Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Ana Luiza de Avelar Cabral

MONOGRAFIA DE PROJETO ORIENTADO EM COMPUTAÇÃO II

CONSTRUÇÃO DE MONITORAÇÃO PARA AMBIENTE EM NUVEM DO LABORATÓRIO SPEED

Belo Horizonte

2019 / 1º semestre
Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

CONSTRUÇÃO DE MONITORAÇÃO PARA AMBIENTE EM NUVEM DO LABORATÓRIO SPEED

por

Ana Luiza de Avelar Cabral

Monografia de Projeto Orientado em Computação II

Apresentado como requisito da disciplina de Projeto Orientado em Computação II do Curso de Bacharelado em Ciência da Computação da UFMG.

Prof. Dorgival Olavo Guedes Neto
Orientador

Belo Horizonte
2019 / 1º semestre

Dedico este trabalho à minha avó e parceira, que queria comemorar a formatura comigo e não pode, as conquistas serão sempre nossas.

AGRADECIMENTOS

Queria agradecer muito ao professor Dorgival por aceitar me orientar e sugerir possibilidades para Projetos a serem implementados, pela visão que me passou da área, por todas as possibilidades que me ofereceu, pelo cuidado e atenção. Sou muito grata pelas oportunidades de aprendizado e crescimento.

Agradecer também ao Vinícius infinitamente pelas orientações no POC1, pela base sólida de sua dissertação que me orientou no meu projeto, pela parceria, pelas conversas divertidas.

Ao Guilherme não tenho nem como agradecer. Pela paciência, pela dedicação de me mostrar várias formas de melhorar meu trabalho e como ser melhor profissional. Pelo cuidado, parceria, por me dar tão boas orientações e ter sido em vários momentos ainda mais que uma forte visão técnica, um amparo calmo pros momentos de desespero.

Ao Nestor pelo suporte, parceria e conversas divertidas. Aos amigos do LSL pelos momentos divertidos. À minha família e amigos pela paciência com as ausências.

Se eu vi mais longe foi por estar sobre ombros de gigantes.

Isaac Newton

RESUMO

Pesquisas e trabalhos envolvendo o processamento de *Big Data* exigem infraestrutura projetada para atender às demandas específicas deste cenário. Esta infraestrutura precisa viabilizar a instanciação e destruição constante de Máquinas Virtuais (VMs) para que seja feito o processamento distribuído intrínseco à atividade. O fornecimento destes recursos computacionais (VMs) é feito com a construção de um ambiente em nuvem. Os servidores do data center proveem recursos físicos para que seja construída sobre eles uma nuvem; que pode ser vista como um *pool* de recursos a partir do qual as VMs são providas, modelo de fornecimento de serviços em nuvem conhecido como *Infrastructure as a Service (IaaS).* Uma vez provido este serviço, a monitoração deste precisa ser feita para garantir a qualidade do serviço sendo entregue. Um sistema de monitoração possibilita acompanhar o comportamento do ambiente sendo monitorado, planejar capacidade e recursos, atender à demandas e identificar gargalos e quedas de desempenho. Este trabalho implementa um sistema de monitoração para o ambiente em nuvem como o descrito aqui, de forma a garantir a qualidade da *IaaS* sendo fornecida neste contexto. A proposta também contempla que o sistema de monitoração seja construído usando todos os componentes Software Livre.

Palavras-chave: Ambientes em Nuvem; Monitoração; Data Center; Quality of Service (QoS); Software Livre.

Lista de Figuras

Lista de Siglas

ELK Elasticsearch, Logstash e Kibana, page 15

IaaS Infrastructure as a Service, page v

POC Projeto Orientado em Computação, page iii

UFMG Universidade Federal de Minas Gerais, page i

VM Máquina Virtual, page v

Sumário

1 Notas Iniciais e acesso à ferramenta em funcionamento

Como o projeto desenvolvido é do tipo tecnológico, onde são avaliados 10 pontos para a ferramenta e 10 para o relatório técnico, disponibilizo aqui o acesso para a ferramenta.

Estes links de acesso são temporários, precisam expirar após as avaliações finais do projeto pela segurança do data center. (O sistema de monitoração será mantido, só o acesso à ele pelos links aqui que precisa expirar após o resultado das avaliações).

Para acessar, por segurança, é necessário estar acessando de um IP na UFMG ou conectado à VPN do DCC (passo-a-passo para usar a VPN do DCC no site do CRC-DCC, aqui). Uma vez conectado à VPN ou no campus, é possível acessar:

- Para um *overview* do monitoramento, acesse aqui:

  Mostra os hosts que estão sendo monitorados pelo sistema e as coletas das métricas de consumo em cada um (as métricas ficam visíveis ao clicar em um host).

- Para ver os dashboards de análise de recursos, acesse aqui:

  Mostra para todos os hosts monitorados, com tempo up, consumo de CPU com tempo, tráfego de rede, entre outras análises diversas. A diagramação e construção dos dashboards foi construída de modo a facilitar a análise pelos usuários finais do sistema de monitoração: os administradores do data center e do ambiente em nuvem. Assim, requisitos foram levantados com estes e a diagramação e exibição foi implementada de acordo com o solicitado.

  - Para ver todas as análises, expanda as abas relativas à cada análise (Abas CPU, Load, Network...)
  - Com a caixa de seleção *host* no canto superior esquerdo da página, é possível selecionar o/os host/hosts para os quais você quer ver as análises. Um detalhe é que quanto mais hosts selecionados mais demora para plotar dinamicamente os gráficos, justamente por serem plotados sempre após a seleção dos hosts. Mas o tempo de espera nunca será maior do que 30 segundos para um gráfico, mesmo com mais de 40 hosts a plotar dinamicamente.

- Para ver os logs gerados nos hosts monitorados, acesse aqui:

  Aqui é possível ver todos os logs gerados nos hosts e pesquisar sobre eles.

Como combinado, este documento visa ser relatório final de curso e também documentação do projeto executado, documentando decisões de projeto e permitindo a manutenibilidade por outros administradores. Desta forma, ele tem os itens do template de um relatório de final de curso mas também inclui detalhes de implementação e manutenibilidade em algumas partes do capítulo de desenvolvimento do trabalho.

2 Introdução

2.1 Contextualização e motivação

O Laboratório Speed-DCC-UFMG desenvolve pesquisas envolvendo processamento de *Big Data*, processamento intrinsecamente distribuído devido ao volume de dados sendo trabalhados e formas de processá-lo. Para isto, o laboratório de pesquisa conta com infraestrutura dedicada (Data Center) para atender às suas demandas. A partir dos recursos dedicados foi construída uma nuvem privada usando *OpenStack* para gerenciar o *pool* de recursos e fornecer as Máquinas Virtuais necessárias para o processamento dos dados. Este serviço é conhecido como *IaaS - Infrastructure as a Service*. [MEL 11]

Uma vez o serviço sendo fornecido, é essencial monitorar o Data Center e o ambiente em nuvem, acompanhando o que está acontecendo em tempo real neste ambiente. A monitoração do ambiente em nuvem é a forma de aferir a qualidade do serviço oferecido, identificar gargalos e problemas ocorrendo, planejar e adequar o provisionamento de capacidade e recursos à demanda. [ACE 13]

Quando este projeto foi proposto, estava consolidado no Laboratório um sólido ambiente em nuvem fornecido com uso do *Openstack* usando os recursos do Data Center. À época era necessário um sistema de monitoração para o acompanhamento dos serviços fornecidos. Assim, o Projeto Orientado é justamente a construção de um sistema de monitoração para este ambiente.

2.2 O processo de desenvolvimento

2.2.1 Ambiente de produção

O ambiente de produção do projeto vale ser mencionado aqui porque influi em decisões de projeto e na organização do repositório.

O provisionamento e gerência da infraestrutura no laboratório seguem a tendência atual de serem feitos como código (*Infrastructure as Code*). Para dar suporte ao desenvolvimento e gerência deste código, o Laboratório possui um serviço de gestão de desenvolvimento, GitLab, contendo repositório de código com controle de versionamento.

O *GitLab* é usado por toda a equipe de infraestrutura para administrar a infraestrutura e recursos. Assim, como o desenvolvimento nele é em equipe, submeter ou ajustar códigos nele precisa ser feito seguindo padrões, de modo a ficar compreensível para os outros membros e compreensível e manutenível por outros administradores que trabalharão no futuro no laboratório. Os padrões seguidos pela equipe são as mesmas boas práticas de desenvolvimento em repositórios de construção Software Livre. Entre elas:

- Divisão de ambiente de produção e ambiente de desenvolvimento.

Na *branch* principal do repositório, normalmente a **branch** master, fica o código que está sendo usado (código do ambiente de produção). Em outras *branches* ficam *features* sendo atualmente desenvolvidas pela equipe (códigos aplicados somente em ambiente de desenvolvimento).

- Inclusão de novas features somente por *Pull Request*.

  No há edição direta do código que está em produção: toda inclusão de funcionalidade ou construção é feita por

  1. Mescla os *commits* em *commit* único por funcionalidade implementada, e **submete pedido** pra merjar o código na *branch* de produção.
  2. Os membros **administradores do repositório efetuam** o *merge*: Incluem o novo código no ambiente em produção e excluem a *branch* que foi usada para o desenvolvimento.

2.2.2 Forma de Desenvolvimento, documentações e decisões norteadoras

2.2.2.1 Estrutura e Dinâmica do repositório

O sistema foi desenvolvido em um repositório no GitLab exlusivo para ele. O repositório *monitoring-service* contém o código para o servidor e na pasta *monitored-hosts* nele contém o código para o cliente com os agentes da camada de coleta.

Os administradores da infraestrutura do laboratório e do *GitLab* deixaram nele scripts de automação para que ao incluir novo código / feature no repositório, esse código produzido já seja enviado direto para o servidor de monitoração, já seja colocado em produção.

Além disso, são os administradores da infraestrutura do laboratório que autorizam no repositório o merge de novo código para o ambiente de produção. Isto é feito para a segurança do ambiente em nuvem e das rotinas do laboratório. Desta forma, a cada nova parte do sistema de monitoração desenvolvido um pedido de inclusão é feito, e este é autorizado pelos admins. No fim do projeto eu mesma já estava autorizando meus pedidos após análises de risco com os administradores, além de desenvolver direto na branch em produção devido ao tempo.

2.2.2.2 Documentação do código desenvolvido

O Desenvolvimento foi feito de modo a deixar o sistema de monitoração o máximo possível compreensível por outros. Isto porque ele permanecerá no laboratório e sua manutenção será feita por outras pessoas que não o desenvolveram. Também porque pretendo disponibilizá-lo no GitHub para uso como software livre, então ele vai ser usado e personalizado por outros. Assim, para facilitar sua compreensão, diversas estratégias foram adotadas.

A primeira estratégia é deixar o repositório claro e simples, e incluindo READMEs com pontos importantes. A estrutura do repositório está de forma padrão. Os nomes das pastas

e arquivos, além da organização lógica destes, foram feitos de forma a deixar o repositório o mais intuitivo possível. Arquivos duplicados e temporários foram removidos. Comentários essenciais foram feitos para explicar trechos principais. Além disso há o README principal do repositório com as instruções de personalização e configuração. Em locais onde foi necessário foi inserido um README relativo àquele contexto. Um exemplo deste tipo de caso é na pasta *monitored-hosts*, pasta com o código para os clientes de coleta (que pode ser colocado ou direto em cada host ou pode ser transplantado para automatizadores de infraestrutura como puppet, ansible, etc.). Instruções específicas dos códigos dos hosts estão no README desta pasta. Além disso, os commits feitos ao longo do desenvolvimento foram feitos de modo a incluir uma funcionalidade unicamente por commit e com título claro do que está sendo alterado, sendo possível acompanhar a inclusão e o código relativo à cada parte da monitoração.

Outra estratégia de documentação é a documentação de decisões de projeto. Ao longo da construção do sistema tudo foi sendo documentado e escrito em documentos em uma pasta no *Google Drive*. Agora todo este conteúdo está sendo transplantado para um relatório de decisões tomadas e porquê. Este relatório ainda não está pronto porque é muito conteúdo a ser transplantado. Mas está sendo feito e vai ser entregue após este. Após o fim do Projeto Orientado ainda continuarei no Laboratório e no Projeto por mais 2 meses, neste tempo finalizarei o transplante das decisões de projeto.

A terceira forma de documentar alguns pontos é este documento.

2.2.2.3 Ciclo de Desenvolvimento, prazos e escolhas de features por versão lançada

Ao longo da construção do sistema de monitoração várias possibilidades de implementação de features *extras* foram aparecendo, desde mais camadas ao sistema de monitoração à pequenos detalhes de implementação (por exemplo: definir template personalizado para formatação das notificações por e-mail de queda / indisponibilidade de um host).

Ao começar o POC defini um cronograma para execução do projeto. Como em todo projeto, o cronograma de execução sofreu alguns ajustes por estimativas incertas (etapas que ao fim levaram menos ou mais tempo do que o previsto). Além disso, para tentar incluir as *features* possíveis comecei a tentar ajustar o cronograma para cabê-las. Após um certo momento percebi que era impossível incluir tudo e obedecer ao prazo de entrega ao mesmo tempo.

A partir deste momento comecei a separar as funções possíveis para implementação de acordo com a prioridade de implementação. As de maior prioridade, essenciais para o funcionamento da monitoração, foram incluídas na primeira entrega do produto e no prazo inicial. *Features* não indispensáveis foram documentadas para implementação nas próximas versões do sistema de monitoração.

Expansão futura do sistema de monitoração

Dentro do GitLab, onde o sistema é desenvolvido, há em cada projeto uma seção de *Milestones* e *Issues*. Milestones são etapas do desenvolvimento, como por exemplo: clusterizar, incluir coleta de métricas dos hosts, incluir camada de armazenamento de métricas. Cada Milestone / Etapa inclui diversas tarefas, as issues.

Para documentar todas as features extras possíveis, criei uma Milestone (Etapa) chamada ”Pós-POC”incluindo várias issues (tarefas) extras, onde cada tarefa é uma funcionalidade possível de ser incluída em próximas versões do sistema de monitoração. Em cada funcionalidade eu coloquei onde buscar informações sobre isso, contexto, arquivos.

2.3 O produto e trabalhos relacionados

Monitorar ambientes em nuvem envolve monitorar diversos aspectos do funcionamento de ambientes em nuvem. Há no mercado uma vasta gama de ferramentas para tal, onde cada ferramenta cobre um trecho a ser monitorado. Para montar um sistema completo de monitoração, é necessário

- Escolher as partes que serão monitoradas;
- Uma vez escolhido o que vai ser monitorado, para cada parte monitorada é necessário escolher uma entre as diversas ferramenta possíveis para ser usada para implementar esta funcionalidade;
- Montar uma arquitetura de comunicação e lugar para as funcionalidades do sistema de forma que atenda o máximo possível entre dos desejados abaixo:

    1. **Pouco invasivo (Adaptabilidade)**: Um sistema de monitoração precisa ser menos invasivo possível, interferindo o mínimo possível no funcionamento da nuvem e consumindo o mínimo possível de recursos dela.
    2. **Maior número de funcionalidades**: O desejável com um sistema de monitoração é que ele consiga monitorar o funcionamento da nuvem na maior parte do aspectos que a compõe (comportamento de daemons, consumo de recursos físicos, tráfego de rede...).
    3. **Pontualidade**: É desejável que a monitoração seja em tempo real ou o mais próximo possível disso, uma monitoração com grande atraso não faz sentido.
    4. **Elasticidade**: É necessário que o sistema permita com facilidade mudanças nas entidades monitoradas e na quantidade delas, sem que estas mudanças, que são comuns, obriguem a reformulação da arquitetura.
    5. **Extensibilidade**: É desejável que o sistema permita a adição de novas funcionalidades que mais pra frente venham a ser necessárias sem precisar se reformular inteiro.

6. **Escalabilidade**: É desejável que o sistema não seja rígido e comporte o crescimento, caso necessário, também das entidades monitoradas.

A proposta deste trabalho é montar uma monitoração completa com componentes Software Livre.

Há no mercado ferramentas para monitoração que cobrem de forma muito completa todas as partes que se deseja monitorar, e com grande adesão do mercado, porém sem ser software livre. Um exemplo destas ferramentas é o *Splunk*.

Dentro da proposta de Software Livre, ainda não há ferramentas completas que atendem à tantas demandas e nem com grande adesão como as de Software proprietário. Ao longo dos anos é possível ver que cada ferramenta que monitora cada trecho tem ou crescido em funcionalidades ou facilitado a conexão com outras de modo a fornecer mais funcionalidades.

Um exemplo desta tendência que podemos citar é o banco de dados temporal *InfluxDB* muito usado pela comunidade para se armazenar valores numéricos coletados dos hosts. Há três anos ele era oferecido independente, agora, em sua última versão, está sendo oferecido como *InfluxData*, que traz integração com as novas ferramentas *Telegraf* de coleta de dados, *Chronograf* de visualização destes dados e *Kapacitor* para processamento e correlações destes dados.

Também nos últimos três anos houve a expansão da pilha ELK (Elasticsearch, Logstash e Kibana). As três ferramentas sempre tiveram muita adesão na comunidade para tratamento de dados textuais de monitoração (como logs), onde *Logstash* faz o parse e tratamento dos dados recebidos, *ElasticSearch* é banco de dados para armazená-los e motor de busca para estes, e *Kibana* fornece interface gráfica para interação com o *ElasticSearch*. Recentemente, *Kibana* começou a oferecer a geração de Dashboards estatísticos com os dados. Além disso, está sendo oferecido agora como parte da pilha a ferramenta *Beats*, agente leve para coleta de dados em um host, e também soluções integradas que usam a pilha ELK:

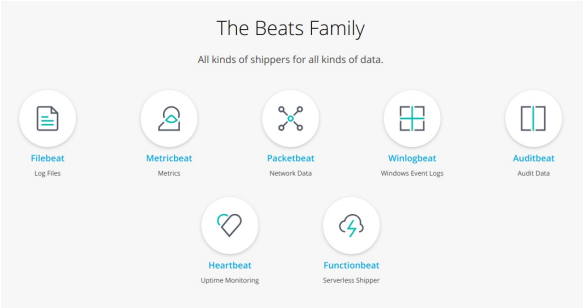

Figura 1: Coletor de dados Beats

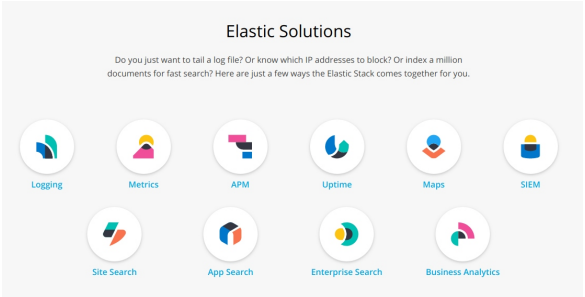

Figura 2: Soluções integradas ELK

Nos dois exemplos citados e em geral na maioria dos casos, as ferramentas são ainda muito recentes e pouco testadas em desempenho e possíveis gargalos ainda pela comunidade. Algumas opções oferecidas pela Elastic também avisam em sua documentação que ainda são features experimentais, podendo ser descontinuadas mais à frente.

Neste cenário, foi feita a decisão de projeto de usar ferramentas já com ampla adesão da comunidade e desempenho conhecido para a primeira versão do sistema de monitoração.

3 O Sistema de monitoração construído

Recapitulando da seção anterior, ao construir uma monitoração para ambientes é nuvem é necessário, a priori e em uma visão macro:

- Definir o que será monitorado, funções que a monitoração terá
- Definição de uma arquitetura de integração entre as ferramentas e funcões que consiga cumprir o máximo dos 6 requisitos desejáveis listados no capítulo anterior.
- Escolha de qual/quais ferramentas serão usadas para implementar uma dada função na monitoração.

Como discutido também, há inúmeras possibilidades de arquitetura e integração entre as funções. A escolha da arquitetura é um ponto crítico pois é o que mais afeta o desempenho do sistema de monitoração e da nuvem monitorada também.

3.1 Arquitetura de Monitoração

De forma a atender bem às demandas no ambiente monitorado do laboratório, e de forma a cumprir ao máximo os 6 requisitos listados para uma boa monitoração, foi escolhido usar, com adaptações, a arquitetura de monitoração e integração para ambientes em nuvem *Seshat*, proposta na dissertação [CON 18].

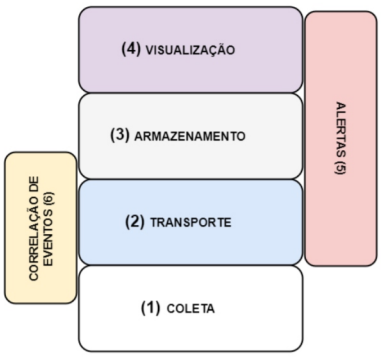

Figura 3: Arquitetura em Camadas *Seshat*

A arquitetura *Seshat* propõe uma divisão das funções de monitoração em camadas. Ela propõe que apenas a camada de coleta esteja nas entidades monitoradas, enquanto todas as outras camadas estejam em servidor dedicado para a monitoração (que, por sua vez, é recomendado que seja distribuído).

As vantagens do uso da arquitetura *Seshat* para o ambiente monitorado no laboratório são inúmeras, pontuadas a seguir.

O requisito desejável 1, ser um sistema de monitoração não invasivo no ambiente monitorado, é atendido na arquitetura *Seshat* ao usar um servidor dedicado para a monitoração e deixar nos hosts apenas a camada de coleta (composta por agentes leves executando nas entidades monitoradas): o menos invasivo possível.

Esta arquitetura com servidor dedicado também possibilita o atendimento do requisito 6, escalabilidade das entidades monitoradas. *Seshat* propõe um modelo *push* para os agentes de coleta onde eles próprios se registram, se desconectam e enviam seus dados para o servidor dinamicamente; em oposição ao modelo *pull* de monitoração (O servidor que faz contato com as entidades monitoradas e atua sobre elas). A proposta de camada de coleta diretamente nos hosts e com modelo *push* de interação permite a escalabilidade fácil das entidades monitoradas, sem precisar mudar em nada a arquitetura ou configurações (o registro/eliminação de uma entidade monitorada é feito dinamicamente por ela mesma).

A coleta direto nos hosts viabiliza também o cumprimento do requisito 4, elasticidade (permitir mudanças e diversidade das entidades monitoradas de forma simples). O modelo *push* de envio de dados coletados permite que mudanças nas entidades monitoradas não impactem o sistema de monitoração.

Por último, a camada de coleta direto nos hosts também facilita muito o cumprimento do requisito desejável 2, **expansão horizontal** da monitoração e aumento de funcionalidades. A possível coleta de novos dados e partes do ambiente é feita direto nos hosts, e a arquitetura montada já provê o encaminhamento deste novo fluxo de coleta até cada destino final possível, o que permite fácil expansão das partes monitoradas (monitorar novos serviços, aplicações, recursos, etc). A implementação prática deste ponto é citada com detalhes mais à frente.

Além disto, a arquitetura em camadas tem um fluxo de circulação dos dados simples, bottom-up, para as métricas coletadas e para os logs coletados. Uma construção com fluxo de dados simples (ao invés de uma construção com fluxo de dados semelhante à caminhar em grafos, com ramificações e ciclos) a simples permite fácil inclusão de novos componentes de monitoração sobre os dados coletados (exemplo: camada de correlações, que usa os dados do fluxo principal em uma função da monitoração fora do fluxo estrutural principal). Assim, atende ao requisito desejável 5, extensibilidade.

Por ser uma arquitetura de fluxo dos dados único e direcionado (da coleta à visualização, sem ciclos ou caminhar em grafos - o que por sua vez aumentaria a possibilidade de entraves locais), o requisito desejável para monitoração de ambientes em nuvem 3, pontualidade, é também atendido. O modelo permite que os dados estejam disponíveis com atraso muito pequeno, o que é ideal para o ambiente monitorado.

3.1.1 Adaptação da arquitetura ao ambiente do laboratório

Seshat foi escolhida por atender a diversos requisitos desejáveis para a monitoração de ambientes em nuvem, de forma a trazer o melhor possível desempenho do sistema de monitoração e do ambiente monitorado. Porém foram feitas nela algumas adaptações às demandas específicas

do laboratório.

Figura 4: Adaptação da arquitetura *Seshat*

Na monitoração do ambiente em nuvem do laboratório, bursts de dados são menos frequentes e não há dados que não possam ser perdidos de forma alguma ou seria crítico. Assim, a clusterização da monitorização (incluindo a camada de transporte, outra forma de lidar com a expansão) não é uma demanda indispensável para a primeira entrega do produto. A camada de transporte e clusterização estão documentadas como funcionalidades extras a serem incluídas nos próximos ciclos de versões do produto, como discutido na seção 2.2.2.3.

A camada *Seshat* de correlações, neste contexto de monitoração, não é indispensável para a primeira entrega do produto. Faz mais sentido que ela seja implementada após a monitoração estar coletando métricas e dados de aplicações, pois esses dados também serão usados nela. Assim, essa camada está listada nas features opcionais futuras mas somente após implementar a outra feature opcional expansão horizontal (métricas de aplicações, além dos logs), também na lista.

Outra adaptação à arquitetura *Seshat* foi trazer a camada de alertas para logo após a camada de coleta. No ambiente monitorado do laboratório uma das funcionalidades mais importantes da monitoração são os alertas. Os alertas avisam em tempo real a indisponibilidade de serviços,

que requer ação imediata neste contexto. Indisponibilidade na nuvem atrapalha o andamento de experimentos e pesquisa sendo feitos usando as VMs fornecidas pelo ambiente em nuvem, congestionam ou congelam o trabalho de todos os usuários. O quanto antes for resolvido melhor; e se não depender de usuários buscarem e conseguirem entrar em contato com os administradores para avisar e para começar a solucionar é o cenário ideal.

Por fim, outra adaptação para o sistema de monitoração para o ambiente em questão, feita na arquitetura *Seshat*, é a eliminação de agentes leves coletores de logs. O ambiente monitorado em questão não precisa de agentes específicos para coleta de logs, apenas os logs padrão do sistema. Há casos em que agentes específicos são necessários (no caso analisado em [CON 18], a análise do comportamento do processamento usando *Spark* e *Hadoop* é muito mais adequado se feito com coletores de logs específicos para aplicações construídas em *Java*). Porém no ambiente analisado aqui ainda não há esta demanda, só a dos logs do sistema. Assim, não há agente de coleta e sim o redirecionamento dos logs do Log Manager do próprio Sistema Operacional para, ao invés de arquivos de log, para enviar para o servidor de monitoração remoto.

3.2 Componentes da Monitoração

Cada parte do ambiente a ser monitorada, e partes das camadas de monitoração, foram implementadas com componentes Software Livre. O desenvolvimento de cada componente em si levaria um tempo muito maior do que o tempo do Projeto Orientado para ficar pronto, e mesmo assim não faria sentido. A adoção destes componentes se dá porque cada um deles já tem amplo uso no mercado, eles tem enormes equipes de desenvolvimento, além da própria comunidade que contribui para o desenvolvimento (são Software Livre). As equipes tem anos de experiência e estudo nos componentes que desenvolvem, além de muitas versões lançadas e testadas e experiência adquirida. Um único desenvolvedor, sozinho, sem tanta experiência teórica e prática na área de cada componente, desenvolvendo cada um desses componentes em um prazo mínimo comparado aos anos de lançamento de versões deles sem sombra de dúvidas traria um componente muito pior do que o que já amplamente utilizado e otimizado. Isso traria prejuízo para o sistema de monitoração. É dito que "*Não faz sentido reinventar a roda*", ainda mais para uma roda pior. Sendo assim, componentes de amplo uso de software livre foram usados como base para implementar funcionalidades da monitoração.

Como visto no diagrama da monitoração, figura 4, os fluxos dos dados na monitoração se dividem em fluxo de logs e fluxo de métricas dentro das camadas (Logs são os componentes verdes e métricas são os componentes rosa). O sentido disso é que dados como logs (dados textuais) ou métricas (dados numéricos discretos) tem campos que os compõe muito diferentes, padrões diferentes e tem componentes otimizados para o uso de cada um dentro de uma mesma funcionalidade (por exemplo Bancos de Dados. Bancos de Dados para dados textuais tem demandas e estruturas diferentes de Bancos da Dados para valores numéricos). Assim, ao longo da análise das camadas são analisados os dois fluxos distintos.

Um ponto importante a se destacar também é que todos os componentes foram implementados usando containers **Docker**. Esta decisão foi feita de modo a otimizar o ambiente de execução de cada componente, e também para permitir a portabilidade do sistema de monitoração sem precisar de extensas configurações. Devido ao uso dos componentes em containers *Docker*, todos em uma única pilha de containers em um único arquivo *docker-compose.yaml*, permite que o simples comando

```
docker-compose up
```

executado dentro do diretório do código da monitoração construa e execute a monitoração em qualquer máquina, sem precisar fazer mais nada.

3.2.1 Camada de Coleta

Logs: Rsyslog

A coleta do logs foi feita usando o prórpio *Log Manager*(LM) do Sistema Operacional de cada entidade monitorada. Essa decisão de eliminação de agente de coleta e uso do próprio LM do host diminui overhead de monitoração: a coleta de logs já é intrínseca ao host, ela só está sendo direcionada para servidor remoto, o que consome muito menos recursos.

Explicação de detalhes da arquitetura do rsyslog, configuração completa, também todas as decisões de projeto de implementação estão no manual de tutoriais e configuração para cada componente, dentro do espaço do projeto no GitLab.

Métricas: Sensu

A escolha do *Sensu* para a coleta de métricas se deu por três fortes motivos.

O primeiro é ele estar entre as opções de coleta com arquitetura cliente-servidor, possibilitando a implementação da arquitetura construída: os *daemons sensu-client* executam nos hosts monitorados, coletando os dados. São agentes leves. E o Sensu tem Servidor, Banco de Dados, Fila de Comunicação, API de acesso e Dashboard de acompanhamento todos o compondo, independentes, com configuração própria cada um, e todos estes permanecem **remotos** ao cliente, no servidor dedicado para monitoração.

O segundo motivo da escolha do Sensu é que ele usa scripts de monitoração nos hosts (conhecidos internamente nele como *plugins*) também usados em diversos outros softwares semelhantes que tem ampla adesão da comunidade (por exemplo o *Nagios*). Essa compatibilidade faz com que ele seja uma excelente opção para monitorar os mais diversos aspectos, possibilitando um sistema de monitoração que monitora diferentes partes, com o uso de uma tão extensa biblioteca de plugins.

O último motivo que torna atrativa a escolha do Sensu é seu amplo uso pela comunidade. Ele está há quase 10 anos no mercado, o que para este tipo de software é um tempo altíssimo.

Já passou por diversas versões e otimizações e tem ampla adoção pela comunidade o que é um indicador de qualidade.

Explicação de detalhes da arquitetura do Sensu, integração e configuração de cada um dos 6 componentes que o compõe, também todas as decisões de projeto de implementação, explicação e diretrizes para novos plugins estão no manual de tutoriais e configuração, dentro do espaço do projeto no GitLab.

3.2.2 Camada de Alertas

Para implementar os Alertas foram usados *handlers*, que são plugins em tratadores de eventos dentro do componete principal dos 6 componentes Sensu, o Sensu server. A escolha de implementar os alertas como handlers dentro do Sensu Server foi para que houvesse a comunicação mais rápida o possível. Um dado coletado que precisa ser alertado passa em primeiro lugar, após sua coleta, no Sensu Server. Assim, para o ambiente monitorado que requer alertas com menor atraso possível, foi feita a escolha por alertas usando o Sensu Server.

Explicação da construção do código para handlers, de como inserir handlers como tratadores de dados em dados sendo coletados, e decisões de projeto de implementação estão no manual de tutoriais e configuração, dentro do espaço do projeto no GitLab.

3.2.3 Camada de Tratamento de Dados

Os dados textuais coletados (logs do rsyslog) precisam sofrer processo de parse para então entrarem no banco de dados textual. Para permitir a busca de texto no Banco de Dados estes dados precisam estar já separados em campos. Essa tarefa é feita usando o Software Logstash.

A escolha do Logstash se deu por ele ser o componente padrão de parse de dados para o banco de dados e motor de busca ElasticSearch. O ElasticSearch é um Banco de Dados de time-series para dados textuais e também forte motor de busca sobre eles. A pilha ELK, composta por ElasticSearch, Logstash e Kibana são os softwares integrados para tratamento de logs e dados textuais mais consolidados no mercado. Assim, foi feita a escolha da pilha ELK para o tratamento de logs, e dentro dela o Logstash para esta camada essencial de parsing dos dados.

Explicação de detalhes da arquitetura do logstash, configuração completa, também todas as decisões de projeto de implementação estão no manual de tutoriais e configuração para cada componente, dentro do espaço do projeto no GitLab.

3.2.4 Camada de Armazenamento

A camada de armazenamento armazena os dados de monitoração coletados.

Logs: ElasticSearch

A escolha como Banco de Dados ElasticSearch se motiva por este ser o banco de dados temporal para dados textuais e motor de busca mais consolidado no mercado.

Explicação de detalhes da arquitetura do ElasticSearch, configuração completa, também todas as decisões de projeto de implementação estão no manual de tutoriais e configuração para cada componente, dentro do espaço do projeto no GitLab.

Métricas: InfluxDB

InfluxDB é um banco de dados NoSQL , time-series, otimizado para valores numéricos como métricas.

A escolha dele como Banco de Dados foi devida às suas características citadas acima, além de desempenho conhecido, ser muito consolidado e usado por diversas organizações. Além disso o InfluxDB possui várias formas diferentes, desenvolvidas pela comunidade, de se conectar ao Sensu. Ele possui também opção para integração direta com o Grafana, melhor software para visualizações neste contexto.

Explicação de detalhes da arquitetura do InfluxDB, configuração completa, também todas as decisões de projeto de implementação estão no manual de tutoriais e configuração para cada componente, dentro do espaço do projeto no GitLab.

3.2.5 Camada de Visualização

Logs: Kibana

Kibana é uma interface gráfica para interação com o motor de busca dos logs, o ElasticSearch. Além de interface gráfica e de exibir resultados de pesquisas graficamente, agora o Kibana também permite construir gráficos estatísticos com os dados.

Explicação de detalhes da arquitetura do Kibana, configuração completa, também todas as decisões de projeto de implementação estão no manual de tutoriais e configuração para cada componente, dentro do espaço do projeto no GitLab.

Métricas: Grafana

Grafana é um software que permite a criação de Dashboards interativos a partir da conexão com uma fonte de dados, que na maioria das vezes é um Banco de Dados time-series. Ele tem enorme adesão pela comunidade e organizações, é um dos maiores repositórios do GitHub, com milhares de colaboradores, extremamente sólido e com opções de configuração. Todos estes pontos motivaram sua escolha para como ferramenta para a construção dos Dashboards.

Explicação de detalhes da arquitetura do Grafana, configuração completa, montagem de Dashboards, provisionamento e automação de sources e dashboards após construídos, formas diversas de autenticação, também todas as decisões de projeto de implementação estão no manual de tutoriais e configuração para cada componente, dentro do espaço do projeto no GitLab.

Visão do data center: Uchiwa

O Uchiwa é um componente do Sensu que permite o acompanhamento das entidades monitoradas. Foi escolhido para uso por ser de fácil uso e já possuir integração com o Sensu.

Explicação de detalhes da arquitetura do Uchiwa, configuração completa, integração com o componente Sensu API e também todas as decisões de projeto de implementação estão no manual de tutoriais e configuração para cada componente, dentro do espaço do projeto no GitLab.

4 Conclusão

4.1 Resultados e Discussões

A monitoração foi construída e está funcionando, como é possível ver acessando-a com os links disponibilizados na seção 1 e também nas imagens aqui:

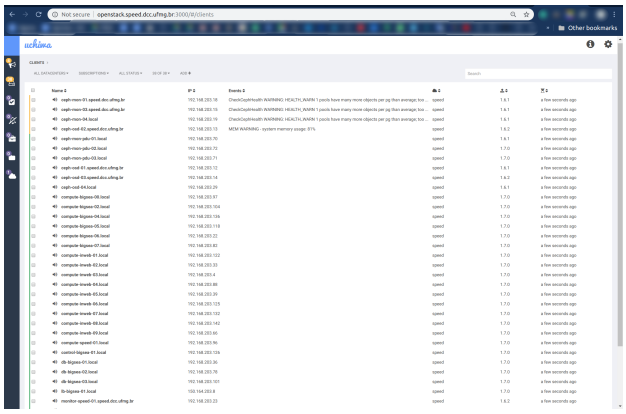

Figura 5: Lista de hosts sendo monitorados

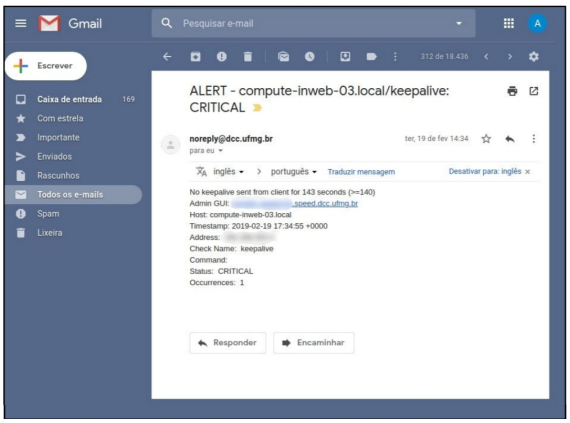

Figura 6: Alerta de indisponibilidade (queda)

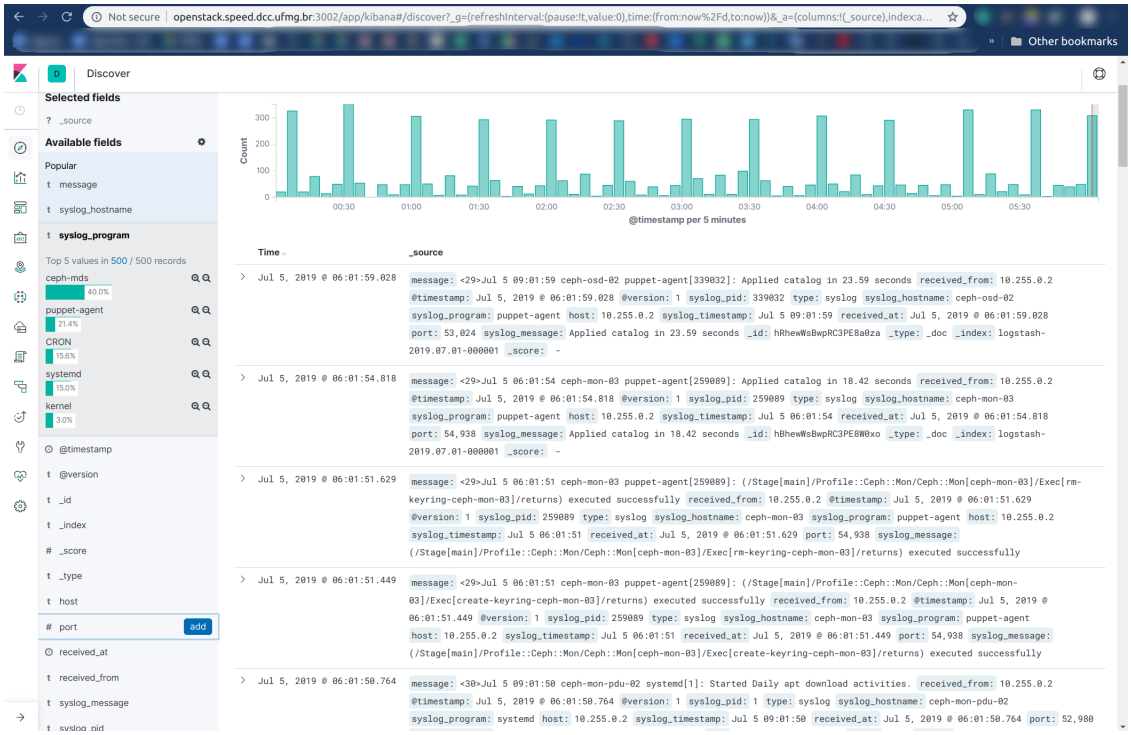

Figura 7: Logs sendo coletados nos hosts

Figura 8: Parte dos Dashboards de análise de métricas

Ao construir a monitoração primeiro foi construído o fluxo de métricas (fluxo rosa da figura 4), e depois o fluxo de logs (fluxo verde da mesma figura). Assim, os dashboards de métricas construídos com *Grafana* já estavam prontos enquanto o fluxo de logs era construído. A partir deles já foi possível identificar alguns gargalos e melhorar o desempenho do ambiente em nuvem monitorado mesmo antes da conclusão total de monitoração:

- **Aumento do número de cores de processamento em alguns hosts**

  Cruzando as informações nos Dashboards Grafana em dados instantes no tempo, foi possível ver que em alguns hosts deveria estar acontecendo um gargalo devido ao baixo número de núcleos de processamento.

  Resultado: Administradores aumentaram o número de núcleos nas máquinas e o desempenho delas melhorou muito.

- **Troca de Armazenamento distribuído CEPH para armazenamento local**

  Nos Dashboards Grafana foi possível ver picos momentâneos de longos tempos de espera para escrita ou leitura em disco, chegando a 12 segundos (o valor esperado é da ordem de grandeza de milissegundos). Apesar de serem apenas picos e de serem pouco frequentes e bem distantes de tempo um do outro, picos de tempo de espera para operações de leitura e escrita dessa ordem de grandeza são inaceitáveis. Cruzando com as informações de tráfego de rede nestes instantes, foi decidido então testar a troca do armazenamento distribuído por armazenamento local.

Resultado: Os picos de tempo de espera para leitura e escrita em disco diminuíram de pior caso 12 segundos para pior caso 5 segundos, mas na maioria 3 segundos quando ocorrem.

- **Identificação de picos de carga intervalados e sempre iguais em diversos os hosts**

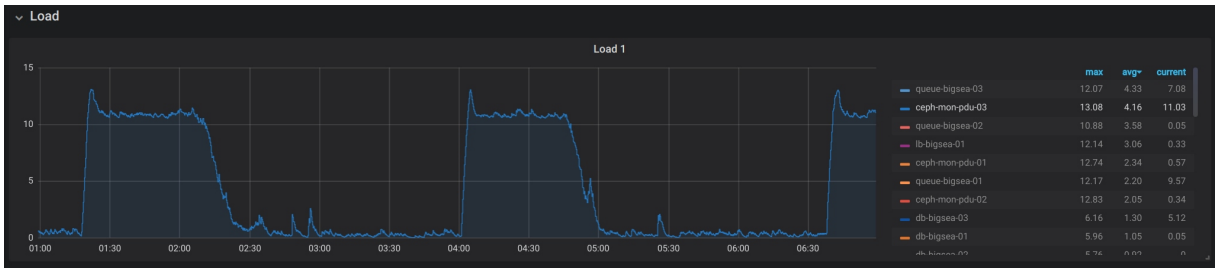

Figura 9: Exibição de Curvas de Load para média 1 min em um host

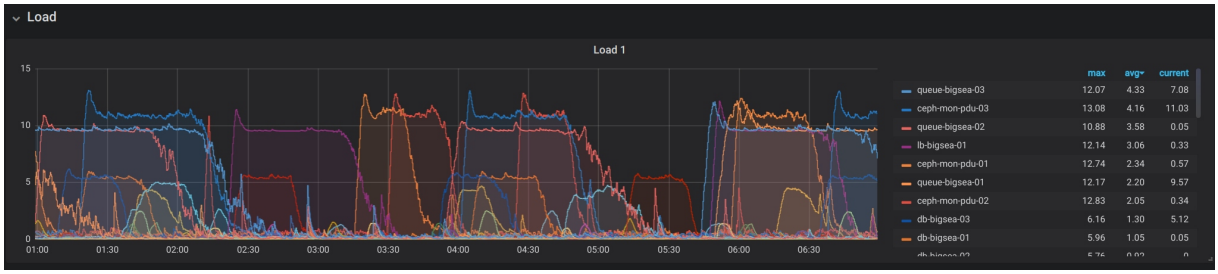

Figura 10: Exibição de Curvas de Load para média 1 min em vários hosts

Foi identificado que todos os hosts tem picos de carga com mesmo padrão de pico e repetido periodicamente. A forma do pico/curva e o intervalo de frequência entre elas são sempre iguais dentro da mesma entidade monitorada, porém a forma da curva e o intervalo de tempo entre as curvas varia muito ao se comparar a típica de cada entidade monitorada.

Resultado: Pretende-se cruzar as informações de logs com carga de modo a identificar o que está causando este comportamento (No momento de escrita deste manual o fluxo de logs tinha ficado pronto recente, logo ainda não tinha sido usado para análises).

- **Identificação de problema com arquivamento de dados antigos em Banco de Dados de entidade monitorada**

  Após a troca para armazenamento local e não distribuído, em uma das entidades monitoradas o consumo de disco estava aumentando muito rápido. Foi então feita uma análise dentro da entidade para encontrar onde estava essa quantidade de dados armazenados aumentando tão rápido. Essa região era região de armazenamento de dados de um Banco de Dados local. Foi observado que estava havendo um problema com o arquivamento e descarte de dados antigos.

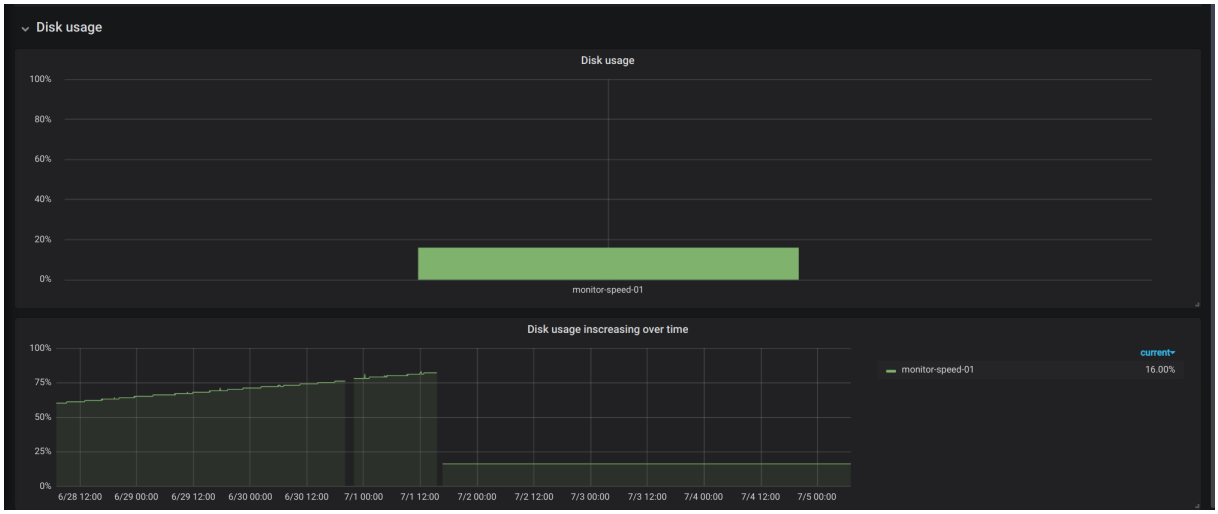

Figura 11: Consumo de disco. As falhas no gráfico são o tempo que a entidade monitorada passou por queda (1a falha) e manutenção (2a falha).

Resultado: Foi identificado o problema e consertado.

4.2 Trabalhos Futuros: Possíveis expansões de funções

Com a monitoração pronta, o primeiro que está sendo feito é a identificação e correção de gargalos encontrados através dela.

A partir desta versão lançada pretende-se ir progressivamente incluindo features opcionais e extras nela e também atualizando os componentes dela conforme atualizações deles sejam lançadas. Como discutido na seção 2.2.2.3, ao longo da construção todas as features extras possíveis encontradas foram documentadas no Gitlab dentro da Milestone ”Pós-POC”do repositório do sistema de monitoração. Assim, diversas features extras possíveis para a monitoração estão descritas e documentadas lá (por exemplo: template personalizado para as notificações de indisponibilidade de uma entidade monitorada por e-mail).

Entre os extras possíveis, o primeiro do Roadmap desejado a se fazer é uma **expansão horizontal**: monitorar mais partes das entidades, aumentar a abrangência da monitoração. O primeiro passo pretendido é coletar métricas e dados relativos ao funcionamento do Openstack e de aplicações.

Devido à arquitetura do sistema de monitoração, realizar a expansão horizontal é muito simples:

- Instalar, com apenas um comando, novo script de monitoramento na entidade monitorada
- Duplicar um arquivo de checks na entidade monitorada e apenas trocar nele para o nome do novo script/plugin de monitoramento.
- No Grafana, duplicar um Painel e mudar o nome do dado que se deseja exibir (que já irá aparecer na lista dos dados possíveis).

Após a expansão horizontal, a próxima feature do RoadMap é a clusterização do servidor de monitoração. E assim as features extras vão sendo implementadas, como um desenvolvimento de software comum: primeiro é lançado o indispensável, e em novas versões são incluídas mais funcionalidades a cada versão.

Antes da implementação das features extras também pretede-se disponibilizar o sistema construído no GitHub. Antes é necessário remover dados sensíveis, atualizar os READMEs do repositório e traduzi-los para apenas em inglês.

4.3 Conclusões

A monitoração em funcionamento permite agora acompanhar melhor o andamento do ambiente monitorado, identificar problemas e otimizar seu uso.

O sistema ser facilmente extensível permite que ele se adapte à mudanças no ambiente monitorado e inclua com facilidade novas funcionalidades de monitoração, para que persista ao longo do tempo e de mudanças e não se torne obsoleto.

Agora no instante inicial o primeiro foco está em resolver problemas identificados pela monitoração no ambiente monitorado.

Referências

[ACE 13] ACETO, G. et al. Cloud monitoring: A survey. **Computer Networks**, [S.l.], v.57, n.9, p.2093–2115, 2013.

[ALE] ALECRIM, E. **Cluster: conceito e características**. https://www.infowester.com/cluster.php. Acessado em 2017.

[AND] ANDREAS, S. **Fotos e mapa dos data centers do Google nlo Mundo**. http://forum.outerspace.com.br/index.php?threads/fotos-e-mapa-dos-data-centers-do-google-nlo-mundo-gigantesco-data-center-da-nsa-em-utah-354827/. Acessado em 2017.

[APH a] APHYR. **Query Grammar**. https://github.com/riemann/riemann/blob/master/resources/query.g4. Acessado em 2017.

[APH b] APHYR. **Query test**. https://github.com/riemann/riemann/blob/master/test/riemann/query_test.clj. Acessado em 2017.

[ASH] ASHRAF, W. **Introduction to docker swarm**. https://www.slideshare.net/WalidAshraf2/introduction-to-docker-swarm-69294913. Acessado em 2017.

[BEN] BENTLEY, W. **OpenStack and Virtualization: What's the Difference?** https://blog.rackspace.com/openstack-and-virtualization-whats-the-difference. Acessado em 2017.

[BUY 99] BUYYA, R. High performance cluster computing. **New Jersey: F'rentice**, [S.l.], 1999.

[BV] BV, E. **Elastic Search**. https://www.elastic.co/products/elasticsearch. Acessado em 2017.

[COM a] COMPUTING, R. C. **Hypervisors**. https://docs.openstack.org/ocata/config-reference/compute/hypervisors.html. Acessado em 2017.

[COM b] COMPUTING, R. C. **What is OpenStack?** https://www.openstack.org/software/. Acessado em 2017.

[CON 18] CONCEIçãO, V. S. **Uma arquitetura de monitoração escalável para ambientes em nuvem**. Universidade Federal de Minas Gerais, 2018. Dissertação de Mestrado.

[das] **Riemann monitors distributed systems**. http://riemann.io. Acessado em 2017.

[DBI] **InfluxDB**. https://en.wikipedia.org/wiki/InfluxDB. Acessado em 2017.

[DG a] DIAZ-GONZALEZ, J. **beaver**. https://python-beaver.readthedocs.io/en/latest/. Acessado em 2017.

[DG b] DIAZ-GONZALEZ, J. **Python-Beaver**. https://github.com/python-beaver/python-beaver. Acessado em 2017.

[GUE 12] GUEDES, D.; FERREIRA, R.; MEIRA, W. **Minicursos da Escola Regional de Informática de Minas Gerais**, chapter10, p.1– 25. ERI-MG, 2012. Capítulo Processamento de dados massivos.

[HAG] HAGELBERG, P. **Leiningen**. https://leiningen.org/. Acessado em 2017.

[INC a] INC., D. **Best practices for writing Dockerfiles**. https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/. Acessado em 2017.

[INC b] INC., D. **docker**. https://docs.docker.com/engine/reference/commandline/docker/. Acessado em 2017.

[INC c] INC., D. **Dockerfile reference**. https://docs.docker.com/engine/reference/builder/. Acessado em 2017.

[INC d] INC., D. **How services work**. https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/. Acessado em 2017.

[INC e] INC., S. **Number of daily active Facebook users worldwide as of 3rd quarter 2017 (in millions)**. https://www.statista.com/statistics/346167/facebook-global-dau/. Acessado em 2017.

[INC 16] INC., D. **eBook: Docker for the Virtualization Admin**.

[INF] INFLUXDATA, I. **InfluxDB**. https://www.influxdata.com/time-series-platform/influxdb/. Acessado em 2017.

[KIN a] KINGSBURY, K. **Manual: Clojure from de ground up**. https://aphyr.com/tags/Clojure-from-the-ground-up, Acessado em 2017.

[KIN b] KINGSBURY, K. **Working with Riemann**. Monitorama PDX 2015 Session.

[KRA] KRAZIT, T. **The incredible shrinking operating system: How containers and serverless computing are changing the cloud**. https://www.geekwire.com/2017/will-operating-systems-evolve-cloud-era-containers-serverless-changing-game/. Acessado em 2017.

[MEL 11] MELL, P.; GRANCE, T.; OTHERS. The nist definition of cloud computing. [S.l.], 2011.

[MSE] **Sensu (Monitoring)**. https://de.wikipedia.org/wiki/Sensu_(Monitoring). Acessado em 2017.

[OPS] **Openstack**. https://pt.wikipedia.org/wiki/Openstack. Acessado em 2017.

[RAB] **RabbitMQ**. https://en.wikipedia.org/wiki/RabbitMQ. Acessado em 2017.

[RIE] **Streams**. http://riemann.io/concepts. Acessado em 2017.

[ROU] ROUSE, M. **Hypervisor**. http://searchservervirtualization.techtarget.com/definition/hypervisor. Acessado em 2017.

[SAN a] SANTANA, L. **Big Data com Apache Spark - Parte 1: Introdução**. https://www.infoq.com/br/articles/apache-spark-introduction#. Acessado em 2017.

[SAN b] SANTANA, L. H. Z. **Elasticsearch: Como gerenciar logs com Logstash**. https://www.devmedia.com.br/elasticsearch-como-gerenciar-logs-com-logstash/32939. Acessado em 2017.

[SCH] SCHWEDER, J. A. **Extração de dados com Logstash**. http://www.butecopensource.org/logstash/. Acessado em 2017.

[SEN] SENSU, I. https://sensuapp.org/. Acessado em 2017.

[SHA] SHAPLAND, R. **Cloud containers – what they are and how they work**. http://searchcloudsecurity.techtarget.com/feature/Cloud-containers-what-they-are-and-how-they-work. Acessado em 2017.

[SIX] SIX, G. T. **Is OpenStack a new hypervisor or what is really?** https://gusgon26.wordpress.com/2014/07/26/is-openstack-a-new-hypervisor-or-what-is-really/. Acessado em 2017.

[SOF] SOFTWARE, P. https://www.rabbitmq.com/. Acessado em 2017.

[SPA] **Linux namespaces**. https://en.wikipedia.org/wiki/Linux_namespaces. Acessado em 2017.

[WHI] WHITE, C. **Isolation with Linux Containers**. https://www.engineyard.com/blog/isolation-linux-containers. Acessado em 2017.

[WIE] **ElasticSearch**. https://pt.wikipedia.org/wiki/ElasticSearch. Acessado em 2017.

[WIL] WILSON, B. **What Is Docker? How Does It Work?** https://devopscube.com/what-is-docker/. Acessado em 2017.