquer informação. Pode, por exemplo, conter informações sobre um processo ou trabalho que deve iniciar em outro aplicativo ou pode ser uma simples mensagem de texto (JOHANSSON, 2018).

O *message broker* armazena as mensagens até que um aplicativo de recebimento se conecte e retire uma mensagem da fila. O aplicativo de recebimento processa adequadamente a mensagem. Um produtor pode adicionar mensagens a uma fila sem precisar esperar que elas sejam processadas.

A Figura 7 ilustra o fluxo de mensagens no RabbitMQ em forma de etapas. Na etapa 1 o produtor publica uma mensagem para uma *exchange*. O *exchange* recebe a mensagem na etapa 2 e efetua o roteamento da mensagem. Além disso examina diferente atributos e chaves da mensagem. Na figura é visto duas *bindings* na etapa 3 em duas filas diferente na *exchange*. A *exchange* encaminha a mensagem para a fila correta, dependendo dos seus atributos. As mensagens permanecem nas filas até que sejam consumidas por um consumidor, como mostrado na etapa 4 e 5 da figura.

Pode-se ter os seguintes tipos de *exchanges* no RabbitMQ:

- `Direto`: A *exchange* entrega mensagens na fila com base em uma chave de roteamento. Na *exchange* direta, a mensagem é roteada para a fila, cuja chave de ligação corresponde exatamente a chave de roteamento da mensagem;
- `Tópico`: No *exchange* em forma de tópico é realizado uma correspondência curinga entre a chave de roteamento e o padrão de roteamento especificado na ligação.;
- `Fanout`: Nesse tipo de *exchange*, as mensagens são roteadas para todas as filas vinculadas à *exchange*;

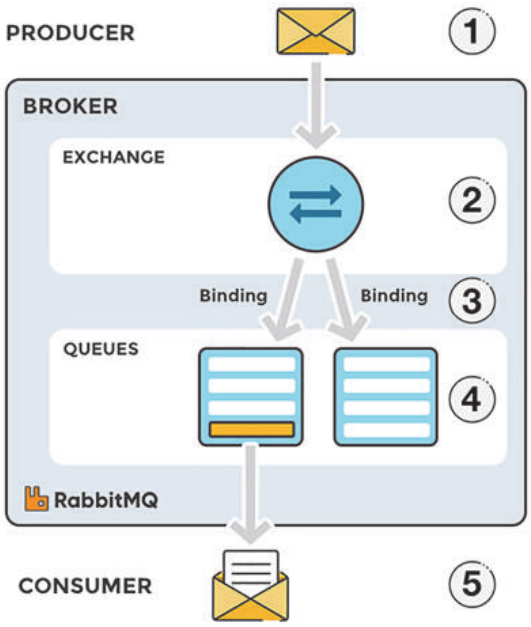


Figura 7 – *Workflow* do RabbitMQ. Fonte:(JOHANSSON, 2018)

.

- `Headers`: A *exchange* utiliza os atributos no cabeçalho de mensagem para realizar o seu roteamento.

Na Figura 8 é mostrado 3 (três) diferentes tipos de *exchanges*: Direto, Tópico e Fanout. Na *exchange* direta as mensagens foram publicadas para a fila que possui a chave de roteamento *PDF process*, enquanto que na *exchange* em forma de tópico as filas assinadas possuem o padrão de roteamento que iniciam com *eu.de.** e contenham no início da palavra *us.**. O caractere * pode ser substituído por exatamente uma palavra e o caractere # pode ser substituído por zero ou mais palavras.

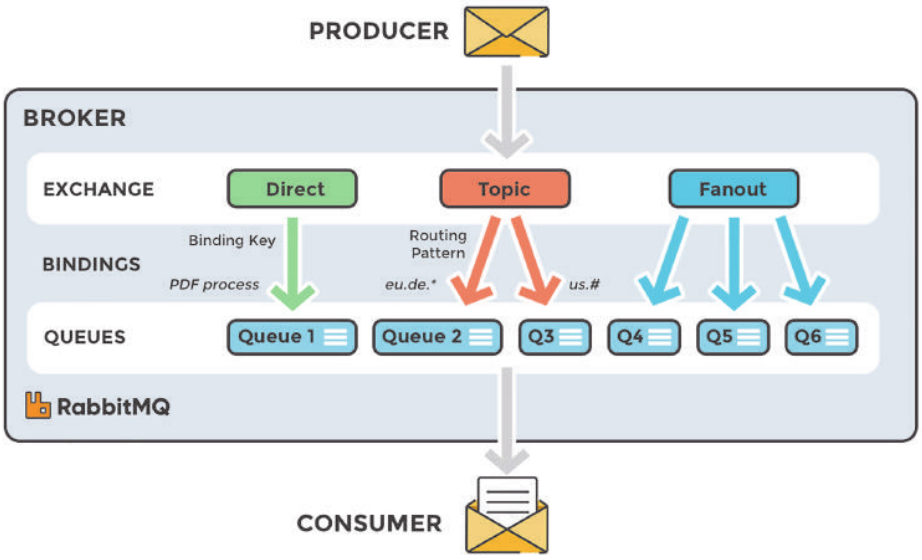


Figura 8 – Três diferentes tipos de *exchange* do RabbitMQ. Fonte: (JOHANSSON, 2018).

3.6 Grafana

O Grafana é uma conjunto de *softwares open source* que permite consultar, visualizar, alertar e interpretar métricas armazenadas em qualquer banco de dados a qual possui integração. É utilizado por diversas empresas de grande porte como o *Ebay*, *Paypal*, *StackOverflow*, *RedHat* e *Uber* (GRAFANALABS, 2019).

Possui os seguintes tipos de *plugins*:

- `Data Source`: *Backend* de armazenamento, a qual os dados temporais são capturados. Como exemplo temos o *Gnocchi*, *Graphite* e *CloudWatch*, como *Data Source* suportados pelo Grafana;
- `Painel`: Bloco básico de visualização dentro do Grafana. Cada painel fornece um editor de consulta (*Query Editor*) que permite a extrair a visualização dos dados temporais utilizando os parâmetros configurados no editor de consultas;
- `Editor de Consulta`: Editor capaz de consultar os recursos do *Data Source*. Através do Editor de Consulta é possivel construir uma ou mais consultas no banco de dados temporal;
- `Dashboard`: Os *Dashboards* podem ser considerados como um conjunto de um ou mais painéis que podem ser compartilhados, importados ou exportados.

O Grafana também fornece uma *Rest API* para a criação de *Dashboards* de forma automatizada. Na Figura 9 temos o exemplo de uma *Dashboard*, que exibe o uso de memória RAM em uma máquina virtual no OpenStack.

Figura 9 – Exemplo de uma *Dashboard* no Grafana.

4 Projeto do CloudMetric

Neste capítulo é descrito o modelo de arquitetura conceitual para a construção de um arcabouço para monitoramento e criação de métricas personalizadas em múltiplas nuvens computacionais. Esta proposta consiste de uma ferramenta que após configurada corretamente, irá proporcionar aos usuários as seguintes funcionalidades:

- Monitoramento de múltiplas Nuvens em um arcabouço único, *multi-tenant*, hospedada em um servidor que possui conexão *vpn(Virtual Private Network)*, *internet* ou *intranet*;
- Visualização e exportação de métricas criadas em cada nuvem, como por exemplo utilização da CPU e memória RAM nas máquinas virtuais, dentre outras;
- Criação de métricas de aplicações personalizadas, criadas ou gerenciadas pelo próprio usuário.

Para utilizar o CloudMetric é necessário possuir um conhecimento prévio da utilização da *dashboard* Horizon do OpenStack e o serviço de telemetria ceilometer.

Na Figura 10 é ilustrado os pré-requisitos de uso do CloudMetric. O usuário irá acessar o CloudMetric por meio de um navegador *Web* em um *notebook*, smartphone ou computador pessoal e realizar todas as interações com as múltiplas nuvens OpenStack a que possui acesso. Para que o usuário consiga efetuar o *login* no CloudMetric e ter acesso as todos as funcionalidades descritas anteriormente é necessário que o servidor ao qual o CloudMetric esteja hospedado tenha uma comunicação via intranet (rede local), *vpn* (rede privada virtual) ou *Internet* com essas múltiplas nuvens OpenStack e credenciais de acesso.

Para visualizar melhor os componentes do CloudMetric, na Figura 11 é apresentado a visualização em camadas da arquitetura. A camada de Clientes é formado pelos usuários que possuem acesso ao arcabouço do CloudMetric e efetuam trocas de requisições com o servidor. Os componentes servidor, banco de dados relacional e agentes (CloudMetricAgentInput e CloudMetricAgentReadBroker), estão agrupados em uma camada única e executam tarefas relativas as requisições enviadas pelos usuários para as múltiplas nuvens OpenStack. Dentro de cada nuvem computacional existe um banco de dados temporal, uma máquina virtual com um *broker* e um serviço de armazenamento de arquivos compartilhado.

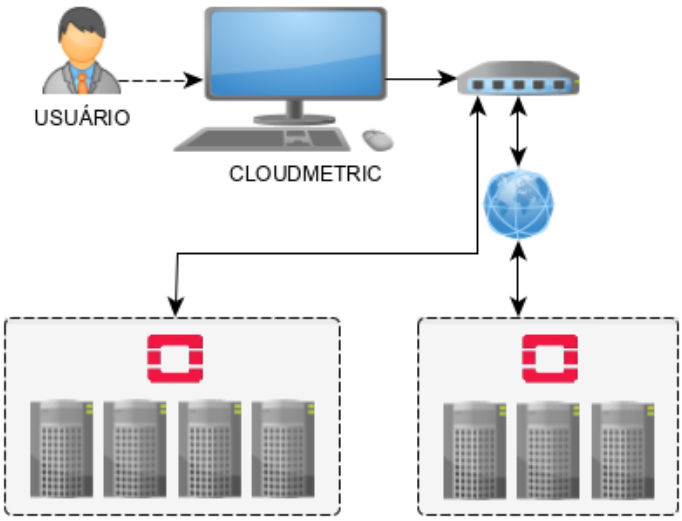


Figura 10 – Pré-requisitos de uso do CloudMetric.

O papel de cada componente está descrito a seguir:

- `Interface`: Responsável pela interação de forma gráfica de todas as funcionalidades dos usuários;
- `Servidor`: Responsável por processar solicitações requisitadas dos usuários e exibir o resultado dessas solicitações no navegador *web*;
- `Banco de dados relacional`: Responsável por armazenar todos os dados de cadastros(usuários, métricas personalizadas) dos usuários do CloudMetric e fornecer informações para os agentes;
- *`Banco de dados temporal da nuvem`*: Banco de dados que armazena dados temporais das métricas já existentes da nuvem e das métricas personalizadas. O banco de dados precisa ser do tipo temporal por ter o armazenamento em forma de séries temporais realizado de forma eficiente em comparação a outro tipos de bancos (Relacional e não relacional);
- `Serviço de armazenamento`: Serviço responsável por armazenar os dados das métricas personalizadas do usuários em formato *json (JavaScript Object Notation)*;
- `CloudMetricAgentInput`: Agente responsável por ler os arquivos das métricas personalizadas enviada pelos usuários e realizar a publicação no *broker*.
- `CloudMetricAgentReadBroker`: Possui a função de receber medidas do CloudMetricAgentInput através da assinatura dos tópicos e armazena essas medidas recebidas no banco de dados temporal da nuvem;

- **`Broker`**: Reponsável pelo *Publish* do agente CloudMetricAgentInput e *Subscribe* do agente CloudMetricAgentReadBroker.

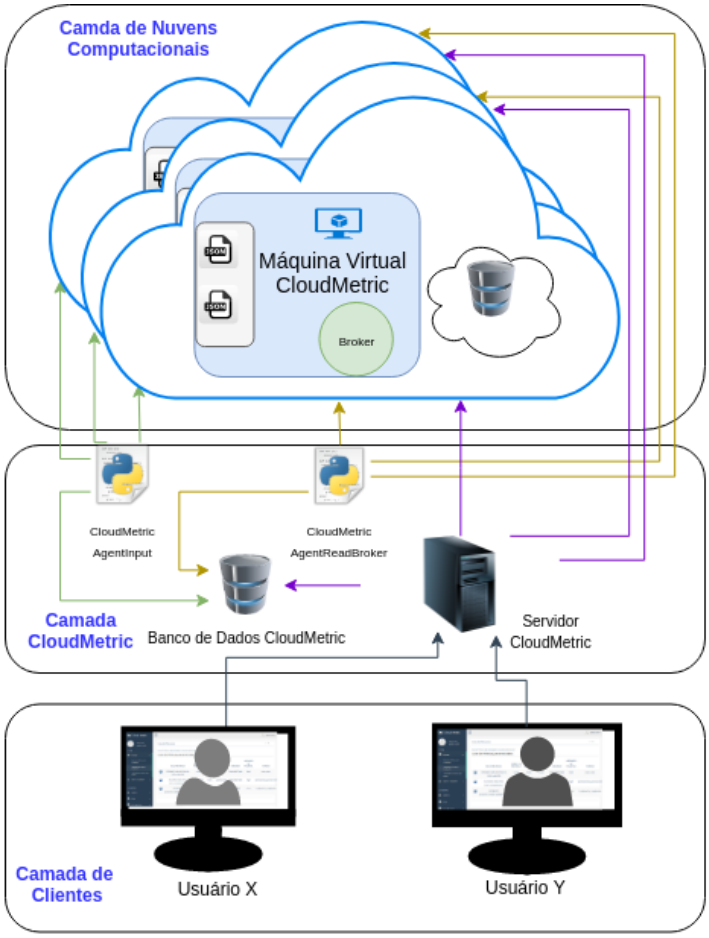


Figura 11 – Visão em camadas da arquitetura do CloudMetric.

Todos os componentes apresentados possuem papéis distintos na arquitetura do CloudMetric. De forma mais detalhada é apresentado na Figura 12 uma visão mais detalhada de todos os componentes que compõem a arquitetura do CloudMetric, as funções de cada componente em uma única nuvem computacional OpenStack, além do ciclo de processamento de métricas personalizadas criadas pelos usuários X e Y através dos 2 agentes. A seguir é descrito as etapas desde o envio da métrica personalizada pelo usuário, até o armazenamento no banco de dados temporal da nuvem.

- `Etapa 1`: Nessa primeira etapa os usuários X e Y enviam às medidas das métricas X e Y1 no formato pré-definido para o serviço de armazenamento da nuvem;

- `Etapa 2`: Na segunda etapa o agente CloudMetricAgentInput periodicamente verifica nos arquivos *json* das métrica X e Y1 a existência de novas medidas. Com a chegada de novas medidas é efetuado o processo da etapa 3;

- `Etapa 3`: Toda medida capturada pelo CloudMetricAgentInput é publicado para ao tópico de interesse no serviço de *Broker*, denominado como Tópico X (usuário X) e tópico Y1(usuário Y);
- *`Etapa 4`*: Na etapa 4 o agente CloudMetricAgentReadBroker consulta as métricas ativas no banco de dados do CloudMetric e subscreve nos tópicos de interesse, esperando pela chegada de novas informações em cada tópico. Com a chegada das informações das métricas X e Y1 é disparado a etapa 5;
- `Etapa 5`: Na última etapa do funcionamento do CloudMetric, todas as métricas capturadas no tópico de interesse no *broker* é salvo no banco de dados temporal da nuvem.

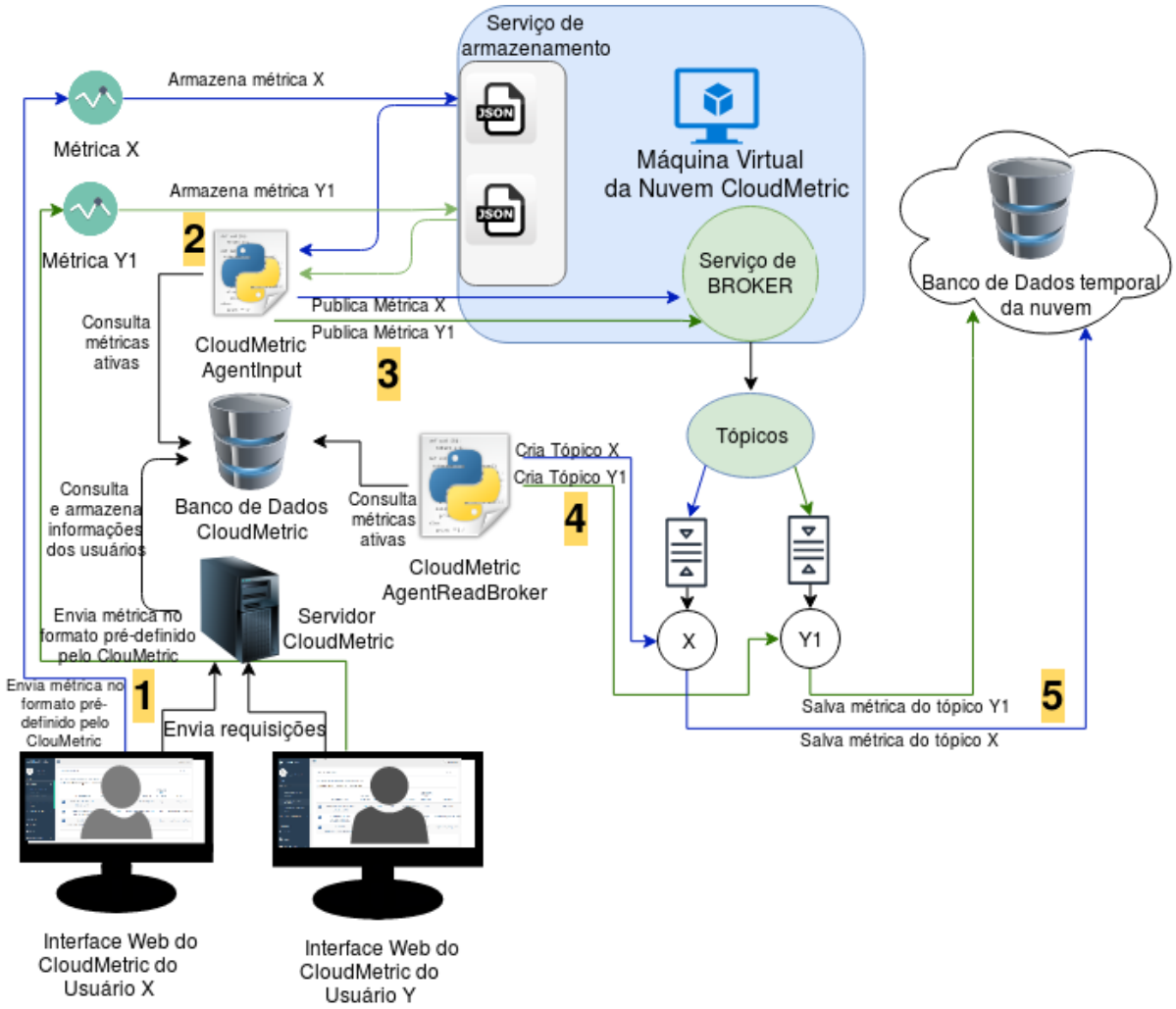


Figura 12 – Funcionamento do CloudMetric com os 2 agentes

Os usuários também conseguem criar e enviar métricas personalizadas de publicando diretamente ao *broker*, sem a necessidade do serviço de armazenamento e do agente CloudMetricAgentInput conforme mostrado na Figura 13. A seguir é descrito as etapas

desde o envio da métrica personalizada de forma direta pelo usuário Y, até o armazenamento no banco de dados temporal da nuvem.

- `Etapa 1`: Nessa primeira etapa o usuário Y publica no tópico Y2 às medidas da métrica Y2 no formato pré-definido, de forma direta ao serviço de *broker*;
- `Etapa 2`: Na etapa 2 o agente CloudMetricAgentReadBroker consulta as métricas ativas no banco de dados do CloudMetric e subscreve no tópico de interesse, esperando pela chegada de novas informações. Com a chegada das informações da métrica Y2 é disparado a etapa 5;;
- `Etapa 3`: Na última etapa do funcionamento do CloudMetric, todas as métricas capturadas no tópico de interesse no *broker* é salvo no banco de dados temporal da nuvem.

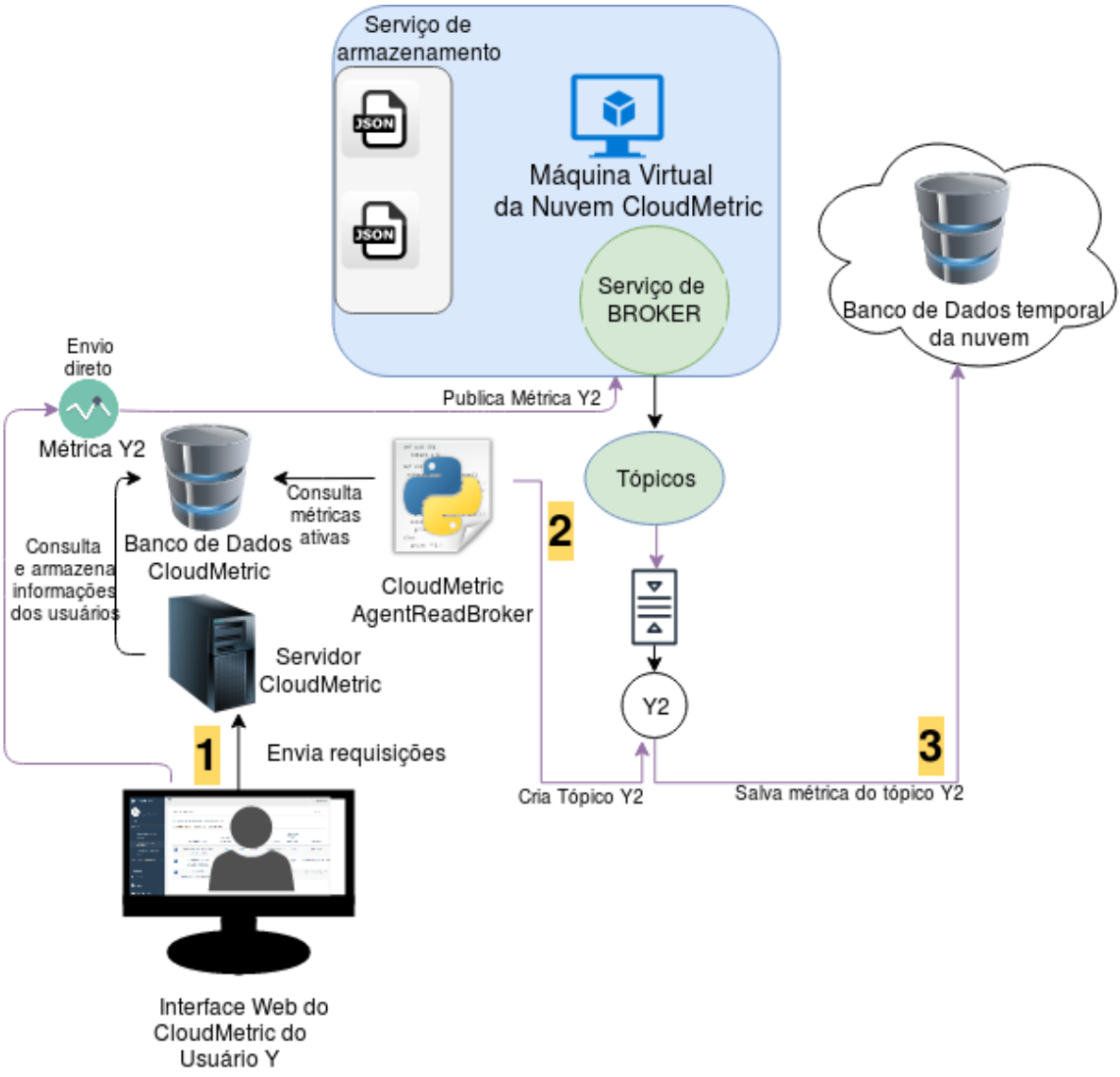


Figura 13 – Funcionamento do CloudMetric com 1 agente

Nas próximas subseções serão apresentadas de forma detalhada todos os componentes que compõem a arquitetura do CloudMetric.

4.1 Interface

No módulo Interface ocorre toda a interação de forma gráfica das funcionalidades do usuário. Essa interface *web*, deve ser acessada por um dispositivo e executada por uma navegador *web*. Conforme mostrado na Figura 12, temos como exemplo dois usuários utilizando o arcabouço CloudMetric. Os usuários, denominados Usuário X e Usuário Y, enviam requisições para o servidor do CloudMetric, que processa as requisições consultando as informações no banco de dados e retornando o resultado dessas consultas para processamento. Como resultado o usuário verá alguma ação ocorrendo em seu navegador, como por exemplo a listagem de métricas personalizadas de cada usuário.

O CloudMetric suporta múltiplos usuários simultâneos, limitados à capacidade de processamento do servidor, do banco de dados, dos agentes e das nuvens acessadas pelos usuários.

4.2 Banco de dados do CloudMetric

O banco de dados do CloudMetric deve ser relacional e possui como função armazenar todas as configurações e registros de acesso do CloudMetric. Possui 3 funções específicas: salvar todas as interações dos usuários com a interface *web*, fornecer periodicamente a lista de métricas ativas e que tenham um caminho de entrada de dados definido para o agente CloudMetricInput e fornecer a lista de métricas ativas para serem processadas nos respectivos tópicos pelo agente CloudMetricAgentReadBroker.

4.3 Serviço de armazenamento

O serviço de armazenamento é um serviço no qual os usuários armazenam os dados das métricas personalizadas de suas aplicações de forma parametrizada em um arquivo no formato *json*. Na Listagem 4.1 é mostrado o modelo padronizado pelo CloudMetric para envio das métricas de usuário utilizando a interface *web*. Esse modelo é um arquivo *json* que possui os campos ID do recurso (gerado no cadastro no CloudMetric), nome, *timestamp* e valor.

O serviço de armazenamento compartilhado deve ser acessado pelo CloudMetric em uma máquina virtual alocada em cada nuvem que for instalado.

A Figura 12 mostra um exemplo: o Usuário X e Y enviam periodicamente a respectivamente as métrica X e Y1 de forma padronizada para o serviço de armazenamento.

Essas métricas serão periodicamente lidas pelo agente CloudMetricAgentInput e repassadas para o *broker*.

Listagem 4.1 – Modelo de arquivo json.

```
{
    "<ID do recurso fornecido pelo CloudMetric>: {
        "<nome da metrica cadastrada no CloudMetric>": [
            {
                "timestamp": "<timestamp>",
                "value": <valor>
            },
            {
                "timestamp": "<timestamp>",
                "value": <valor>
            }
        ]
    }
}
```

4.4 Agentes do CloudMetric

Nas subseções subsequentes serão apresentados e discutidos de forma mais detalhada as funcionalidades dos agentes CloudMetricAgentInput e CloudMetricAgentRedBroker.

4.4.1 CloudMetricAgentInput

O agente CloudMetricAgentInput é responsável por consultar no banco de dados do CloudMetric quais métricas estão ativas e configuradas com um caminho definido no serviço de armazenamento. Sempre que o usuário enviar as métricas para o arquivo no serviço de armazenamento o agente fará uma verificação se o arquivo está no padrão definido pelo CloudMetric. Se estiver no padrão, os dados são empacotados e enviado nos formato de mensagem para o serviço de *broker* do CloudMetric. Se não estiver, altera-se o status da métrica no banco de dados de configuração do CloudMetric para inativo. A inativação da métrica previne que o agente efetue leituras de arquivos que não estejam no padrão especificado. Sempre que o arquivo json for enviado para o *broker*, o seu conteúdo é lido e excluído, garantindo sempre que o agente efetue a leitura dos dados mais recentes.

Quando o usuário publica à métrica de forma direta ao *broker*, o agente CloudMetricAgentInput não realiza os procedimentos de verificação da métrica publicada pelo usuário.

4.4.2 CloudMetricAgentReadBroker

O CloudMetricAgentReadBroker consulta as métricas ativas no banco de dados CloudMetric, e a partir delas cria tópicos no *broker*, subscreve e armazena os valores no

banco de dados temporal. Periodicamente o agente se inscreve nos tópicos criados, captura os valores de medição em cada fila e armazena as informações no banco de dados temporal da nuvem.

As métricas que são publicadas diretamente no *broker* através da Interface *web* pelo usuário são assinadas de forma direta pelo agente CloudMetricAgentReadBroker.

4.5 Serviço de broker

O serviço de *broker* é componente do CloudMetric e reponsável pelo *Publish* (CloudMetricAgentInput) e *Subscribe* (CloudMetricAgentReadBroker). Possui a responsabilidade de receber todas as mensagens do agente *CloudMetricAgentInput* e dos usuários do CloudMetric e efetuar o roteamento das mensagens dos tópicos associados a elas. O *broker* possui uma fila de conexão com o CloudMetricAgentReadBroker para que as mensagens sejam recebidas de acordo com seu tópico.

Os agentes precisam de autenticação e autorização para publicar ou subscrever no *broker*. Caso ocorra qualquer problema de autenticação, o CloudMetricAgentReadBroker muda o estado da métrica para inativa evitando novas tentativas de conexão.

4.6 Banco de dados temporal da Nuvem

O banco de dados temporal da nuvem armazena dados associados a instâncias de tempo. Possui a função de armazenar uma tupla formada por *timestamp* e valor para todas as métricas criadas e gerenciadas pelo CloudMetric.

Oferece tipos de dados temporais e armazena informações relacionadas a métricas em um curto intervalo de tempo, como por exemplo: 1 segundo, 5 segundos, 1 minuto e 1 hora.

Com a arquitetura e funcionamento definidos do CloudMetric foi planejado a implementação de cada componente individualmente. Os detalhes de implementação foram realizados de forma que houvesse os resultados esperados nos 2 (dois) casos de Teste

5 Prova de Conceito e Implementação

A arquitetura conceitual com descrições abstratas sobre o projeto do CloudMetric foi apresentado no capitulo anterior. Este capítulo tem como finalidade complementar essas informações com detalhes sobre a implementação dos elementos utilizados no desenvolvimento do arcabouço. Toda a solução foi projetada utilizando o *software* Astah Community [1], com diagramas UML de casos de Uso e diagrama de classes.

5.1 Diagrama de Casos de Uso e Diagrama de Classes

Nesta seção é descrito as funcionalidades do arcabouço em casos de uso. Na definição da solução do projeto de software foi mapeado um ator principal, nomeado Usuário. Além disso outros dois atores foram mapeados denominados CloudMetricAgentInput e CloudMetricAgentReadBroker. Esses dois últimos atores são nativos do arcabouço e responsáveis pelo processamento das métricas. Na Tabela 2 é apresentado a descrição de cada ator.

No diagrama de casos de uso, apresentado na Figura 14 estão mapeadas todas as funcionalidades do sistema e na Tabela 3 estão descritos todos os casos de uso. É apresentado as funcionalidades do ator Usuário, que representa os usuários que efetivamente utilizam o CloudMetric e dos atores CloudMetricAgentInput e CloudMetricAgentReadBroker, que representam os atores em forma de agentes do CloudMetric.

Na descrição de casos de uso foi utilizado o termo "Manter"com objetivo de apresentar as quatro operações básicas: Criar, Remover, Listar e Editar.

Na Figura 15 é apresentando o diagrama de classes, exibindo toda à estrutura

Tabela 2 – Atores

Nome	**Descrição**
Usuário	Ator que utiliza todas as funcionalidades do CloudMetric
CloudMetricAgentInput	Ator do sistema que processa as métricas personalizadas, coletando os dados em um servidor de arquivos compartilhados Samba, disponível na nuvem, e enviando para o *broker* do RabbitMQ.
CloudMetricAgentReadBroker	Ator do sistema que processa as métricas personalizadas, assinando os tópicos, capturando as medições no *broker* do RabbitMQ e salvando as medições na nuvem.

[1] http://astah.net/editions/community

Tabela 3 – Descrição dos Casos de Usos

Nome	**Descrição**
[UC-A01] Login	O sistema deverá permitir que o usuário efetue o login no sistema
[UC-A02] Trocar Senha	O sistema deverá permitir que o usuário troque de senha.
[UC-B01] Manter Métrica Personalizada	O sistema deverá permitir que o usuário crie, remova, liste e edite as métricas personalizadas.
[UC-C01] Manter Nuvem	O sistema deverá permitir que o usuário crie, remova, liste e edite as nuvens computacionais monitoradas pelo ClodMetric no sistema.
[UC-D01] Visualizar Nuvens Cadastradas	O sistema deverá permitir que o usuário visualize as nuvens computacionais cadastradas no sistema.
[UC-D01.1] Visualizar recursos	O sistema deverá permitir que o usuário visualize os recursos nativos monitorados nas nuvens cadastradas no sistema.
[UC-D01.1.1] Visualizar métricas	O sistema deverá permitir que o usuário visualize as métricas dos recursos selecionados no sistema.
[UC-D01.1.1.1] Visualizar métricas no Grafana	O sistema deverá permitir que o usuário visualize métricas geradas de forma automática no painel do Grafana.
[UC-D01.1.1.2] Exportar métrica	O sistema deverá permitir que os usuários exportem métricas para o navegador *web* no formato json.
[UC-E01] Visualizar Métricas Personalizadas	O sistema deverá permitir que as métricas personalizadas criadas pelo usuário sejam visualizadas.
[UC-E01.1] Exportar métrica personalizada	O sistema deverá permitir que as métricas personalizadas criadas pelo usuário sejam exportadas para o navegador *web* no formato json.
[UC-E01.2] Visualizar métrica personalizada no Grafana	O sistema deverá permitir que as métricas personalizadas criadas pelo usuário sejam geradas de forma automática no painel do Grafana.
[UC-F01]Manter Recurso	O sistema deverá permitir que o usuário crie, remova, liste e edite recursos no sistema.
[UC-G01]Manter Tipo de Recurso	O sistema deverá permitir que o usuário crie, remova, liste e edite tipos de recursos no sistema.
[UC-H01] Manter *broker*	O sistema deverá permitir que o usuário crie, remova, liste e edite *brokers* no sistema..
[UC-I01] Ler dados do arquivo json da métrica personalizada	O sistema deverá permitir que o usuário leia dados de arquivos json no serviço de armazenamento do CloudMetric.
[UC-I01.1] Enviar dados para o *broker*	O sistema deverá permitir que o usuário envie os dados lidos para o *broker* do CloudMetric.
[UC-J01] Assinar tópico de métrica personalizada no *broker*	O sistema deverá permitir que o usuário assine tópicos no *broker* do CloudMetric.
[UC-J01.1] Armazenar valor lido no tópico para o banco de dados temporal	O sistema deverá permitir que o usuário leia as mensagens das filas no *broker* do CloudMetric e armazene os dados no banco de dados temporal.

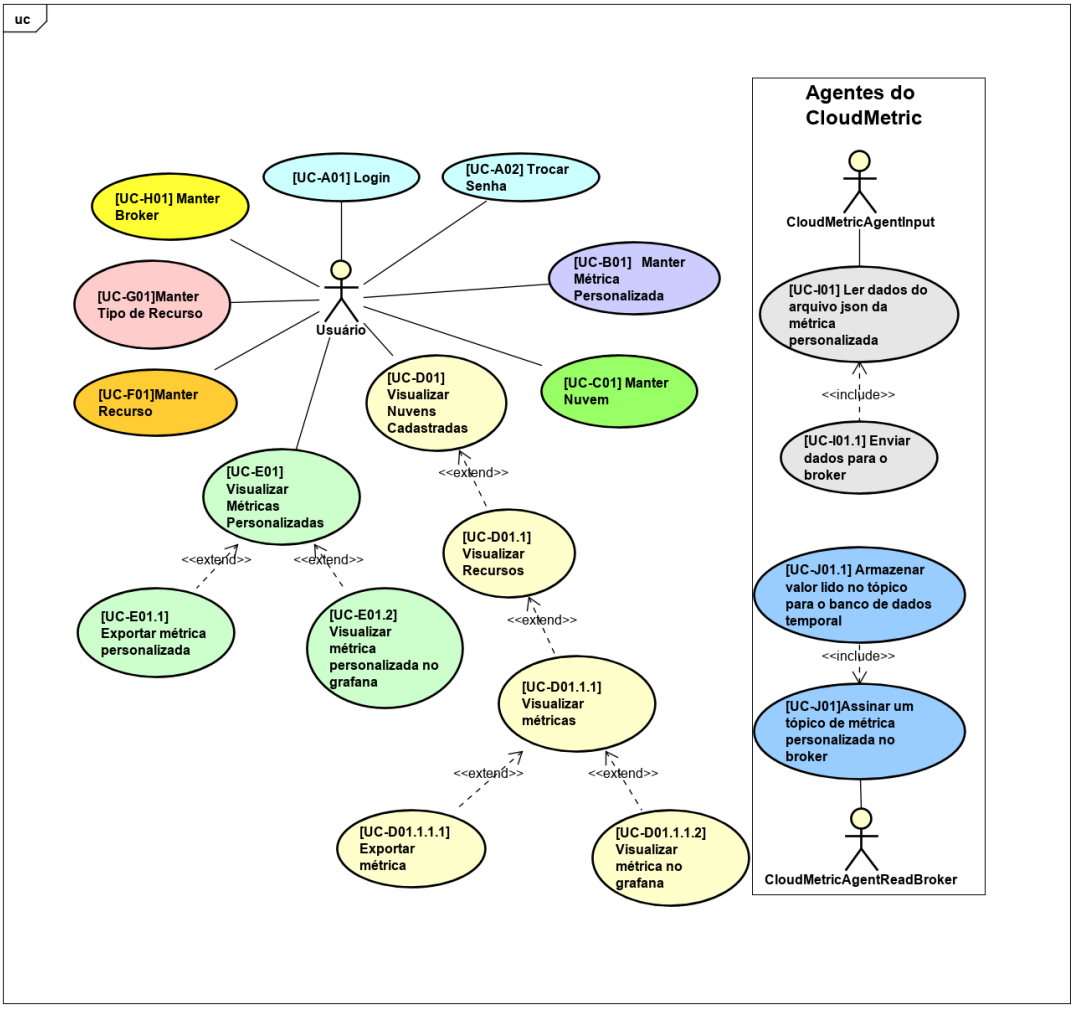


Figura 14 – Diagrama de Casos de Uso

utilizada para a implantação do CloudMetric por meio de classes, atributos, operações e relações.

5.2 Recursos Tecnológicos

Os recursos tecnológicos utilizados para a implementação do arcabouço CloudMetric são descritos nesta seção. O CloudMetric utilizou o *framework* Django[2] de aplicações *web*. O Django é um conjunto de componentes escrito em Python que que utiliza o padrão de desenvolvimento MVT(Modelo, Visão e Template).

Na camada de Template, que contém os arquivos de Template do sistema, foram utilizados o Gentelella[3], juntamente com componentes visuais de Javascript e CSS (*Cascading Style Sheets*), como JQuery[4], Bootstrap[5], Datatable.js e datetimepicker.js. Na camada

[2]https://www.djangoproject.com/
[3]https://github.com/ColorlibHQ/gentelella
[4]https://github.com/ColorlibHQ/gentelella
[5]https://getbootstrap.com/

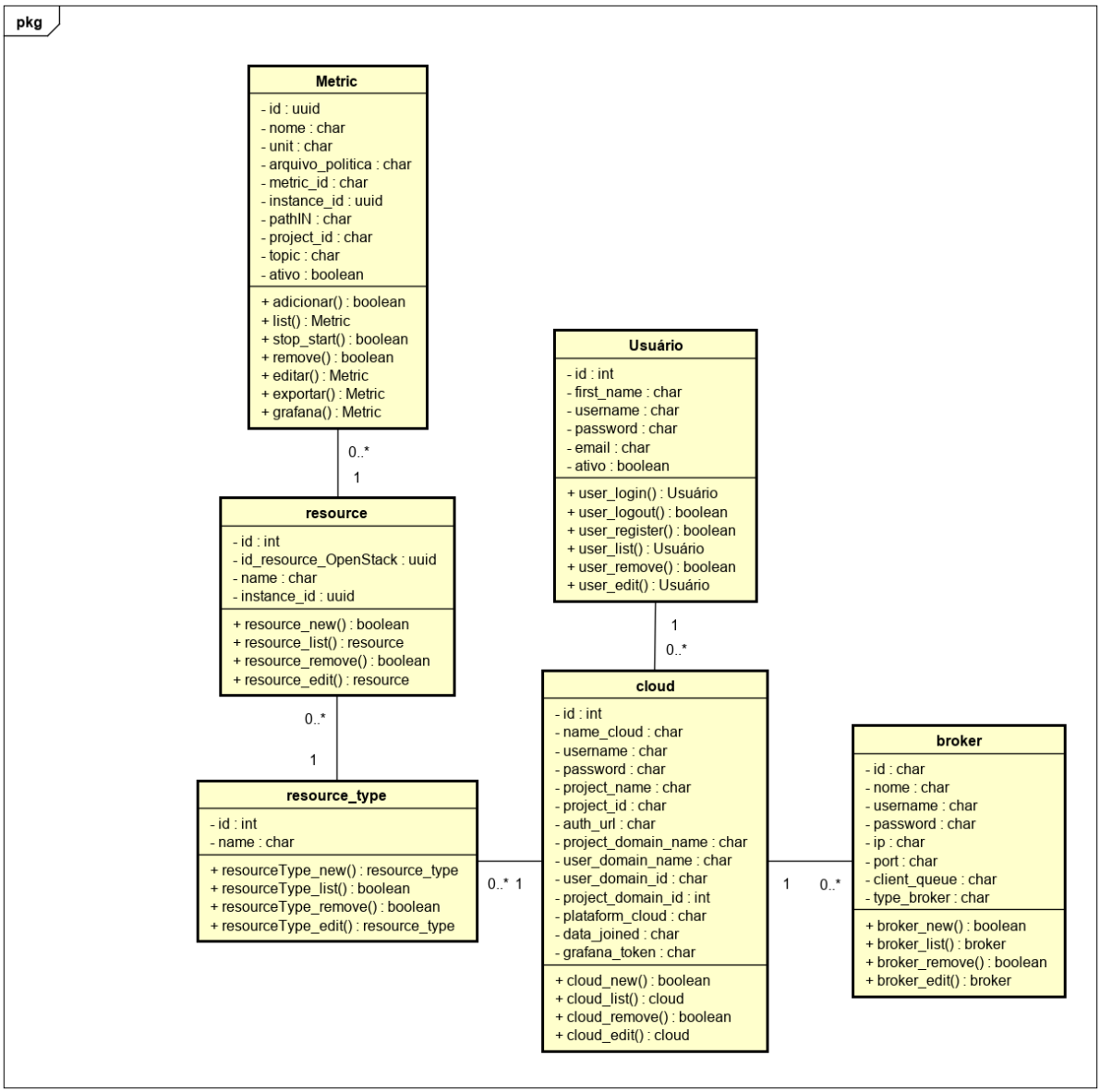


Figura 15 – Diagrama de Classes

de Visão, que contém o controle do arcabouço, foram utilizadas as bibliotecas do Django. A camada de Modelo contém as entidades mapeadas pelo Django e a interação das entidades com o banco de dados. Para realizar essa interação foi utilizada a *QuerySet* API, que possui a funcionalidade de criar, recuperar, atualizar e excluir objetos instanciados no arcabouço.

O banco de dados utilizado para armazenamento dos dados relacionais foi o MySQL [6]. Além disso, diversas outras APIs foram utilizadas tais como: OpenStack SDK [7], Gnocchi client[8] e Grafana API.

[6] https://www.mysql.com/
[7] https://docs.openstack.org/openstacksdk
[8] https://gnocchi.xyz/gnocchiclient/api.html

No agente CloudMetricAgentInput foi utilizado a linguagem Python como desenvolvimento do agente, juntamente com as bibliotecas de Pika(comunicação com o RabbitMQ), MysQLConnector(comunicação com o banco de dados), Python SMB(comunicação com o servidor samba) e *Thread*.

O agente CloudMetricAgentReadBroker foi criado em forma de *script* Python utilizando como biblioteca de desenvolvimento Pika(comunicação com o RabbitMQ), MysQLConnector(comunicação com o banco de dados) e gnocchiclient(comunicação com o banco de dados temporal Gnocchi) e keystoneauth1(autenticação com o OpenStack).

Como recursos tecnológicos externos foram utilizados o visualizador de métricas Grafana, o software de mensageria RabbitMQ e o servidor de armazenamento Samba.

6 Avaliação do CloudMetric

Com o intuito de demonstrar as funcionalidades previstas no projeto do CloudMetric foram realizados 2 (dois) Casos de Teste em diferentes cenários: i) Controle de Trajetória do Robô em um Espaço Inteligente (FUTEBOL, 2017), ii) Monitoramento de métricas de serviço em um servidor *web* Apache. Esses Casos de Teste estão descritos nas seções seguintes, demostrando todas as funcionalidades do CloudMetric utilizadas no monitoramento de métricas nativas do OpenStack com a criação de métricas personalizadas.

6.1 Descrição do Caso de Teste Controle de Trajetória do Robô em um Espaço Inteligente

Atualmente há uma nova geração de aplicações de robótica, denominadas "robótica como serviço" em espaços inteligentes, que utilizam controle em tempo real, remoto, e são aplicadas em diversas áreas, tais como terapia de reabilitação, localização e navegação de robôs e robótica social. Espaços inteligentes podem ser descritos como locais que tem sensores conectados a rede (câmeras e microfones, por exemplo) que coletam informações do ambiente e repassam a um sistema supervisor que analisa as informações, habilitam tomadas de decisão, interações entre usuários e a execução de uma ação de um dispositivo conectado a rede (robôs, dispositivos móveis, monitor de informações) (HVIZDOš et al., 2017).

A Figura 16 apresenta a arquitetura física do espaço inteligente, na qual pode-se notar que no ambiente que o robô se encontra há apenas um conjunto mínimo de recursos necessários para este realizar uma tarefa de seguimento de trajetória: câmeras que realizam a aquisição das imagens utilizadas para a localização e uma rede sem fio utilizada para enviar comandos de velocidade e direção para o robô. A tarefa de seguimento de trajetória consiste em manter o robô em uma dada posição no espaço-tempo com uma determinada velocidade. Por conta disso, dado uma taxa de amostragem fixa das câmeras, há o requisito de se computar as etapas de processamento de imagem para a localização, e cálculo da velocidade, além do envio dos comandos de controle para o robô, em um tempo menor ou igual ao período de amostragem. Na figura os componentes que realizam o processamento das imagens e determinam os comandos de trajetória do robô estão todos alocados na nuvem, ou seja, em um *datacenter* remoto.

A infraestrutura de máquinas virtuais realizada no Testbed da UFES foi criada de acordo com a Figura 17 e é composta por 1 (um) serviço servidor de imagem mjpeg (is-mjpeg-server), 2 (dois) *gateways* das câmeras (is-camera-gateway) que as configuram

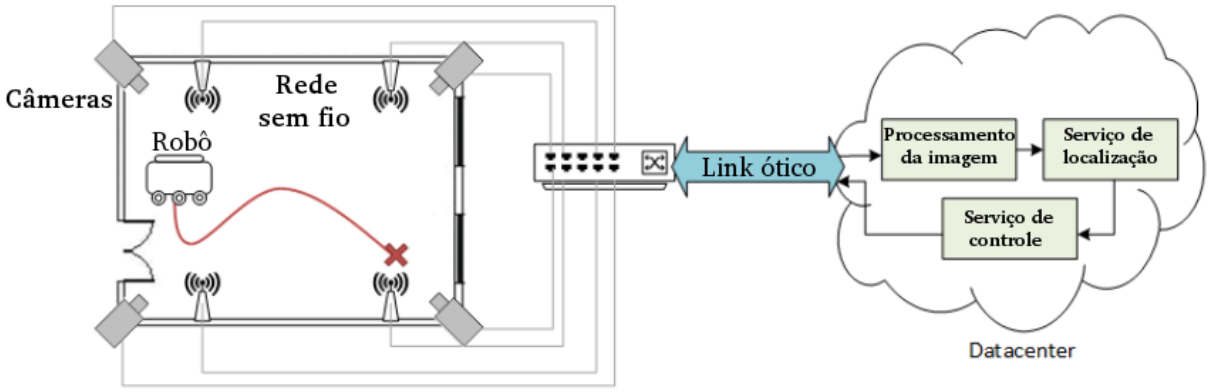


Figura 16 – Descrição do cenário do Caso de Teste.

de acordo com a necessidade da aplicação, 1 (um) serviço de troca de mensagens em nível de aplicação (is-rabbit-mq), 1 (um) controlador do robô (is-robot-controller), 1 (um) serviço de processamento de imagens e localização do robô (is-image-processing) e 1 (um) Controlador SDN (is-sdn-controller).

As máquinas virtuais criadas executam serviços que geram métricas personalizadas de recurso robótico (erro de trajetória e velocidade linear do robô) e *wifi* (relação sinal ruído). As métricas personalizadas associadas aos recursos são importantes por que trazem informações em relação a qualidade de sinal entre o *wifi* e o robô, além de verificar se a nuvem forneceu recursos suficientes para os componentes da aplicação. Através do dimensionador de recursos de nuvem (GIACOMELLI, 2019) integrado ao CloudMetric, a nuvem forneceu recursos suficientes para os componentes da aplicação.

Além das máquinas virtuais criada de acordo com a Figura 17 temos 1 (uma) máquina virtual CloudMetric responsável por armazenar os arquivos em formato json das métricas personalizadas e hospedar o serviço de troca de mensagens RabbitMQ do arcabouço CloudMetric. Essa infraestrutura do espaço inteligente foi construída utilizando-se a plataforma *IaaS* OpenStack *multi-nó* na versão Queens[1].

6.1.1 Monitoramento sem o uso do CloudMetric

Com o intuito de demostrar a importância da utilização do CloudMetric foi realizado o monitoramento da máquina virtual de processamento de imagens e localização do robô *is-image-processing* da Figura 17 com os recursos nativos oferecido pelos serviços do OpenStack. Esses recursos são as APIs de usuários do OpenStack que se comunicam com os serviços de Identidade (Keystone), Computação (Nova), Telemetria (Ceilometer), banco de dados temporal (Gnocchi) e a Dashboard do OpenStack (Horizon).

Para realizar o monitoramento de modo nativo são necessários seguir os passos

[1] https://www.openstack.org/software/queens/

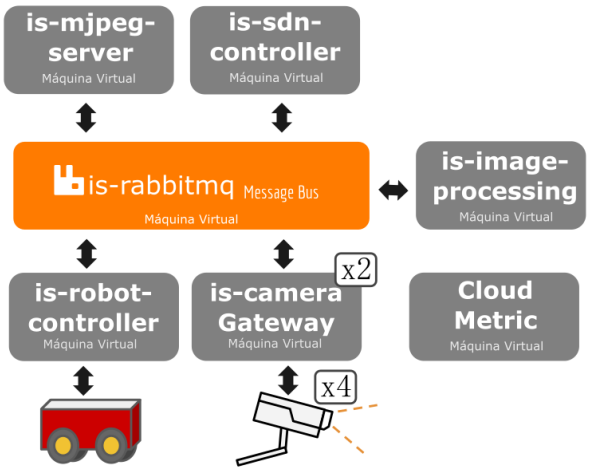


Figura 17 – Infraestrutura de máquina virtuais no OpenStack.

exibidos no fluxograma da Figura. 18.

O primeiro passo do fluxograma foi o acesso ao *Dashboard* de administração (Horizon) utilizando as credenciais do usuário denominado "exp22". Este usuário pertence ao grupo exp22 possuindo todas as permissões de acesso as máquinas virtuais alocadas a esse projeto.

Ao efetuar o *login* de forma correta no *Dashboard* do OpenStack é necessário baixar o arquivo OpenStack RC v3 no menu de opção dos usuários, conforme exibido na Figura 19. Esse arquivo fornece todas as variáveis de ambiente permitindo que o OpenStackClient[2] se comunique com os serviços disponíveis na nuvem.

No terceiro passo do fluxograma é necessário realizar o acesso ao controlador do OpenStack por um terminal utilizando o protocolo *Secure SHell (SSH)*. Referente ao passo 3, na Listagem 6.1, podemos verificar na linha 1 o comando para acessar virtualmente o controlador do OpenStack, através do usuário "stack"e IP do controlador (10.60.0.1). O usuário "stack"refere-se ao usuário de sistema do controlador e não possui ligação com o usuário utlizado no arcabouço CloudMetric.

Nas linhas subsequentes, avançando para o passo 4 do fluxograma temos a exportação de todas as variáveis de ambiente no terminal através do comando *export*. Podemos destacar como variáveis importantes para execução dos comandos disponíveis do OpenStackClient, as variáveis Auth URL (Linha 2), ID do Projeto (Linha 3), Nome do Projeto (Linha 4), usuário e senha do usuário OpenStack (Linhas 11 e 12).

[2]https://docs.openstack.org/python-openstackclient/queens/

Figura 18 – Fluxograma do monitoramento sem CloudMetric.

Listagem 6.1 – SSH e variáveis de ambiente no terminal

```
1  pablo@pablo-Nerds: ~$ ssh -X stack@10.60.0.1
2  stack@controller: ~$ export OS_AUTH_URL=http://10.60.0.1:5000/v3
3  stack@controller: ~$ export OS_PROJECT_ID=135a7c7de8ee4b86b3d4a1e4ef39d679
4  stack@controller: ~$ export OS_PROJECT_NAME="EXP22"
5  stack@controller: ~$ export OS_USER_DOMAIN_NAME="Default"
6  stack@controller: ~$ if [ -z "$OS_USER_DOMAIN_NAME" ]; then unset
       OS_USER_DOMAIN_NAME; fi
7  stack@controller: ~$ export OS_PROJECT_DOMAIN_ID="default"
8  stack@controller: ~$ if [ -z "$OS_PROJECT_DOMAIN_ID" ]; then unset
       OS_PROJECT_DOMAIN_ID; fi
9  stack@controller: ~$ unset OS_TENANT_ID
10 stack@controller: ~$ unset OS_TENANT_NAME
11 stack@controller: ~$ export OS_USERNAME="exp22"
12 stack@controller: ~$export OS_PASSWORD="stack"
13 stack@controller: ~$export OS_REGION_NAME="RegionOne"
14 stack@controller: ~$ if [ -z "$OS_REGION_NAME" ]; then unset OS_REGION_NAME; fi
15 stack@controller: ~$ export OS_INTERFACE=public
16 stack@controller: ~$ export OS_IDENTITY_API_VERSION=3
```

Figura 19 – Menu de opções do usuário exp22.

Em seguida, no Passo 5 temos que buscar o ID da VM a qual se monitora. A busca desse ID é realizada consultando o ID, nome e status das máquinas virtuais, conforme exibido na Listagem 6.2, linha 1. Como resultado desse comando temos a listagem de todas as máquinas virtuais alocadas na infraestrutura do OpenStack, exibida na Figura 17. A máquina virtual utilizada nos próximos passos do fluxograma será a "is-image-processing", com o respectivo ID 2753641b-3aaf-4085-a760-6ec2a0574538.

Listagem 6.2 – Listagem de máquina virtuais

```
1  stack@controller: ~$ openstack server list
2  +--------------------------------------+--------------------------+--------+
3  | id                                   | Name                     | Status |
4  +--------------------------------------+--------------------------+--------+
5  | 094acc20-fa5c-445e-9892-77a1f96ff540 | CloudMetric              | ACTIVE |
6  | 42f22eba-8cef-4d15-907a-1a8d1e86c1b0 | is-rabbitmq              | ACTIVE |
7  | b804364b-5192-473a-bbe5-4a1005ee762c | is-mjpeg-server          | ACTIVE |
8  | 906f5fe8-be88-4b7c-b6e5-5d49ca6af9e6 | is-sdn-controller        | ACTIVE |
9  | 4dc76942-5386-4099-bc77-fd390b17344e | is-robot-controller      | ACTIVE |
10 | 63e72e97-7b1e-4afe-9b29-09c08f16a5d6 | is-camera-gateway-23     | ACTIVE |
11 | a1cbef3b-7f3a-4e7b-90b2-f2eac4530f3a | is-camera-gateway-01     | ACTIVE |
12 | 2753641b-3aaf-4085-a760-6ec2a0574538 | is-image-processing      | ACTIVE |
13 +--------------------------------------+--------------------------+--------+
```

Com o ID do recurso selecionado, no Passo 6 foi buscado os IDs de todas as métricas associadas à VM "is-image-processing", conforme mostrado na Listagem 6.3, linha 1. O resultado desse comando está exibido nas linhas 10 a 41. Para os próximos passos do fluxograma serão exibidos as informações da métrica "Utilização de CPU (cpu_util)"(Linha 14 na Listagem 6.3), com ID "0b2d5886-1f6a-4f76-969b-bfccf0a280a3".

Listagem 6.3 – Listagem de métricas do recurso da VM is-image-processing

```
stack@controller: ~$ openstack metric resource show 2753641b-3aaf-4085-a760-6
    ec2a0574538
+-----------------+-----------------------------------------------------------+
| Field           | Value                                                     |
+-----------------+-----------------------------------------------------------+
| created_proj_id | 6771498c0f024b6fa6764e7d8a22d600                          |
| created_user_id | ff1a87374d504611b5a69fc6a78d485f                          |
| creator         | ff1a87374d504611b5a69fc6a78d485f:6771498c0f024b6fa6764e7d8a |
| ended_at        | None                                                      |
| id              | 2753641b-3aaf-4085-a760-6ec2a0574538                      |
| metrics         | compute.instance.boot.time: 08185063-fcd9-4403-a688-5863aa8 |
|                 | cpu.delta: 8486577b-5941-4bcd-9ff1-5259cc6c4724           |
|                 | cpu: 70236dc0-7c4a-4516-9e78-08690c762149                 |
|                 | cpu_l3_cache: adaeba3e-d78b-447c-bd62-46c6f4cdcff6        |
|                 | cpu_util: 0b2d5886-1f6a-4f76-969b-bfccf0a280a3            |
|                 | disk.allocation: acefeac1-5166-4394-b14f-2906564df254     |
|                 | disk.capacity: 6b5c9aab-1b58-476e-bd11-3e65b887dd26       |
|                 | disk.ephemeral.size: 7786ba44-631e-4493-a80c-29d878ac05da |
|                 | disk.iops: 879ebdcc-99f9-45d0-af83-77ef458bfc72           |
|                 | disk.latency: 9aea0b0c-9da9-497e-abc3-0f7b5e22e39e        |
|                 | disk.read.bytes.rate: d8da01bb-85ce-4437-a314-31a7629b87be |
|                 | disk.read.bytes: d143a5be-77b7-47d9-a6ba-5ae4a883eee2     |
|                 | disk.read.requests.rate: b42c6e95-fae2-4480-bc0b-8c66da11347|
|                 | disk.read.requests: 36380bf0-ead5-4426-9025-26fd39b7c0be  |
|                 | disk.root.size: 0632915e-9143-4eba-a6ca-642344f852a2      |
|                 | disk.usage: a117bb04-d635-467c-a768-8c91dd40d852          |
|                 | disk.write.bytes.rate: b5ed440b-ef60-486e-b2aa-83e08fdae09d |
|                 | disk.write.bytes: 29fbefaf-387a-440d-917d-fabae0ef5e1a    |
|                 | disk.write.requests.rate: 51bc5932-404a-40ad-8861-2e2372ff6a|
|                 | disk.write.requests: 3047da88-8fd9-46de-a582-7e22a40fd110 |
|                 | memory.bandwidth.local: 6f8ef47d-f3a5-48a5-a140-846860b604fb|
|                 | memory.bandwidth.total: 76bd6998-6acb-4ad7-94ae-40b24755bd96|
|                 | memory.resident: be08fe10-ee1b-4f40-9580-57c4a910dede     |
|                 | memory.swap.in: 20e19903-6ef5-4be4-b4cf-09cd40ac429c      |
|                 | memory.swap.out: 6d1e206f-d878-41d8-ad16-2b8549dca28e     |
|                 | memory.usage: c99b6747-6d0b-458a-a27e-a6f50d393a76        |
|                 | memory: 49990ced-0de7-447e-b9aa-f5ed52ddd426              |
|                 | perf.cache.misses: ab06211a-065a-477d-a118-6cd2b67eae18   |
|                 | perf.cache.references: 6f00671a-6942-4e06-afef-9a079ee428c3 |
|                 | perf.cpu.cycles: bae44132-eeab-4139-a855-9ef6ba5a4b05     |
|                 | perf.instructions: e1b33934-7fc2-458e-b2d0-5a60b03bf50f   |
|                 | vcpus: 00f7bbd5-aea9-4508-ba41-e205d42043bb               |
| original_res_id | 2753641b-3aaf-4085-a760-6ec2a0574538                      |
| project_id      | 135a7c7de8ee4b86b3d4a1e4ef39d679                          |
| revision_end    | None                                                      |
| revision_start  | 2019-08-31T20:00:07.098418+00:00                          |
| started_at      | 2019-08-30T18:26:09.826875+00:00                          |
| type            | instance                                                  |
| user_id         | 7e709a124e364963b625d83232124352                          |
+-----------------+-----------------------------------------------------------+
```

Com a métrica utlização da CPU do recurso "is-image-processing"selecionada é possível efetuar duas ações diferentes no fluxograma da 18: Exibir as medidas pelo ID da

métrica no formato json (Passo 7) ou configurar uma Dashboard e visualizar a métrica no Grafana (Passos 8,9 e 10).

Na primeira ação do Passo 7 foi exibido 5 segundos de medições da métrica cpu_util no intervalo 01/09/2019 16:05:00 à 01/09/2016 16:05:05. Na Listagem 6.4, temos o comando utilizado no OpenStackClient (linha 1), e o arquivo no formato json nas linhas subsequentes. Como resultado foi retornado a média da última hora, último minuto e 5 medidas de 1 segundo com o intervalo de tempo selecionado.

Listagem 6.4 – Medições da métrica CPU_UTIL da VM is-image-processing

```
1  stack@controller: ~$ openstack metric measures show 0b2d5886-1f6a-4f76-969b-
       bfccf0a280a3 --start 2019-09-01T16:05:00-03:00 --stop 2019-09-01T16
       :05:05-03:00 -f json
2  [
3    {
4      "timestamp": "2019-09-01T16:00:00-03:00",
5      "value": 60.3843262410245,
6      "granularity": 3600.0
7    },
8    {
9      "timestamp": "2019-09-01T16:05:00-03:00",
10     "value": 60.32492045326,
11     "granularity": 60.0
12   },
13   {
14     "timestamp": "2019-09-01T16:05:00-03:00",
15     "value": 60.3085623329,
16     "granularity": 1.0
17   },
18   {
19     "timestamp": "2019-09-01T16:05:01-03:00",
20     "value": 61.1070769124,
21     "granularity": 1.0
22   },
23   {
24     "timestamp": "2019-09-01T16:05:02-03:00",
25     "value": 60.9161185656,
26     "granularity": 1.0
27   },
28   {
29     "timestamp": "2019-09-01T16:05:03-03:00",
30     "value": 57.0107457603,
31     "granularity": 1.0
32   },
33   {
34     "timestamp": "2019-09-01T16:05:04-03:00",
35     "value": 59.3337486622,
36     "granularity": 1.0
37   }
38 ]
```

Da mesma forma que é possível visualizar as medições das métricas em formato json, podemos utilizar os dados dos Passos 5 e 6 para montar um Dashboard de visualização das medições no Grafana.

Para a visualização da métrica no Grafana, o usuário precisa acessar com suas credenciais de acesso e criar um painel com as informações fornecidas nos Passos 5 e 6 do fluxograma da Figura 18. Na Figura 20, para que esse painel seja montado é necessário inserir as seguintes informações: Data Source = Gnocchi, Query Type = resource search, Resource Type = instance, Query = ID = 2753641b-3aaf-4085-a760-6ec2a0574538 and ended_at=None , Metric Regex=cpu_util, Aggregator=mean, granularidade=1, Label=Utilização de CPU e Unidade=%, está ultima configurada na aba de eixos. Por fim no passo 10, após a configuração correta das variáveis no Passo 9 temos a visualização da métrica na Figura 20.

A partir do Passo 7 ou Passo 10 do fluxograma da Figura 18 podemos encerrar o monitoramento do OpenStack, ou visualizar outra métrica do recurso is-image-processing, mostrado na Listagem 6.3. É necessário refazer a sequência de passos 7, para visualizar a métrica em formato json ou Passos 8,9,10 para configurar e visualizar a métrica no Grafana. Em caso de monitorar outra máquina virtual da infraestrutura da Figura 17 será necessário voltar novamente ao Passo 5.

6.1.2 Monitoramento usando o CloudMetric

Nesta subseção será demostrado como é realizado o monitoramento utilizando o arcabouço CloudMetric. No fluxograma da Figura 21 é exibido o conjunto de passos necessários para monitorar as máquinas virtuais.

No Passo 1 do fluxograma é realizado o cadastro do usuário "admin", no arcabouço CloudMetric conforme mostrado na Figura 22(a). É necessário preencher informações básicas como *username*, nome completo, email e senha. Após efetuado o cadastro do usuário é realizado o *login* no CloudMetric, conforme exibido na Figura 22(b). O fluxo do Passo 1 é obrigatório somente nos casos em que o usuário não tenha cadastro. Se já houver cadastro ativo, pode seguir direto para o Passo 2.

Com o usuário autenticado no CloudMetric é necessário cadastrar ao menos uma nuvem OpenStack (Passo 3 do fluxograma) acessando o Menu **Nuvem -> Adicionar** exibido na Figura 23. Os campos preenchidos no formulário da Figura 23 são os campos obtidos a partir do *Dashboard* do OpenStack, especificamente no arquivo OpenStack RC File v3, com o acréscimo do campo *Token* do Grafana e nome da nuvem criada pelo usuário, no exemplo "amprod".

Com o término de cadastro da nuvem OpenStack denominada "amprod", o usuário "admin"irá acessar o menu **Métricas -> Listar Métricas Criadas pela Nuvem**,

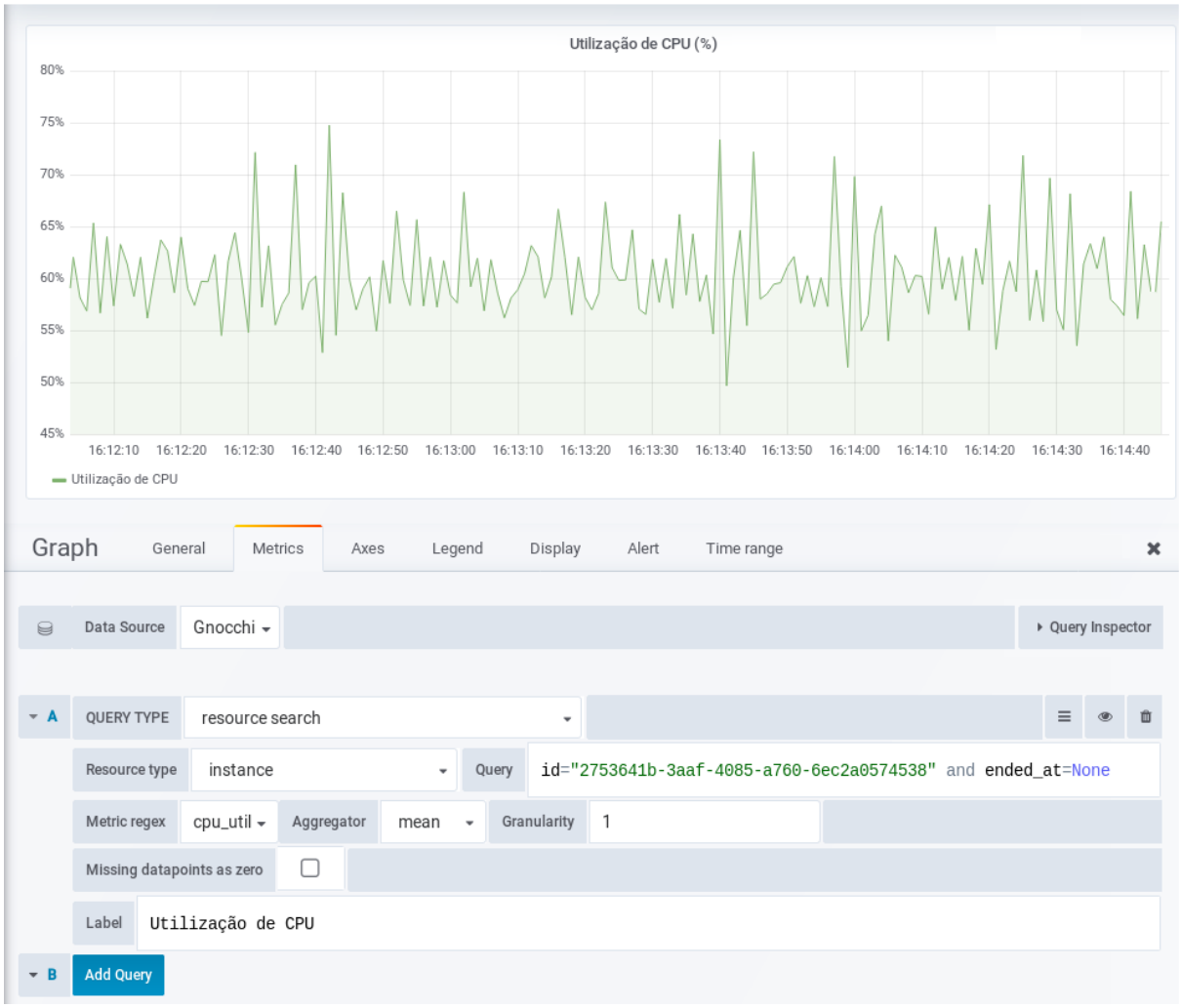


Figura 20 – Configuração e visualização da métrica CPU_Util no Grafana.

conforme mostrado na Figura 24, correspondendo ao Passo 4 do fluxograma da Figura 21.

Ao escolher a nuvem "amprod"no CloudMetric conforme mostrado na Figura 24 é exibido a lista de máquinas virtuais gerenciadas pelo usuário admin, de forma gráfica e simples. Para listar as métricas de qualquer recurso da lista é necessário somente uma ação de clique no botão selecionar associado ao recurso da, como mostrado na Figura 25. Essa abordagem de selecionar o recurso é mais simples em comparação a abordagem anterior a qual o usuário precisava copiar o ID do recurso e executar o comando na Linha 1 da Listagem 6.3.

Na Figura 26 temos a lista de métricas do recurso is-image-processing (Passo 5), resultado da ação de clique no botão selecionar recurso na VM is-image-processing da Figura 24. A Figura exibe uma pequena amostra das métricas disponíveis e associadas a esse recurso para ilustrar, de maneira mais clara, a facilidade de uso do CloudMetric.

Com as funcionalidades visualizadas como descrito no Passo 7 do fluxograma é possível seguir dois caminhos distintos para a visualização da métrica: visualizar a métrica

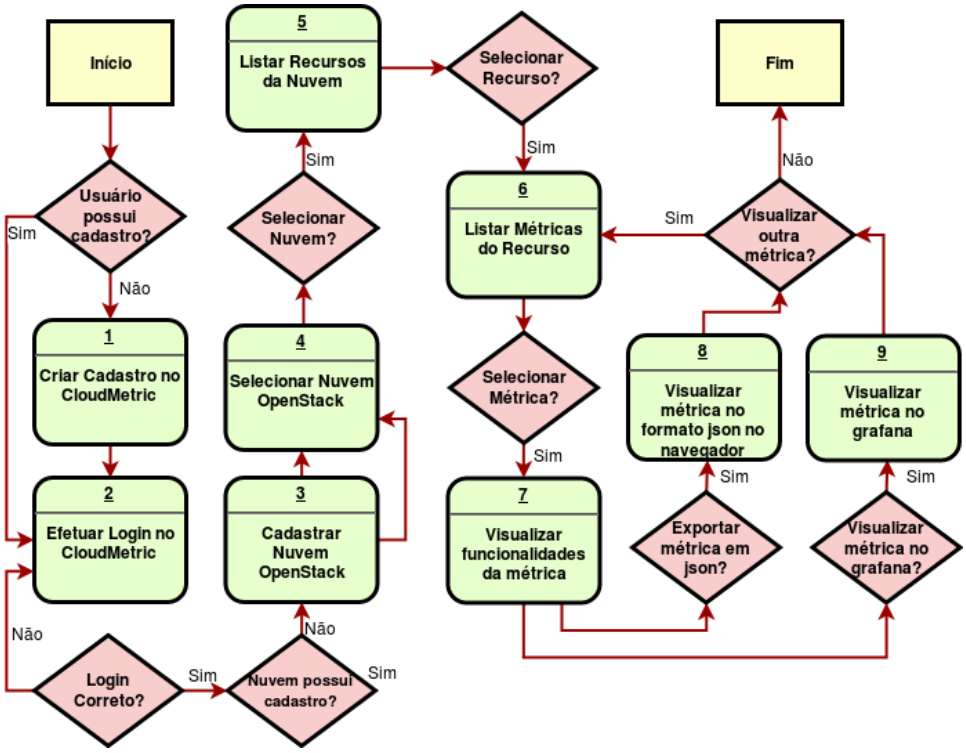


Figura 21 – Fluxograma do monitoramento com CloudMetric.

no Grafana de forma automatizada (Passo 8) ou visualizar no formato json no navegador *web* (Passo 9).

Ao efetuar o comando de *click* no botão "exportar"no formato json da métrica cpu_util da Figura 26, será carregado no CloudMetric uma janela para configurar data e hora inicial e final no formato de um calendário, como mostrado na Figura 27(a). Após realizada a inserção do período de interesse foi efetuado o comando de *click* no botão exportar, gerando o arquivo json com a lista de medidas da métrica cpu_util do recurso is_image_processing, conforme demonstrado na Figura 28(a).

Retornando a Figura 26 pode-se também criar, de forma totalmente automática, a visualização das métricas no Grafana. Ao efetuar o comando de *click* no botão visualizar no Grafana da métrica cpu_util, será carregada a visualização dessa métrica em tempo real, sem que o usuário precise se preocupar em configurar as variáveis no painel do OpenStack. Com o uso do CloudMetric, o usuário terá, de forma automática e instantânea, todas as informações da Figura 20 para quaisquer de suas métricas.

Após concluir à visualização das medidas da métrica cpu_util pode-se visualizar outra métrica retornando ao Passo 4 do fluxograma da Figura 21, selecionando novamente o recurso is-image-processing na Figura 25, listando as métricas desse recurso e visualizando, por exemplo, a métrica "memory.usage". Se o passo escolhido for o Passo 8 (visualizar métrica no formato json) o usuário irá configurar datas e hora de início e fim, conforme mostrado na 26. Em seguida pode-se exportar as métricas conforme mostra a Figura 28(b).

(a) Criação do usuário no CloudMetric (b) Login do usuário no CloudMetric

Figura 22 – Criação e login do usuário admin no CloudMetric.

Se o passo escolhido for o Passo 9 (visualizar a métrica no Grafana) o usuário irá clicar no botão visualizar no Grafana, conforme Figura 26, obtendo como resultado a Figura 29(b)

Em comparação com monitoramento sem a utilização do CloudMetric, podemos destacar diversas vantagens do para o usuário, tais como:

- Concentração do monitoramento de diversas nuvens OpenStack em um único arcabouço;
- Com o cadastro efetuado no CloudMetric, a recuperação de todos os recursos monitorados é efetuada de forma automática a partir de uma interface gráfica, sem a necessidade da abertura de um terminal com um *script* específico para cada nuvem OpenStack gerenciada pelo usuário;
- Listagem das métricas criadas pelo OpenStack de maneira bem simples, sem a necessidade de utilização de terminais com acesso SSH;
- Visualização dos valores das medidas das métricas dentro da aplicação, sem o uso de comandos no terminal do OpenStack;
- Configuração forma automática da visualização dos valores da métrica no Grafana;
- Total abstração dos comandos do OpenStackClient para os usuários.

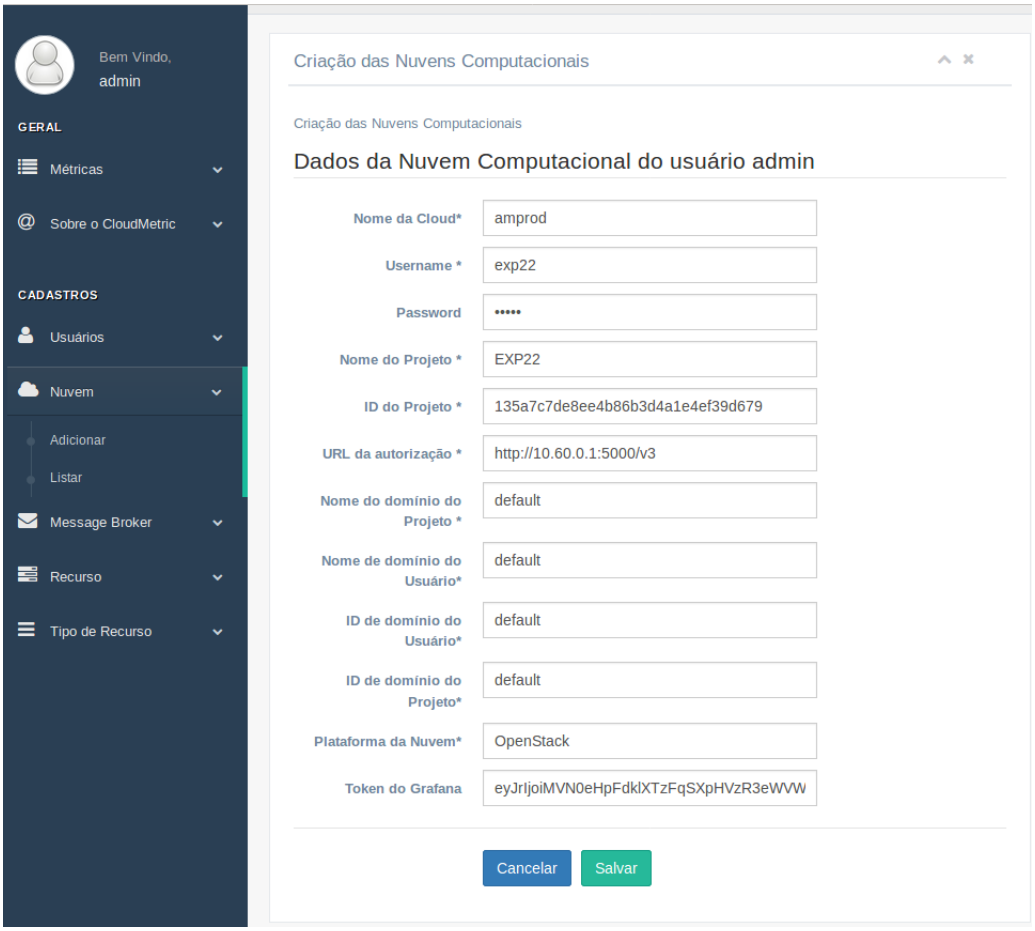


Figura 23 – Cadastro da nuvem computacional no CloudMetric.

6.1.3 Criação de métricas personalizadas sem o uso do CloudMetric

Para cadastrar uma métrica personalizada que monitore o erro de trajetória do robô, sem o uso do CloudMetric, é necessário seguir os passos exibidos no fluxograma da Figura 30. Os 4 primeiros passos do fluxogramas são idênticos aos 4 primeiros passos apresentados no fluxograma da Figura18 na Subseção 6.1.1.

Na Listagem 6.5 foi realizado o cadastro do tipo e nome do recurso da métrica personalizada erro_trajetoria, seguindo os Passos 5 e 6 do fluxograma. O tipo de recurso foi cadastrado com o nome "FUTEBOL", conforme mostrado no comando da Linha 1 da Listagem 6.5 e o nome do recurso foi cadastro como "robo", conforme mostrado no comando da Linha 8 da Listagem.

Listagem 6.5 – Criação do tipo de recurso e do recurso

```
1 stack@controller: ~$ openstack metric resource-type create FUTEBOL
2 +------------+-----------+
3 | Field      | Value     |
4 +------------+-----------+
5 | name       | teste     |
6 | state      | active    |
7 +------------+-----------+
```

Figura 24 – Selecão da nuvem computacional.

Lista de Recursos

ID DO RECURSO	NOME DA MÁQUINA VIRTUAL	NOME DO RECURSO	TYPE	
094acc20-fa5c-445e-9892-77a1f96ff540	CloudMetric	CloudMetric	instance	☑
42f22eba-8cef-4d15-907a-1a8d1e86c1b0	is-rabbitmq	is-rabbitmq	instance	☑
b804364b-5192-473a-bbe5-4a1005ee762c	is-mjpeg-server	is-mjpeg-server	instance	☑
906f5fe8-be88-4b7c-b6e5-5d49ca6af9e6	is-sdn-controller	is-sdn-controller	instance	☑
4dc76942-5386-4099-bc77-fd390b17344e	is-robot-controller	is-robot-controller	instance	☑
63e72e97-7b1e-4afe-9b29-09c08f16a5d6	is-camera-gateway-23	is-camera-gateway-23	instance	☑
a1cbef3b-7f3a-4e7b-90b2-f2eac4530f3a	is-camera-gateway-01	is-camera-gateway-01	instance	☑
2753641b-3aaf-4085-a760-6ec2a0574538	is-image-processing	is-image-processing	instance	☑

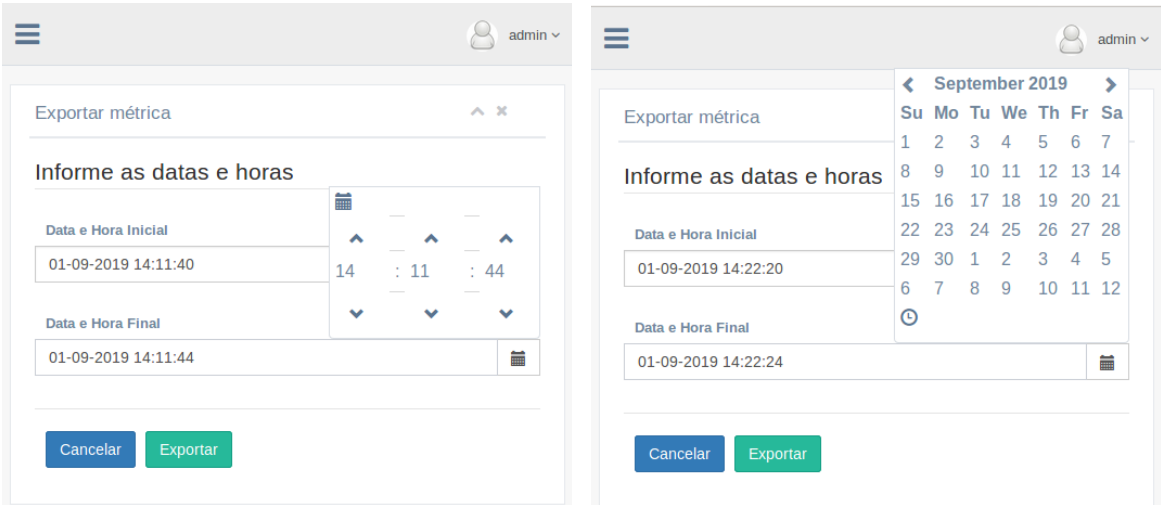
Figura 25 – Lista de recursos da nuvem amprod.

```
8  stack@controller: ~$ openstack metric resource create --type FUTEBOL Robo
9  +-----------------------+--------------------------------------+
10 | id                    | 9cafb16e-5386-48a7-b782-497782283860 |
11 | metrics               |                                      |
12 +-----------------------+--------------------------------------+
```

Após o término dos Passos 6 e 7 do fluxograma é necessário a criação da métrica personalizada no Passo 8 utilizando o cadastro do tipo e nome do recurso. Na Listagem 6.6, na linha 1 temos o comando da criação da métrica personalizada chamada de erro_trajetoria.

ID DA METRICA	NOME DA METRICA	ARQUIVO DE POLÍTICA	GRANULARIDADE	TIPO DE RECURSO	UNIDADE
0b2d5886-1f6a-4f76-969b-bfccf0a280a3	cpu_util	high	1	instance	%
08185063-fcd9-4403-a688-5863aa86ad4c	compute.instance.booting.time	high	1	instance	sec
c99b6747-6d0b-458a-a27e-a6f50d393a76	memory.usage	high	1	instance	MB
20e19903-6ef5-4be4-b4cf-09cd40ac429c	memory.swap.in	high	1	instance	None
6f8ef47d-f3a5-48a5-a140-846860b604fb	memory.bandwidth.local	high	1	instance	None
00f7bbd5-aea9-4508-ba41-e205d42043bb	vcpus	high	1	instance	vcpu
acefeac1-5166-4394-b14f-2906564df254	disk.allocation	high	1	instance	None

Figura 26 – Lista de métricas do recurso is-image-processing.



(a) Exportação da métrica cpu_util (b) Exportação da métrica memory_usage

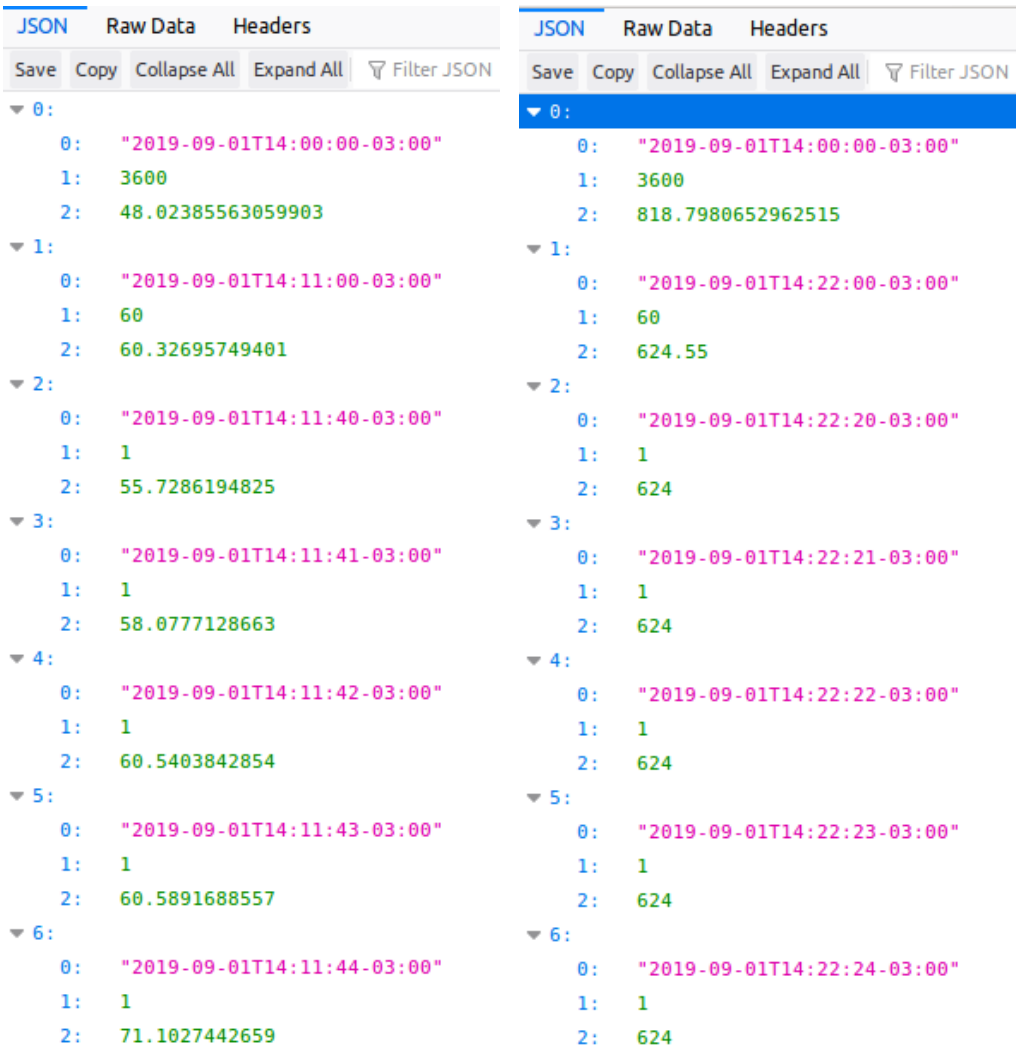
Figura 27 – Configuração das datas inicial e final para exportação no formato json das métricas uso de memória e utilização de CPU do recurso is-image-processing.

Para a criação da métrica foi necessário informar o ID do recurso que a métrica faz parte, o tipo do recurso e associar a ela um arquivo de política de acesso e monitoramento, retornando o ID da métrica.

Listagem 6.6 – Criação da Métrica Personalizada

```
1 stack@controller: ~$ openstack metric create --resource ID 9cafb16e-5386-48a7-
      b782-497782283860 --archive-policy-name HIGH erro_trajetoria
2 +-----------------------+--------------------------------------+
3 | ID                    | dd3bcf37-1bc0-46f9-9c37-97f89f01c227 |
4 | metrics               |                                      |
5 +-----------------------+--------------------------------------+
```

Após obter o ID da métrica personalizada o usuário precisa criar um *script* de envio das medidas do erro de trajetória do robô. O *script* deve consumir as métricas da aplicação no *broker* e enviar para o banco de dados temporal Gnocchi. Na Listagem 6.7 temos o pseudocódigo bem simplificado de envio das medidas capturadas do erro de trajetória do



(a) Lista de medidas da métrica CPU_UTIL (b) Lista de medidas da métrica memory_usage

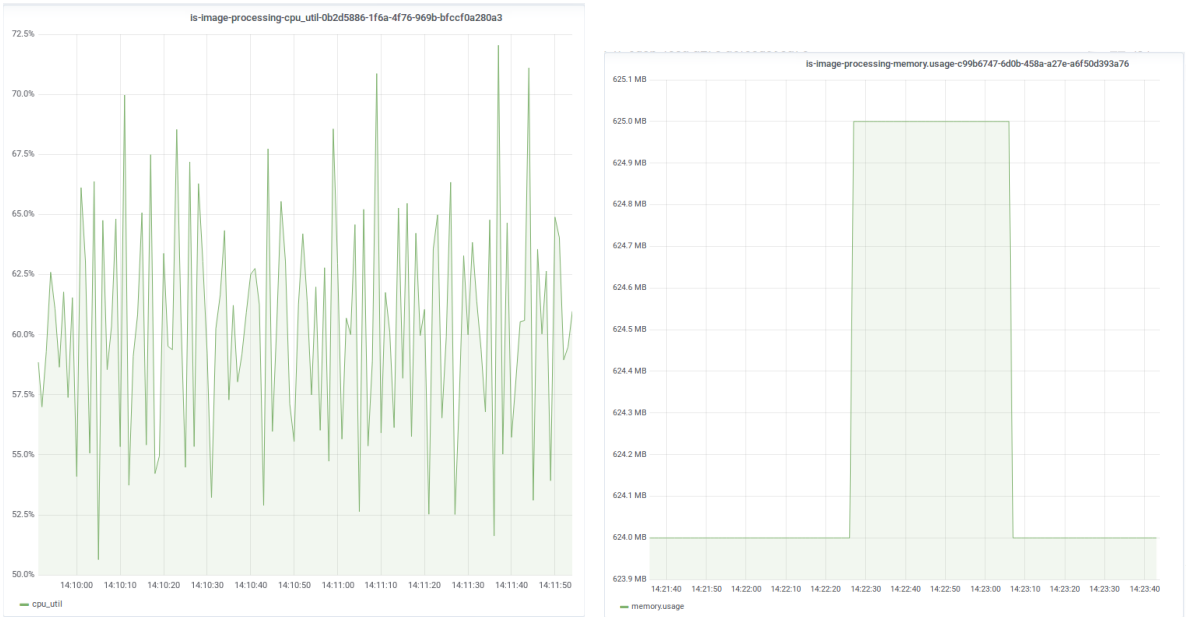
Figura 28 – Lista de medidas no formato json.

robô para o banco de dados temporal Gnocchi.

Na Linha 5 o envio é realizado continuamente até receber uma *flag* de parada do bloco de repetição. A *flag* é representada por uma condição em que altera-se o status da métrica para inativo, como por exemplo o recebimento do valor -1. Dentro do bloco de repetição, temos a leitura do valor de erro de trajetória na aplicação do robô na linha 6 e envio da medida para o banco de dados temporal a cada 1 segundo, passando como parâmetros o nome da métrica, ID do recurso e valor do erro de trajetória consumido.

Listagem 6.7 – Pseudocódigo de envio das medidas da métrica erro de trajetória

```
1 Variáveis erro_trajetoria: Reais;
2
3 Inicio
4     valor_erro_trajetoria = Cosumir erro_trajetoria()
5     Enquanto (erro_trajetoria != -1.0) faça
```



(a) Visualização da métrica CPU_Util (b) Visualização da métrica memory_usage

Figura 29 – Visualização das métricas cpu_util e memory_usage no Grafana.

```
6           adicionar_medida_gnocchi("erro_trajetoria","9cafb16e-5386-48a7-b782
                -497782283860",valor_erro_trajetoria)
7           valor_erro_trajetoria = Cosumir erro_trajetoria()
8       Fim-enquanto
9   Fim
```

Com o envio das medições da métrica erro_trajetoria de forma contínua é possível monitorar a métrica personalizada seguindo os Passos 9 a 11, retornando as medidas no formato json para o usuário ou seguindo os Passos 9 a 12 configurando a métrica no Grafana e visualizando-a em tempo real.

Essa forma nativa do OpenStack de criação de métricas personalizadas torna-se bem trabalhosa com o aumento da quantidade de métricas monitoradas pois, para cada métricas é necessário criar *scripts* personalizados para o envio das medidas para o banco de dados temporal e visualização das medidas.

6.1.4 Criação métricas personalizadas do robô com o uso do CloudMetric

Esta subseção tem o objetivo de mostrar o uso do CloudMetric na criação das seguintes métricas personalizadas (FUTEBOL, 2017):

- Erro de trajetória do robô;
- Velocidade Linear do Robô;
- Relação Sinal/Ruído do ponto de acesso fSTA1.

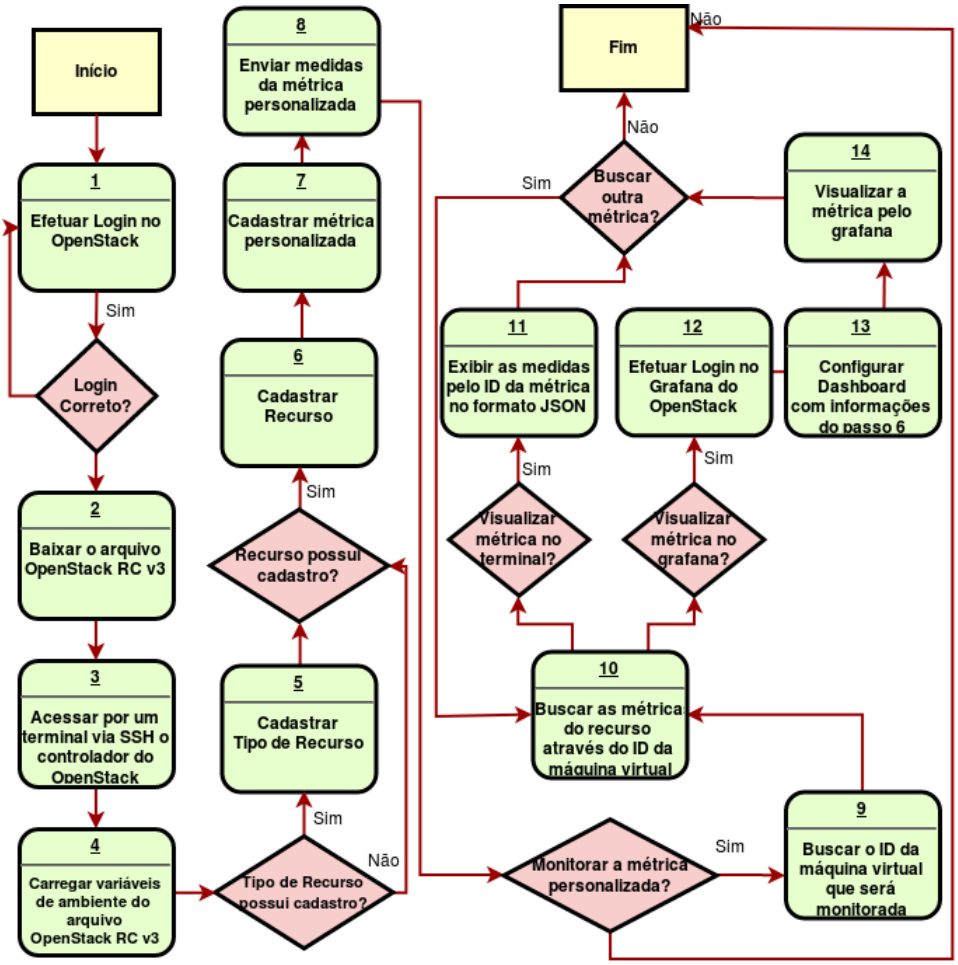


Figura 30 – Fluxograma de criação de métrica personalizada sem o CloudMetric.

As métricas personalizadas Relação Sinal/Ruído do fSTA2, fSTA3, vazão de entrada do fSTA1, fSTA2, fSTA3, fSTA4 e vazão de saída do fSTA1, fSTA2, fSTA3, fSTA4 também foram criadas e monitoradas no CloudMetric, porém seus resultados não serão mostrados aqui dado que o objetivo da seção é apresentar o método de criação, e não as métricas em si.

No fluxograma da Figura 31 todos os passos necessários para criação de métricas personalizadas no CloudMetric.

Como o cadastro do usuário "admin"e a nuvem "amprod"já foi efetuado, as tomadas de decisões do fluxograma até o Passo 3 são todas afirmativas. No Passo 4 temos o cadastro do *broker* alocado na nuvem computacional "amprod". O cadastro do *broker* é realizado com o objetivo de associar as informações do *broker* fornecido pela nuvem, com a nuvem cadastrada pelo usuário.

Acessando o Menu **Message Broker -> Adicionar** exibido na Figura 32 abrirá um formulário para preenchimento das informações do *broker*. Os campos preenchidos são: credenciais de acesso, endereço IP do *broker*, porta, fila e nuvem. O campo *Software Broker* vem preenchido como RabbitMQ.

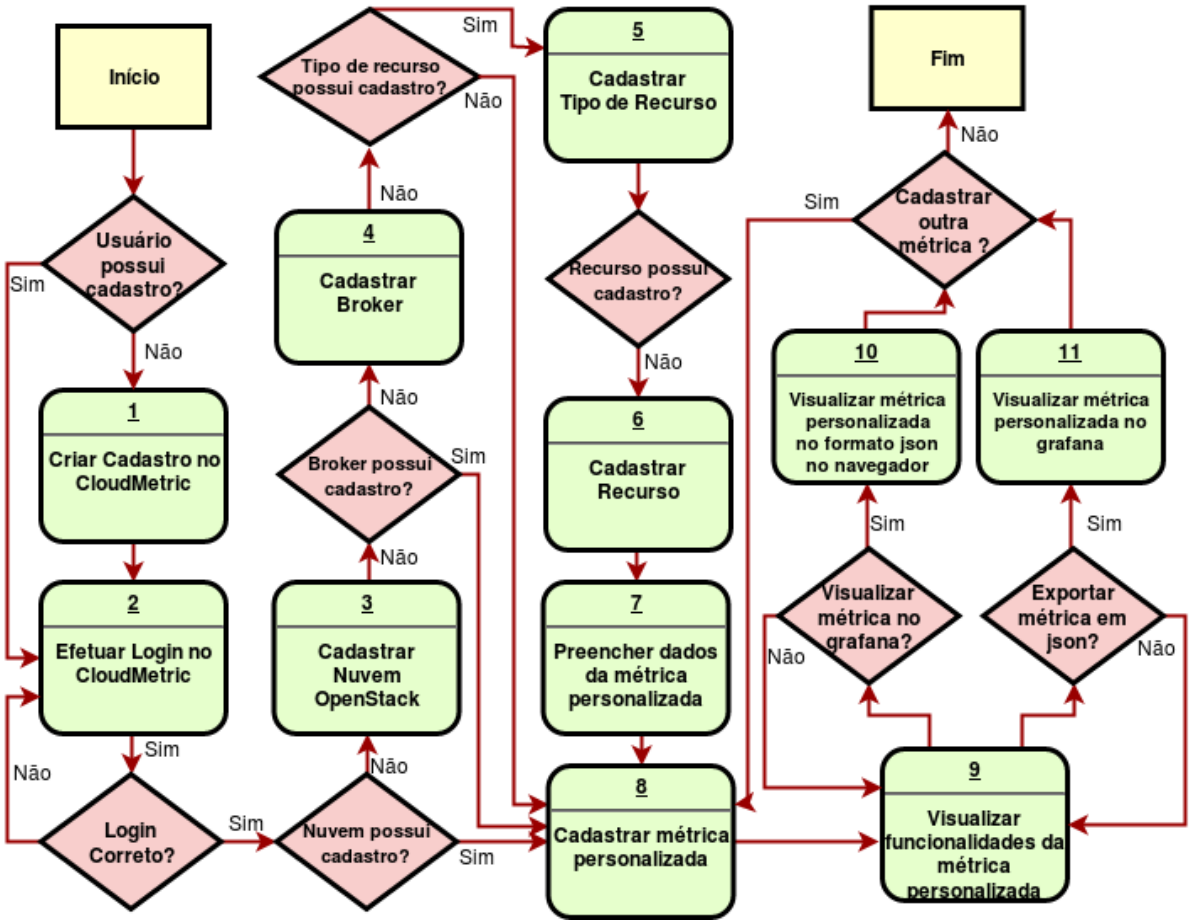


Figura 31 – Fluxograma de criação de métricas personalizadas com o CloudMetric.

Para o cadastro de métricas personalizadas no CloudMetric é necessário que o tipo e o recurso estejam cadastrados previamente no arcabouço. No Passo 5 do fluxograma é realizado o cadastro do tipo de recurso. Na Figura 33 é mostrado uma operação de cadastro do tipo de recurso. Para isso neste exemplo usa-se o menu **Tipo de Recurso -> Adicionar** e cria o tipo "FUTEBOL"pertencente a nuvem computacional "amprod".

De maneira similar foram cadastrados 2 recursos no Passo 6 do fluxograma chamados "Robo"e "wifi". O recurso Robo irá agrupar as métricas personalizadas do robô, enquanto que o recurso wifi agrupará todas as métricas referentes à rede sem fio do espaço inteligente. O tipo de recurso selecionado para os 2 (dois) recursos cadastrados é "FUTEBOL". Após efetuar o cadastro do recurso é gerado o ID da instância que será utilizado na criação da mensagem padronizada para envio das medidas pelo usuário.

Como o término do Passo 6 (cadastrar recurso), temos o preenchimento das informações e a adição da métrica personalizada nos respectivos Passos 7 e 8 do fluxograma. Na Figura 35, acessando o Menu **Metricas -> Adicionar metricas pelo CloudMetric** é aberto um formulário para o preenchimento dos dados da métrica personalizada. Como já observado na Figura 35 foi cadastrada a métrica SNR_fsta1, pertencente ao recurso Wifi e ao *broker* RabbitMQ_CloudMetric_amprod. O caminho de importação de métricas

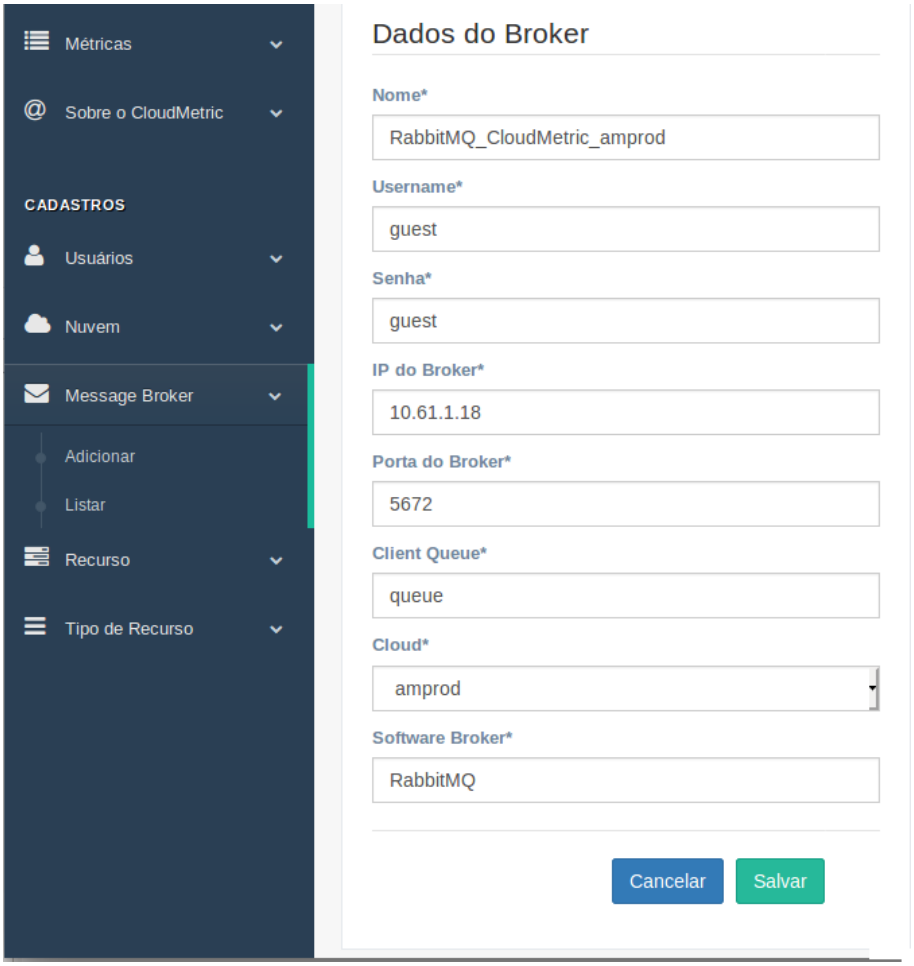


Figura 32 – Cadastro do *Broker*.

foi preenchido com o caminho de rede do servidor Samba que armazena as medidas das métricas enviadas de forma padronizada pelo usuário e o arquivo de política da métrica, escolhida neste exemplo como *high* (granularidade de 1 segundo).

Nas Figuras 36(a) e 36(b) temos respectivamente a criação das métricas personalizadas "velocidade linear"e "erro de trajetória do robô". As duas se diferem da métrica anterior por pertencer ao recurso Robo, e possuírem caminho de importação diferente.

O passo 9 do fluxograma da Figura 31 - visualizar funcionalidades da métrica personalizada - foi representado pela Figura 37. Nessa Figura temos a lista com as 3 (três) métricas personalizadas cadastradas anteriormente. Cada métrica da lista pode ser associada às ações de pausar/ativar, editar e remover, além das funcionalidades de exportar no formato json e visualizar no Grafana. As métricas personalizadas ficam organizadas em uma lista diferente das métricas criadas pelo OpenStack, proporcionando uma melhor organização para o usuário do CloudMetric.

No momento em que o usuário envia as medidas de forma padronizada para o servidor Samba, os agentes do CloudMetric já se encarregam de ler essa medida e armazenar no banco de dados temporal do OpenStack. Essa abordagem simplifica a forma que era

Figura 33 – Cadastro do tipo de recurso.

Figura 34 – Recursos cadastrados no CloudMetric.

enviado de forma nativa o envio dessas medidas. Na Listagem 6.8 temos o exemplo do envio padronizado da métrica erro_trajetoria pelo usuário "admin". De forma análoga para cada métrica personalizada é necessário enviar um dicionário em formato json contendo o resource_id da métrica, nome da métrica e uma lista contendo *timestamp* com horário de envio da medida e valor da aplicação para o servidor de arquivos compartilhado (Samba) configurado no cadastro da métrica personalizada.

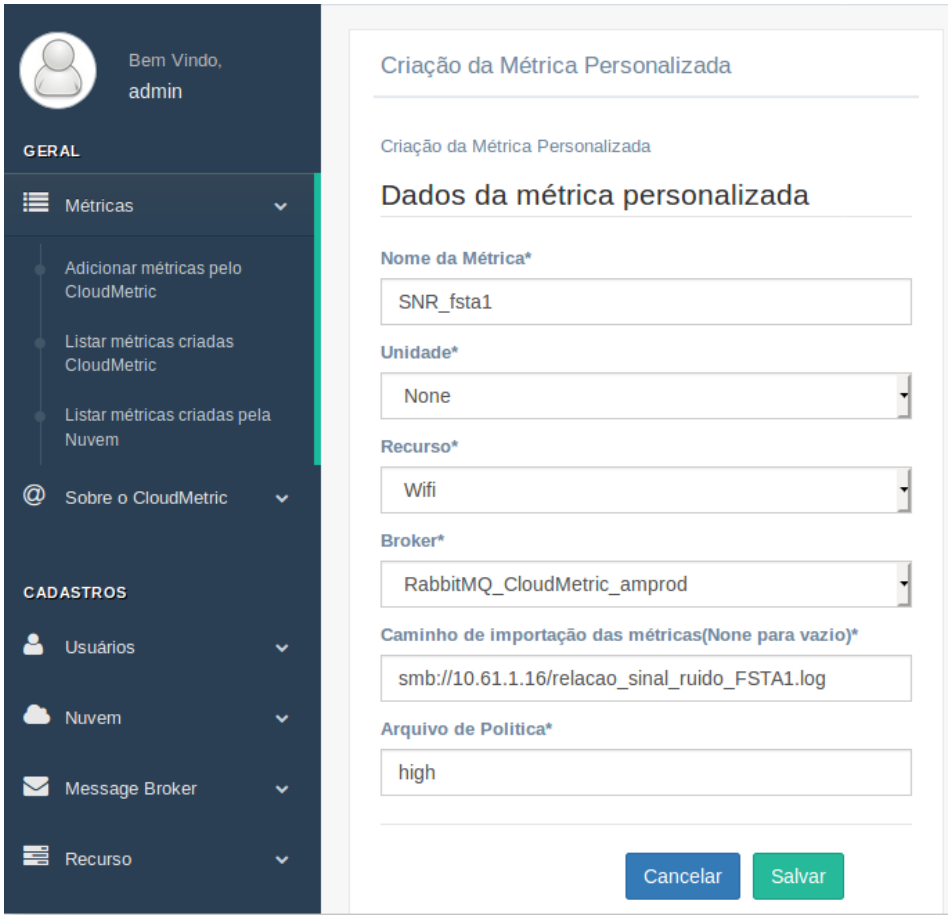


Figura 35 – Criação da Métrica Relação Sinal Ruído do fSTA1.

Listagem 6.8 – Formato de envio padronizado json

```
{
    "9cafb16e-5386-48a7-b782-497782283860": {
        "erro_trajetoria": [
            {
                "timestamp": "2019-09-09T04:26:13.745715",
                "value": 0.32784987315535546
            }
        ]
    }
}
```

As Figuras 38(a), 38(b) e 38(c) apresentam a exportação das medidas de erro de trajetória, velocidade linear e relação sinal ruído no período compreendido entre 07/09/2019 21:23:30 e 07/09/2019 21:23:34.

As Figuras 39(a), 39(b) e 39(c) apresentam a visualização, no Grafana, das medidas de erro de trajetória, velocidade linear e relação sinal ruído.

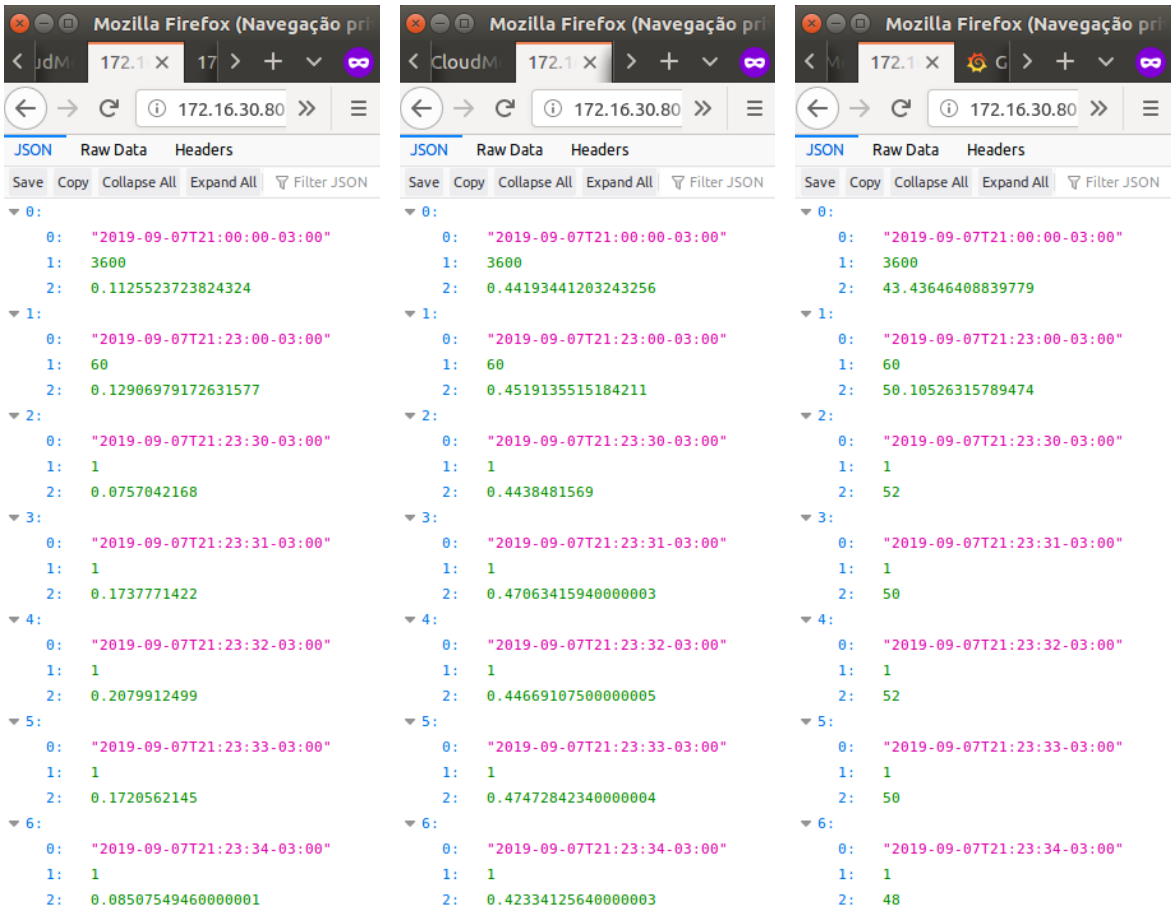
Em comparação com a criação de métricas personalizadas sem a utilização do CloudMetric podemos destacar diversas vantagens para os usuários do arcabouço proposto, tais como:

(a) Criação da métrica velocidade linear (b) Criação da métrica erro de trajetória

Figura 36 – Criação de métricas personalizadas no CloudMetric.

Figura 37 – Lista de métrica personalizadas cadastradas no CloudMetric.

- Concentração em um único arcabouço às métricas personalizadas de diversas nuvens OpenStack;
- Forma de envio das medidas das aplicações de forma padronizada e simples;
- Total abstração dos comandos do OpenStackClient para a criação dos tipos de recursos, nome dos recursos de métricas personalizadas, sendo que todas essas ações são embutidas no arcabouço CloudMetric.



(a) Medidas erro de trajetória (b) Medidas velocidade linear (c) Medidas relação sinal ruído

Figura 38 – Medidas das métricas personalizadas em formato json.

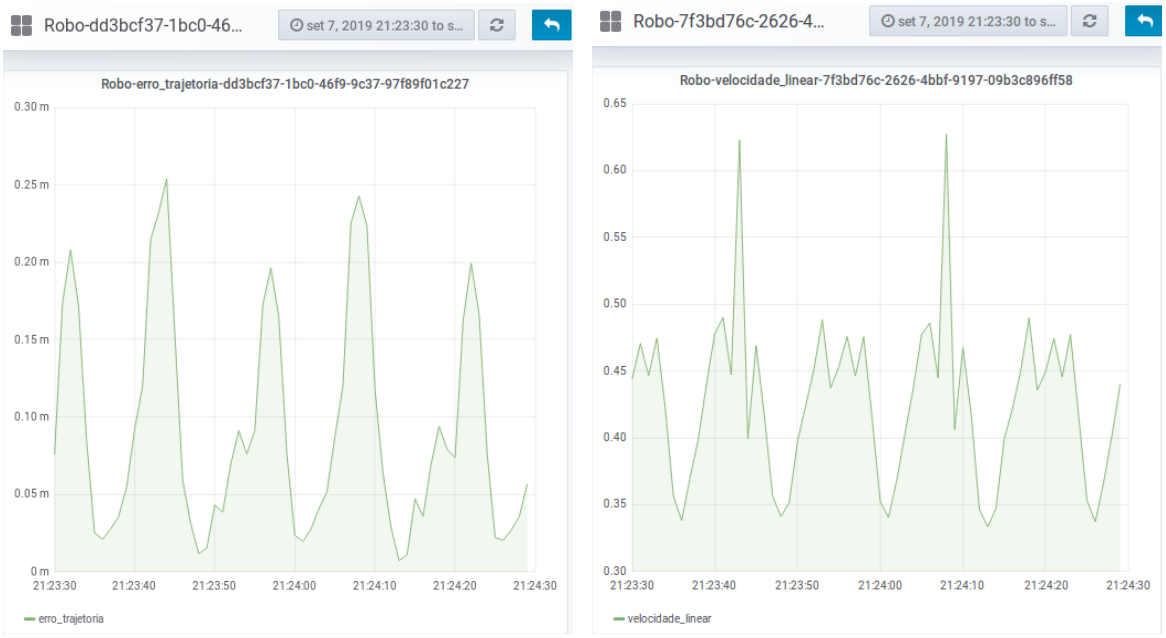
6.2 Descrição do Caso de Teste Monitoramento de um servidor *web*

O monitoramento de um servidor *web* consiste na captura da quantidade de requisições ao longo do tempo enviadas por diversos usuários. Essa captura ocorre através da análise do arquivo "*Log* de Acesso da aplicação apache[3]". O *Log* de acesso registra todas as solicitações processadas pelo servidor. Por exemplo, ao visitar uma página *web*, um registro é gravado e armazena informações como o endereço IP do visitante, página visualizada, tipo de requisição, código de status, a porta e o código do navegador.

O *Log* de acesso fornece diversas informações de controles de aplicações *web* e de tráfego de rede. Essas informações são úteis e devem ser disponibilizada para os administradores de rede, para que seja efetuado o monitoramento constante do servidor.

A infraestrutura de máquinas virtuais para este caso de teste é composta por 1 (um) servidor *web* e 1 (uma) máquina virtual CloudMetric responsável por armazenar os

[3]https://www.apache.org/



(a) Medidas erro de trajetória (b) Medidas velocidade linear

(c) Medidas relação sinal ruído

Figura 39 – Medidas e visualização das métricas personalizadas no Grafana.

arquivos em formato json das métricas personalizadas e hospedar o serviço de troca de mensagens RabbitMQ do arcabouço CloudMetric.

A infraestrutura do espaço inteligente no *datacenter* foi construída através da

plataforma Infraestrutura como serviço (*IaaS*) OpenStack *All-in-One* na versão Rocky [4].

6.2.1 Criação de métricas personalizadas em um servidor *web*

Esta subseção tem o objetivo de mostrar a criação da métrica personalizada quantidade de operações GET em um servidor *web*. Esta informação é extraída através de um *script* que captura o tipo de requisição, neste caso GET, por um tempo determinado.

No fluxograma da Figura 40 temos todos os passos necessários para a criação desta métrica personalizada no CloudMetric.

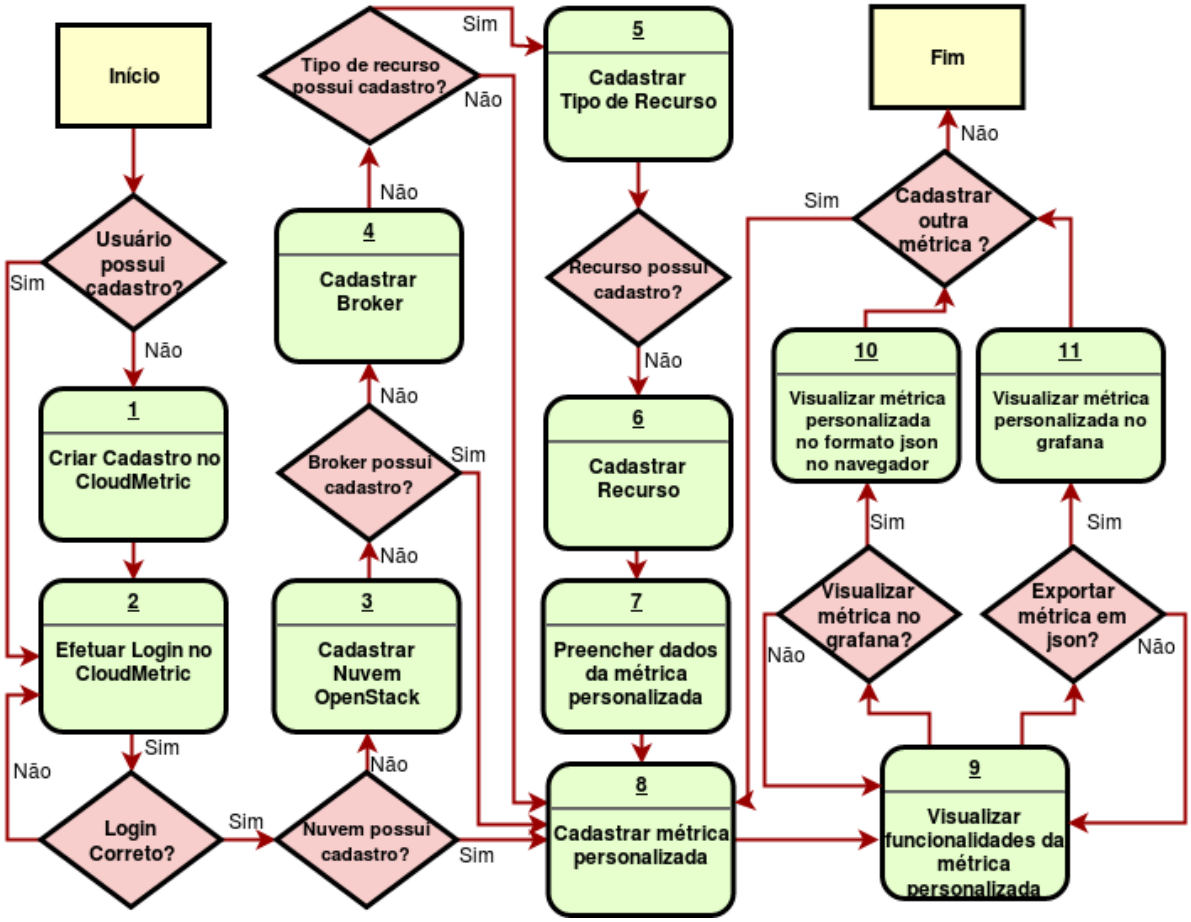


Figura 40 – Fluxograma de criação de métrica personalizada com CloudMetric.

No Passo 1 do fluxograma é realizado o cadastro do usuário "admin", no CloudMetric preenchendo informações básicas de cadastro no formulário como usuário, nome, email e senha. Após efetuado com sucesso o cadastro do usuário "admin"é realizado o *login* no CloudMetric. O Passo 1 é obrigatório somente nos casos que o usuário não tenha sido cadastrado ainda. Pode-se seguir para o Passo 2 caso já se tenha um cadastro ativo no CloudMetric.

Com o usuário autenticado pelo CloudMetric é necessário cadastrar uma Nuvem

[4] https://www.openstack.org/software/rocky/

OpenStack seguindo o Passo 3 do fluxograma, acessando o Menu **Nuvem -> Adicionar** exibido na Figura 41. Os campos preenchidos são obtidos a partir do *Dashboard* do OpenStack, especificamente no arquivo OpenStack RC File v3, com o acréscimo do campo *Token* do Grafana e nome da Cloud criada pelo usuário, como por exemplo OpenStack_WebServer.

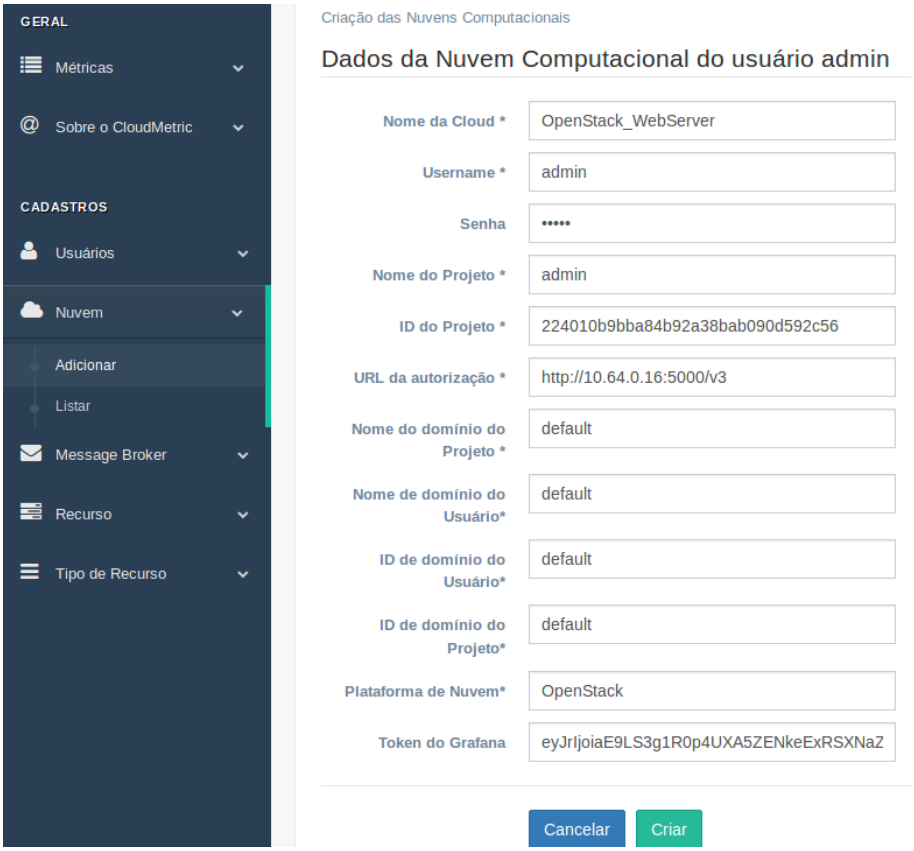


Figura 41 – Cadastro da Nuvem Computacional.

No Passo 4 temos o cadastro do *broker* alocado na nuvem computacional "amprod". Acessando o Menu **Message Broker -> Adicionar** exibido na Figura 42, abrirá um formulário para preenchimento dos campos do *broker*. Os campos preenchidos são as credenciais de acesso ao RabbitMQ, endereço IP do *broker*, porta, fila, nuvem e software de mensageria utilizado. O cadastro do *broker* é realizado com o objetivo de associar as informações do *broker* fornecido pela nuvem com a nuvem cadastrada pelo usuário.

Para o preenchimento do cadastro de métricas personalizadas no CloudMetric é necessário que o tipo e o recurso estejam cadastrados previamente no arcabouço. No Passo 5 do fluxograma é realizado cadastro do tipo de recurso. Na Figura 43(a) é mostrado o cadastro do tipo de Recurso "Web_Server", pertencente a nuvem computacional "OpenStack_Server"e na Figura 43(b) é mostrado a criação do recurso Log_do_Apache, que está associado ao tipo de recurso Web_Server. Após efetuar o cadastro do recurso é gerado o ID da instância que será utilizado na criação da mensagem de envio das métricas pelo usuário.

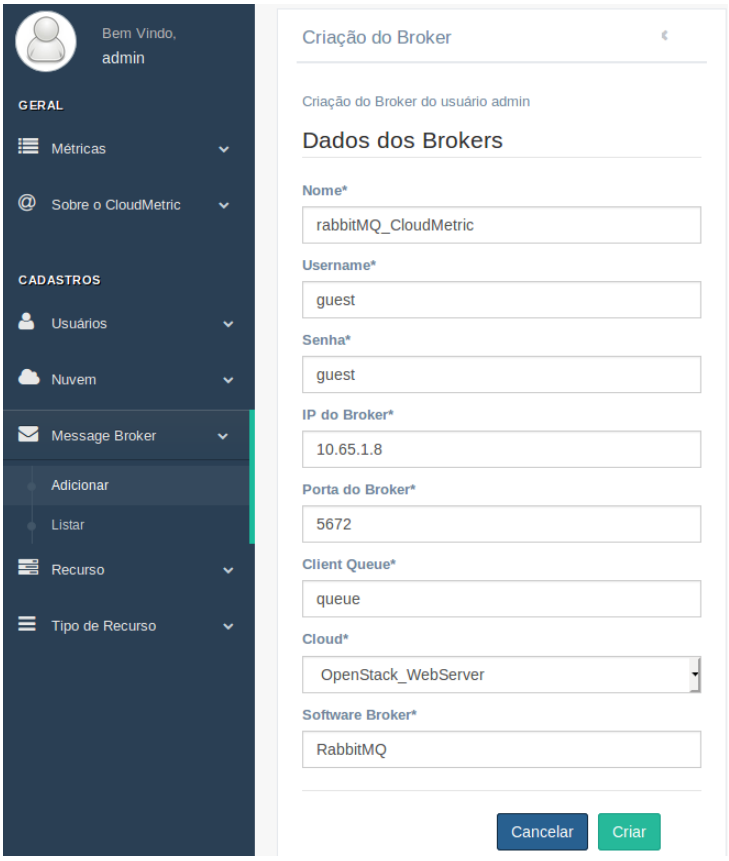
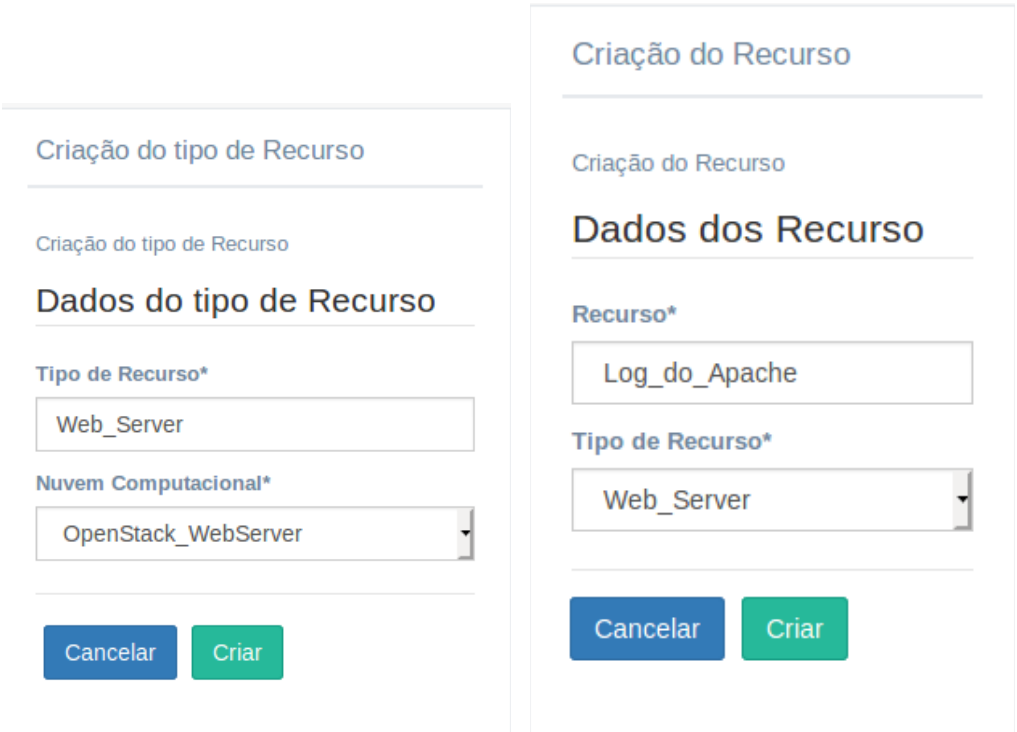


Figura 42 – Cadastro do *Broker*.



(a) Criação do tipo de recurso (b) Criação do recurso

Figura 43 – Criação do recurso e do tipo de recurso.

Nas Figuras 44 temos a criação da métrica personalizada Qtd_Get e a listagem das métricas personalizada. A métrica cadastrada possui como unidade de medidas requisições. Ela está associada ao recurso Log_do_Apache e ao *Broker rabbitMQ_CloudMetric*. Como o caminho de importação das métricas não foi preenchido no cadastro da métrica, o usuário envia de forma direta ao broker, no período de 5 em 5 segundos, a quantidade de requisições GET obtidas no Log de acesso do Apache no Servidor *Web*.

O envio dessa medida se difere do cenário anterior devido ao fato do usuário publicar a medida de forma padronizada de forma direta ao *broker* "rabbitMQ_CloudMetric". Esse envio de forma direta faz com que as medidas sejam processada diretamente pelo agente CloudMetricAgentReadBroker e armazene no banco de dados temporal gnocchi da Iaas OpenStack.

Para cada métrica da lista da Figura 45 podem ser realizadas as ações de pausar/ativar, editar e remover, além das funcionalidades de exportar as medidas no formato json e visualizar no Grafana.

Figura 44 – Criação da Métrica Personalizada.

Nas Figuras 46(a) e 46(b) temos a respectivamente exportação das medidas Qtd_GET no período compreendido entre 08/09/2019 12:41:30 a 08/09/2019 12:41:50 e a visualização das medidas Qtd_GET no Grafana por um período de 3 minutos e 30 segundos.

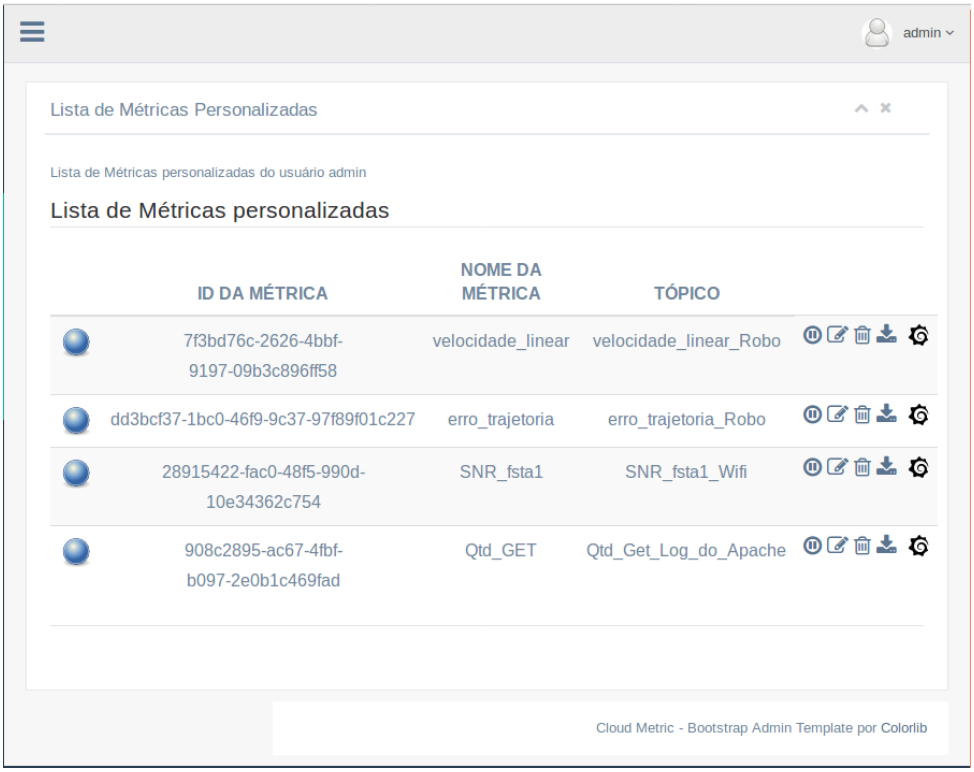
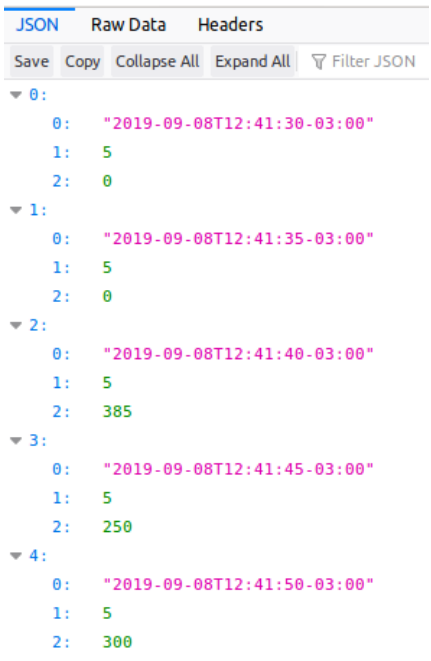
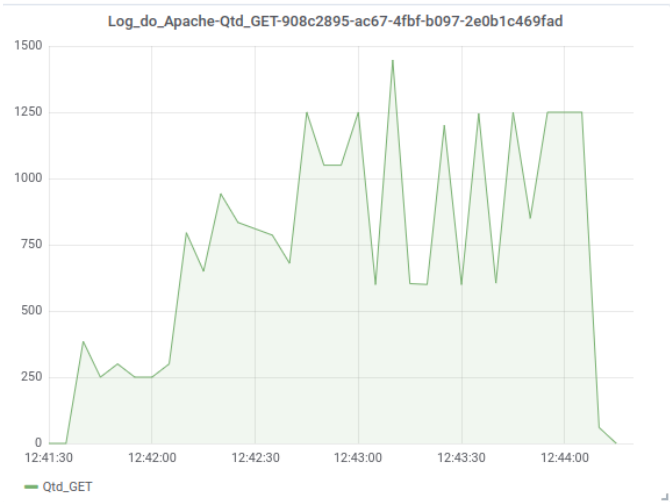


Figura 45 – Lista de métricas personalizadas.



(a) Medidas da métrica Qtd_GET em json



(b) Visualização da métrica Qtd_GET no Grafana

Figura 46 – Medidas e visualização da métrica Qtd_GET.

7 Conclusão

O objetivo dessa dissertação foi demostrar que com uso do arcabouço CloudMetric é possível realizar a criação e monitoramento de métricas personalizadas em nuvens computacionais. Para isso foi apresentado um modelo de arquitetura conceitual para a construção da interface gráfica e dos agentes responsáveis pelo processamento, desde o recebimento da métrica personalizada do usuário, publicação e consumo no tópico do *broker* e envio das medidas das medições para o banco de dados temporal na nuvem. Para validar esse modelo foi desenvolvido um arcabouço com os requisitos necessários para cumprir os objetivos proposto. E por fim apresentado, em 2 (dois) ,Casos de Teste o funcionamento do CloudMetric no OpenStack, demostrando o funcionamento e a viabilidade do modelo conceitual proposto.

Ao final dessa dissertação podemos responder as seguintes perguntas “Como monitorar?”: Através do arcabouço CloudMetric; “O quê monitorar?”: métricas nativas e personalizadas em múltiplas nuvens OpenStack; e “Como integrar métricas de desempenho das aplicações em nuvem?”: a integração é realizada por métricas personalizadas de aplicações criadas pelos usuários do CloudMetric e integradas e gerenciadas pelo arcabouço CloudMetric.

O CloudMetric foi utilizado pelo projeto FUTEBOL (FUTEBOL, 2017) como arcabouço de monitoramento e criação de métricas personalizadas da tarefa de controle de trajetória de um robô em um espaço inteligente. Além disso viabilizou a criação do artigo Monitoramento de Recursos para Aplicações de Robótica em Espaços Inteligentes baseados em uma Nuvem OpenStack (SANTOS et al., 2018); na colaboração do artigo Dispositivos conectados a serviço web em nuvem: complementariedade entre infraestrutura e seletividade para proteção contra DDoS (CORRÊA et al., 2018); e como fornecedor das medições das métricas nativas do OpenStack para a dissertação de mestrado Dimensionamento Vertical Automático de Recursos em Nuvens Computacionais (GIACOMELLI, 2019).

Como trabalhos futuros sugere-se a realização de estudos de viabilidade para integração do arcabouço CloudMetric com outras IaaS de Computação em Nuvem, tais como AWS[1], Azure[2] e GCP[3]. Outros trabalho sugeridos são a agregação das métricas personalizadas para reduzir a complexidade de processamento, a integração do CloudMetric com outros módulos e a adição de novas funcionalidades, como por exemplo o disparo de alarmes associados às métricas personalizadas criadas na nuvem.

[1] https://aws.amazon.com/pt/
[2] https://azure.microsoft.com/pt-br/
[3] https://cloud.google.com/?hl=pt-br

Referências

Aversa, R.; Tasquier, L. Design of an agent based monitoring framework for federated clouds. In: *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. [S.l.: s.n.], 2016. p. 115–120. Citado na página 17.

Aversa, R.; Tasquier, L.; Venticinque, S. Agents based monitoring of heterogeneous cloud infrastructures. In: *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*. [S.l.: s.n.], 2013. p. 527–532. Citado na página 18.

BADGER, L. et al. Cloud Computing Synopsis and Recommendations. In: HALPIN, T.; PROPER, E.; KROGSTIE, J. (Ed.). Publication 800-146. [S.l.]: NIST, 2012. cap. 2. Citado 3 vezes nas páginas 7, 20 e 21.

CHUN, B. et al. Planetlab: An overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 33, n. 3, p. 3–12, jul. 2003. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/956993.956995>. Citado na página 17.

CORRÊA, J. H. et al. Dispositivos conectados a serviço web em nuvem: complementariedade entre infraestrutura e seletividade para proteção contra ddos. In: SBC. *Anais do Workshop de Segurança Cibernética em Dispositivos Conectados (WSCDC-SBRC 2018)*. [S.l.], 2018. v. 1. Citado na página 74.

DIZDAREVI, J. et al. A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 51, n. 6, p. 116:1–116:29, jan. 2019. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/3292674>. Citado na página 28.

FANIYI, F.; BAHSOON, R. A systematic review of service level management in the cloud. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 48, n. 3, p. 43:1–43:27, dez. 2015. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2843890>. Citado na página 14.

FOUNDATION, O. *Openstack*. 2017. Disponível em: <https://www.openstack.org/user-stories/>. Acesso em: 24 de Julho de 2019. Citado na página 22.

FOUNDATION, O. *High-Level Architecture*. 2019. Disponível em: <https://docs.openstack.org/ceilometer/queens/contributor/architecture.html>. Acesso em: 12 de Setembro de 2019. Citado 3 vezes nas páginas 7, 25 e 26.

FOUNDATION, O. *Install and configure Ceilometer for Ubuntu*. 2019. Disponível em: <https://docs.openstack.org/ceilometer/queens/install/install-base-ubuntu.html.> Citado na página 15.

FOUNDATION, O. *Openstack*. 2019. Disponível em: <https://www.openstack.org/>. Acesso em: 24 de Julho de 2019. Citado 2 vezes nas páginas 14 e 22.

FOUNDATION, O. *OpenStack Dashboard User Documentation*. 2019. Disponível em:

<https://docs.openstack.org/horizon/latest/user/index.html.> Citado na página 15.

FUTEBOL. *Specification of FUTEBOL showcases*. 2017. Disponível em: <http://www.ict-futebol.org.br/wp-content/uploads/2017/07/Deliverable2.2.pdf>. Acesso em: 15 de Agosto de 2019. Citado 3 vezes nas páginas 45, 60 e 74.

GIACOMELLI, D. C. *Dimensionamento Vertical Automático de Recursos em Nuvens Computacionais*. Dissertação (Mestrado) — Universidade Federal do Espírito Santo, 2019. Citado 2 vezes nas páginas 46 e 74.

GNOCCHI. *Architecture overview*. 2019. Disponível em: <https://gnocchi.xyz/intro.html.> Citado 3 vezes nas páginas 7, 26 e 27.

GRAFANALABS. *Grafana testimonials*. 2019. Disponível em: <https://grafana.com/grafana/testimonials>. Acesso em: 07 de Agosto de 2019. Citado na página 31.

HVIZDOš, J. et al. Applications of remote controlled robotics in the intelligent space. In: *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. [S.l.: s.n.], 2017. p. 117–122. Citado na página 45.

JOHANSSON, L. The Optimal RabbitMQ guide from beginner to advanced. In: VINKA, D. M. E. (Ed.). First edition. [S.l.]: CloudAMQP, 2018. cap. 1. Citado 3 vezes nas páginas 7, 29 e 30.

LI, A. et al. Cloudcmp: Comparing public cloud providers. In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. New York, NY, USA: ACM, 2010. (IMC '10), p. 1–14. ISBN 978-1-4503-0483-2. Disponível em: <http://doi.acm.org/10.1145/1879141.1879143>. Citado na página 17.

OPENSTACK. *Create OpenStack client environment scripts*. 2019. Disponível em: <https://docs.openstack.org/mitaka/install-guide-obs/keystone-openrc.html>. Acesso em: 20 de Agosto de 2019. Citado na página 25.

OPENSTACK. *Definições*. 2019. Disponível em: <https://docs.openstack.org/keystone/queens/admin/cli-manage-projects-users-and-roles.html>. Acesso em: 18 de Agosto de 2019. Citado na página 24.

OPENSTACK. *OpenStack API Referências*. 2019. Disponível em: <https://docs.openstack.org/queens/api/>. Acesso em: 18 de Agosto de 2019. Citado na página 24.

OPENSTACK. *Serviços do OpenStack*. 2019. Disponível em: <https://www.openstack.org/software>. Acesso em: 18 de Agosto de 2019. Citado 3 vezes nas páginas 7, 23 e 24.

OPENSTACK. *Serviços do OpenStack*. 2019. Disponível em: <https://www.openstack.org/software/project-navigator/openstack-components#openstack-services>. Acesso em: 18 de Agosto de 2019. Citado 2 vezes nas páginas 14 e 22.

OPENSTACK. *Visão Geral*. 2019. Disponível em: <https://docs.openstack.org/install-guide/overview.html>. Acesso em: 18 de Agosto de 2019. Citado 2 vezes nas páginas 7 e 25.

Rabinovich, M. et al. Dipzoom: The internet measurements marketplace. In:

Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications. [S.l.: s.n.], 2006. p. 1–6. Citado na página 17.

Rossigneux, F. et al. A generic and extensible framework for monitoring energy consumption of openstack clouds. In: *2014 IEEE Fourth International Conference on Big Data and Cloud Computing*. [S.l.: s.n.], 2014. p. 696–702. Citado na página 18.

SANTOS, P. B. et al. Monitoramento de recursos para aplicações de rob ótica em espaços inteligentes baseados em uma nuvem openstack. In: SBC. *Anais do XVI Workshop em Clouds e Aplicações (WCGA-SBRC 2018)*. [S.l.], 2018. v. 16. Citado na página 74.

Syed, H. J. et al. Cloudprocmon: A non-intrusive cloud monitoring framework. *IEEE Access*, v. 6, p. 44591–44606, 2018. Citado na página 18.

Young-min Kim et al. Architecture of a network performance monitor for application services on multi-clouds. In: *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*. [S.l.: s.n.], 2013. p. 594–599. Citado na página 17.