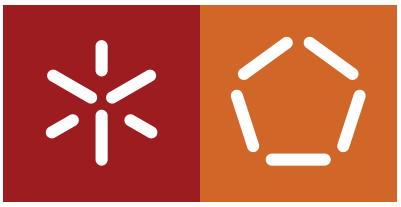

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Mestrado em Engenharia Informática

Bruno Miguel Pereira Andrade

Gestão e análise de eventos de segurança

Uma abordagem ao transporte e armazenamento de eventos em ambientes *Big Data*

Dissertação de Mestrado

Orientada por:
Vítor Francisco Fonte, Professor Auxiliar, Dep. Informática, Universidade do Minho

Supervisionada por:
Nuno Fernandes, Dir. I&D, Eurotux Informática S.A.

Braga, 31 de Outubro de 2015

Anexo 3

DECLARAÇÃO

Nome
Bruno Miguel Pereira Andrade

Endereço electrónico: brunompandrade@gmail.com Telefone: +351936293858 / _____

Número do Bilhete de Identidade: 13751534

Título dissertação ☒/tese ☐
Gestão e Análise de Eventos de Segurança: Uma abordagem ao Transporte e armazenamento de eventos em ambientes Big Data

Orientador(es):
Vítor Francisco Fonte

Ano de conclusão: 2015

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:
Mestrado Engenharia Informática

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

2. É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE/TRABALHO (indicar, caso tal seja necessário, nº máximo de páginas, ilustrações, gráficos, etc.), APENAS PARA EFEITOS DE INVESTIGAÇÃO, , MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

3. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Universidade do Minho, 30/10/2015

Assinatura: Bruno Miguel Pereira Andrade

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer aos meus pais por me terem dado a possibilidade de seguir a vida Acadêmica e por me terem apoiado nas minhas decisões.

À Eurotux Informática e a todos os seus colaboradores, em especial aos seus Administradores e ao Nuno Fernandes, por me terem dado a oportunidade de aprender e evoluir profissional e pessoalmente, naquele espaço.

Ao meu orientador, Professor Vítor Francisco Fonte e ao Professor José Orlando Pereira pelo acompanhamento inicial e durante todo processo de elaboração desta dissertação.

Aos meus amigos que sempre se demonstraram disponíveis para me ajudar e deixar de lado as preocupações e complicações associadas à elaboração de um documento deste género.

Ao meu irmão, cunhada e sobrinho por sempre me impulsionarem a completar esta etapa.

Por fim, a todos aqueles que de alguma forma possam ter contribuído directa, ou indirectamente, para a elaboração desta dissertação e que por algum motivo não estão aqui mencionados.

ABSTRACT

Security information and event management in cloud environments leads us to new challenges, due to the data volume and its dispersion. On the other hand, Big Data techniques and tools bring new opportunities for storage and analysis of this kind of information, at great scale. With that in mind, we can aim at collection and indexing this kind of information in quasi real-time.

In this context, the goal of this dissertation is to design and develop a system, based on opensource components, that can solve this task.

RESUMO

A gestão e análise de eventos de segurança em ambientes *cloud* traz novos desafios quer no volume de dados quer na sua dispersão. Por outro lado, as técnicas e ferramentas *Big Data* trazem novas oportunidades de armazenamento e análise deste género de informação em grande escala. Pode-se assim ambicionar transparte e indexação deste tipo de informação em (quase) tempo-real.

Neste contexto, esta dissertação tem como objectivo o projeto e desenvolvimento de um sistema, baseado em componentes *opensource*, para a realização desta tarefa.

"*Whenever you find yourself on the side of the majority, it is time to pause and reflect.*"

Mark Twain

CONTEÚDO

Conteúdos iii

1 INTRODUCÃO 4
- 1.1 Segurança 5
- 1.2 Eventos de Segurança 8
- 1.3 Enquadramento 9
- 1.4 Contributo 10
- 1.5 Estrutura do Documento 10

2 ESTADO DA ARTE 12
- 2.1 SIEM 12
  - 2.1.1 1º Geração: IDS 12
  - 2.1.2 2º Geração: SIEM 13
  - 2.1.3 3ª Geração: *Next-Gen* SIEM 14
- 2.2 Transporte 14
  - 2.2.1 Rsyslog 15
  - 2.2.2 Logstash 15
  - 2.2.3 Fluentd 17
  - 2.2.4 Apache Kafka 17
- 2.3 Armazenamento 19
  - 2.3.1 Indexação 19
  - 2.3.2 Arquivo 26
- 2.4 Processamento 27
  - 2.4.1 Batch 28
  - 2.4.2 Stream 28
- 2.5 Visualização 30
  - 2.5.1 Kibana 31
  - 2.5.2 D3.JS 33

3 ABORDAGEM AO PROBLEMA 35
- 3.1 Arquitectura 35
  - 3.1.1 Localização única 36
  - 3.1.2 Distribuição geográfica 37

3.1.3 Distribuição geográfica e armazenamento diferenciado 38
3.1.4 Distribuição geográfica e Escalabilidade 40
3.2 Integração das tecnologias 41
3.2.1 Transporte 41
3.2.2 Armazenamento 44
3.2.3 Processamento 45
3.2.4 Visualização 46

4 AVALIAÇÃO 48
4.1 Transporte 48
4.2 Armazenamento 52
4.2.1 Longo-Prazo 52
4.2.2 Curto-Prazo 59

5 CONCLUSÃO 66
5.1 Trabalho futuro 67

A TRANSPORTE 70
A.1 Fluentd 70
A.1.1 Instalação e Configuração 70
A.1.2 Configuração e execução dos testes 72
A.2 Logstash 75
A.2.1 Instalação e Configuração 75
A.2.2 Configuração e execução dos testes 76

B ARMAZENAMENTO 78
B.1 ElasticSearch 78
B.1.1 Instalação e Configuração 78
B.1.2 Configuração e execução dos testes 80
B.2 Apache Solr 83
B.2.1 Instalação e Configuração 83
B.2.2 Configuração e execução dos testes 86
B.3 Apache Hadoop HDFS 87
B.3.1 Instalação e Configuração 87
B.3.2 Configuração e execução dos testes 94

LISTA DE FIGURAS

Figura 1 *Dashboard* configurada com recurso ao *Kibana* 4. 32
Figura 2 *Dashboard* criada com recurso ao *Banana*. 33
Figura 3 *Dashboard NOW*, disponibilizada pela *GoSquared*. 34
Figura 4 *Dashboard "The incredible rise of migrants' remittances"*. 34
Figura 5 Modelo simples e de localização geográfica única 36
Figura 6 Distribuição geográfica simples 37
Figura 7 Distribuição geográfica e armazenamento diferenciado 38
Figura 8 Distribuição geográfica e Escalabilidade 40
Figura 9 Interesse nas soluções *Rsyslog*, *Logstash*, *FluentD* e *Apache Kafka*, recorrendo ao motor de busca *Google*. 42
Figura 10 Interesse nas soluções *Elasticsearch* e *Apache Solr*, recorrendo ao motor de busca *Google*. 45
Figura 11 Interesse nas soluções *Apache Storm* e *Apache Spark*, recorrendo ao motor de busca *Google*. 46
Figura 12 Interesse nas soluções *Kibana* e *D3.js*, recorrendo ao motor de busca *Google*. 47
Figura 13 Comparação de desempenho entre *Fluentd* e *Logstash*, durante 30 minutos, para uma geração de 5000 *eventos/seg*. 49
Figura 14 Comparação de desempenho entre *Fluentd* e *Logstash*, durante 30 minutos, para uma geração de 10000 *eventos/seg*. 50
Figura 15 Comparação de desempenho entre *Fluentd* e *Logstash*, durante 30 minutos, para uma geração de 50000 *eventos/seg*. 51
Figura 16 Interrogação com recurso a *Term*, em *Elasticsearch* 61
Figura 17 Interrogação com recurso a *Query String*, em *Elasticsearch* 62
Figura 18 Interrogação simples em *Apache Solr* 62
Figura 19 Exemplo de um gráfico gerado pelo *Vegeta* apartir de um dos resultados. 83

LISTA DE TABELAS

Tabela 1 Resultado da primeira execução do teste de leituras aleatórias sobre Hadoop Distributed FileSystem (HDFS) 53
Tabela 2 Resultado da segunda execução do teste de leituras aleatórias sobre HDFS 53
Tabela 3 Resultados dos testes de escrita para HDFS 55
Tabela 4 Latência média, em *ms*, durante uma execução de 60 segundos a ambas as soluções de indexamento. 64
Tabela 5 Latência média, em *ms*, durante uma execução de 600 segundos a ambas as soluções de indexamento. 64
Tabela 6 Percentagem de sucesso nos pedidos durante uma execução de 60 segundos a ambas as soluções de indexamento. 64
Tabela 7 Percentagem de sucesso nos pedidos durante uma execução de 600 segundos a ambas as soluções de indexamento. 65

1

INTRODUCÃO

Atualmente, existe uma preocupação cada vez maior com a segurança dos dados e infraestruturas tecnológicas. Isto deve-se, principalmente, ao facto de todos os dias serem reveladas notícias sobre, *e.g.*, vigilância eletrónica, ataques Distributed Denial of Service (DDoS) e sistemas informáticos que são evadidos. Estes tipos de ataque permitem aos seus responsáveis ter acesso a informação privada e, muitas vezes, confidencial.

Uma notícia lançada a 4 de Dezembro de 2014, pela *Newsweek*, um serviço noticioso generalista, diz o seguinte[1]:

> *"Sony Pictures Entertainment experienced one of the most devastating corporate attacks in history. Thousands of files, seized by hackers last week, have been leaked online including personal details of around 6,000 Sony employees, upcoming Sony feature films and the salary details of top executives.*
>
> *The hackers also managed to access details about Deloitte bankers who are Sony's auditors.*
>
> *The initial data breach, which occurred on 24th November, resulted in the shutdown of the entire computer network of one of Hollywood's largest and most powerful studios."*

Este tipo de exposição faz com que as empresas comecem a tomar medidas que permitam prevenir este tipo de ataques, bem como salvaguardar os seus dados, infraestruturas e, principalmente, os seus utilizadores. Estas medidas levam ao investimento em mecanismos que lhes permitam, além de manter os seus dados seguros, ter um maior controlo sobre a sua infraestrutura, *i.e.*, mecanismos que permitam a prevenção e deteção de possíveis intrusões ou outro tipo de ataques.

1 Fonte: http://www.newsweek.com/sony-cyber-attack-worst-corporate-history-thousands-files-are-leaked-289230

Com tudo isto, e com o facto de existirem cada vez mais pessoas e dispositivos eletrónicos conectados entre si e à *Internet*, faz com que a segurança das infraestruturas e, em particular, a segurança da informação, tenham um papel cada vez mais importante dentro das empresas.

Antes de continuar, é necessário definir o que é a segurança, qual a sua importância e a sua história até aos dias de hoje.

No livro *The InfoSec Handbook* [5], a segurança é definida como:

> *"Security, in simple terms, is protecting what you or others have. This same idea applies to entities like government departments, agencies, companies, institutes, and so on, irrespective of their size or function."*

Outra passagem também importante e que permite comparar a segurança a um instinto básico:

> *"A basic animal instinct is to ensure one's own "safety." Every animal, including a human, will fight for its safety. Everyone wants to be safe and preserve whatever they have with them whether that be assets, money, or otherwise."*

Desde sempre o Ser Humano tem a necessidade de proteger e de se certificar que a sua propriedade está em segurança. Com o passar do tempo, há cada vez uma maior necessidade de garantir a segurança, não só dos bens físicos, mas também dos não físicos, *e.g.*, propriedade intelectual, informações confidenciais, *etc*.

Com o aumento, dia após dia, na utilização de dispositivos móveis como *smartphones*, *tablets* e mais recentemente com o conceito da Internet of Things (IoT), torna-se cada vez mais importante explorar a segurança, que cada vez é mais desafiante, uma vez que todos estes dispositivos são capazes de se conectarem entre si e à *Internet*.

Apesar de tudo o que foi dito até aqui, porque é a segurança realmente importante? Porque devem as empresas investir o seu tempo e dinheiro nela?

1.1 SEGURANÇA

Todas as entidades possuem grandes quantidades de dados que não deveriam ser expostos a terceiros de forma a garantir o seu futuro. Estas entidades possuem uma "consciência própria" que permite que se distingam e se diferenciem entre si, mas faz também com que os seus dados sejam de extrema importância.

Como foi falado no início deste capítulo, a probabilidade de uma, qualquer, entidade ser vítima de um ataque à sua infraestrutura, tecnológica, aumenta todos os dias. Este aumento deve-se a um crescendo no número de agentes maliciosos, *"black hats"*, que vêem neste tipo de ataques uma forma de aumentar o ego e ganhar respeito dentro da "comunidade *hacker*". Na grande maioria dos casos, os dados obtidos através de um destes ataques são disponibilizados livremente na *Internet*. Mas estes agentes maliciosos estão a compreender que podem lucrar bastante com estes ataques e, quanto mais confidencial a informação recolhida for, mais importante e lucrativa ela será. Esta é uma das razões pelas quais todas as entidades deverão começar a investir em sistemas de segurança, principalmente na segurança da informação.

Para afirmar ainda mais a importância da segurança nos dias de hoje, ficam aqui mais duas passagens do livro *The InfoSec Handbook*:

> *"Science and technology provide many tools which are at the disposal of entities and people that can be used for either good purposes or bad purposes. Bad guys can always use such facilities or tools for bad purposes. For example, a security tool like Metasploit or Nessus or nMap, if placed in an auditor's hands, can harden infrastructure, whereas in a cracker's hands they become the go-to tools for criminal activity. A proper focus on information security allows only the required details about the entity or person to be known to the outside world. If any entity or person wants "peace of mind" in today's connected world, information security is a MUST."*

> *"With a lot of information getting distributed easily across the globe because of Web and Cloud technologie, there are a lot of challenges to ensure that data and information of value are well protected so that they are not compromised."*

Após entender a importância da segurança é necessário conhecer formas de a implementar e assim, tentar de alguma forma, detetar e prevenir situações que possam levar a problemas de comprometimento da informação, seja ela informação confidencial ou informação de clientes.

Inicialmente, é necessário perceber o contexto onde a entidade se enquadra e tentar determinar quais as medidas necessárias e agir em conformidade com o tipo de infraestrutura e sua importância. Há algumas décadas, seria apenas necessário, na sua grande maioria, proteger a infraestrutura fisicamente, através de portas, cartões de identificação, guardas, chaves, *etc.*. Mas, a grande maioria das entidades encontram-se já conectadas à *Internet* e possuem inúmeros servidores, muitos deles distribuídos globalmente, fazendo com que assegurar apenas a segurança física dos equipamentos não seja suficiente, tendo de recorrer a outros mecanismos de controlo. Um desses mecanismos, extremamente conhecido e largamente usado é o conceito de *firewall*. Uma *firewall* é basicamente um serviço que permite controlar o acesso via rede às infraestruturas e serviços. Apesar de este ser um mecanismo relativamente simples de utilizar, ainda existem milhões de computadores conectados à *Internet* de

forma não segura e, por esta razão, as distribuidores de sistemas operativos estão a incorporar, por predefinição, serviços de *firewall*. Apesar das *firewalls* serem um mecanismo que nos permite já ter algum tipo de segurança, não é suficiente.

Todos os dias são descobertas novas vulnerabilidades de *software* que põem em causa a segurança das infraestruturas e onde nos piores casos podem levar mesmo a que agentes maliciosos ganhem o controlo total. Por exemplo, um empresa que permita o acesso à sua infraestrutura, via Secure Shell (SSH), aos seus funcionários ou pessoas de "confiança", pode muito bem estar vulnerável a ataques que permitam a agentes maliciosos obter o controlo sobre uma ou várias máquinas da infraestrutura, sem que a sua *firewall* bloqueie o agente. Isto porque o problema, neste caso, seria uma vulnerabilidade no SSH. Como nas políticas da empresa, o acesso via SSH à sua infraestrutura era autorizado, a *firewall* estaria a permitir todo ou quase todo o tráfego para o serviço SSH, deixando assim a infraestrutura vulnerável.

Mesmo combatendo estas vulnerabilidades e optando sempre pelas mais recentes versões de *software*, nunca será possível observar um sistema que seja 100% seguro. Tudo isto porque para o número de vulnerabilidades conhecidas e corrigidas poderá existir em igual número, vulnerabilidades desconhecidas publicamente, existindo sempre a probabilidade das mesmas estarem a ser exploradas por agentes maliciosos.

Deve-se então apostar em políticas de segurança e mecanismos que permitam detetar possíveis intrusões.

Estes mecanismos são denominados de Security Information and Event Management (SIEM). Um SIEM combina a gestão de eventos de segurança, Security Events Management (SEM), e a gestão de informações de segurança, Security Information Management (SIM). O principal objetivo deste tipo de sistema é a agregação e correlação de eventos de segurança (ver Secção 1.2), análise automática de eventos, geração de alertas e retenção dos eventos em arquivo para facilitar uma análise histórica para, *e.g.*, detetar intrusões que possam ter ocorrido no passado sem serem detetadas. Tendo em atenção as características de um SIEM, tal como foram enunciadas anteriormente, compreende-se que seja necessário que as empresas disponibilizem uma infraestrutura dedicada a este tipo de sistema, o que implica gastos financeiros. Além de custos em *hardware*, existem os custos em recursos humanos especializados para o controlo da infraestrutura e análise dos dados. Existem já vários sistemas *SIEM* no mercado, soluções comerciais, que oferecem as funcionalidades necessárias e suficientes a uma grande maioria das empresas existentes no mercado. No entanto, existem cada vez mais empresas voltadas para a *cloud* e que lidam já, diariamente, com o conceito de *Big Data*. É principalmente para estas últimas que o objetivo desta dissertação recaí.

Estas empresas, de um modo geral, estão se a voltar para a disponibilização de serviços *online*. Estes serviços, são na sua maioria Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) ou

Infrastructure-as-a-Service (IaaS). Estes serviços fazem com que a necessidade por infraestruturas cada vez mais distribuídas e passíveis de escalar suba, o que aumenta também a dificuldade no seu controlo. Este tipo de investimento oferece comodidade aos consumidores e estes tendem a responder positivamente, passando a utilizar estes serviços, que permitem aceder aos seus dados, serviços, infraestruturas, *etc*., em quase qualquer lugar a partir do seu *smartphone*, *tablets* e/ou computador pessoal.

Os sistemas SIEM são então um desafio cada vez maior, quer pelo volume de dados, quer pela sua dispersão. No entanto, as técnicas e ferramentas *Big Data* trazem novas oportunidades para o armazenamento e processamento necessário a este tipo de sistemas.

1.2 EVENTOS DE SEGURANÇA

Nos dias de hoje qualquer infraestrutura, mesmo que pequena, poderá produzir milhões de eventos por segundo. Estes eventos, analisados, podem passar informações extremamente importantes sobre a nossa infraestrutura e podem ser gerados por aplicações de segurança, *e.g.*, *firewalls*, Network-based Intrusion Detection System (NIDS)/Host-based Intrusion Detection System (HIDS), Anti-Vírus, *etc*. Podem também ser gerados por outras aplicações, *e.g.*, servidores *Web*, servidores *Proxy*, *logs* do Sistema Operativo (SO), entre outras.

Para a implementação de um sistema SIEM eficaz, é necessário recolher todo este tipo de informação de forma a termos uma visão do estado real da nossa infraestrutura. Para isso é necessário implementar uma infraestrutura dentro da organização para o suporte deste tipo de sistema. Em primeiro lugar é necessário compreender que tipo de informação é e que tipo de conhecimento será possível extrair.

Um evento pode ser definido como uma mudança de estado no sistema. Por predefinição, os serviços e o próprio sistema recorrem à escrita destes eventos para ficheiros (*e.g.*, `/var/log/messages`) para guardar um histórico das alterações que o SO sofreu ao longo do tempo.

Nesta dissertação, serão usados dois tipos de eventos. Alertas de um NIDS - *Snort*[2] - e os eventos gerados por um servidor *Web* - *Apache Http Server*[3].

Estes dois serviços, geram dois tipos de eventos distintos. No primeiro caso, o NIDS gera um novo alerta sempre que deteta uma irregularidade no tráfego de rede e escreve então esse alerta para um ficheiro binário que depois poderá ser lido por um *software* auxiliar. O *Apache Http Server* escreve para ficheiro, em sistemas UNIX, `access.log` e `error.log`.

2 Sistema de deteção de intrusão em rede mais utilizado a nível global - https://www.snort.org
3 Servidor *Web* mais utilizado a nível global - http://httpd.apache.org/

Exemplo de um evento gerado por um NIDS:

```
11/05-22:08:59.705515  [**] [1:469:3] ICMP PING NMAP [**] [Classification:
Attempted Information Leak] [Priority: 2] {ICMP} 192.168.206.129 - 192.168.100.5
```

Exemplo de um evento de acesso num servidor *Web*:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700]
"GET /wine_qb.gif HTTP/1.0" 200 2326
```

Exemplo de um evento de erro num servidor *Web*:

```
[Wed Oct 11 14:32:52 2000] [error] [client 127.0.0.1]
client denied by server configuration: /export/home/live/ap/htdocs/test
```

Como é possível verificar pelos três exemplos anteriores, todos possuem um formato diferente. Existem diversos formatos e padrões de eventos, mas a sua maioria consiste, no mínimo, em duas partes: a etiqueta de tempo (denominada a partir de agora como *timestamp*) e os dados, ou mensagem. O *timestamp* representa a data e hora a que o evento ocorreu, enquanto que os dados representam um conjunto de informações sobre o evento.

Apesar de apenas exemplificarmos estes três tipos de *logs*, por serem os mais importantes para esta dissertação, existem muitos mais, apresentando sempre diferentes formatos e padrões. Esta diferença entre formatos é também ela um desafio para a agregação e análise dos eventos, uma vez que todos estes eventos têm de ser analisados e normalizados.

1.3 ENQUADRAMENTO

A Eurotux Informática S.A., surgiu no ano de 2000, como um projeto de concretização empresarial da investigação desenvolvida na Universidade do Minho pelo grupo de Sistemas Distribuídos do departamento de Informática. No grupo de Sistemas Distribuídos existiu desde sempre uma aposta em *software opensource*, tendo sido pioneiros a nível nacional na adoção de *Linux*, contribuindo ativamente para o projeto na sua fase embrionária. Por acreditar no modelo de desenvolvimento *opensource*, na adoção de *Linux* nas organizações, e por verificar uma falta de oferta especializada nesta área, o grupo de Sistemas Distribuídos do departamento de Informática da Universidade do Minho decidiu criar a Eurotux, contando para isso com a qualidade e espírito empreendedor dos profissionais oriundos da Universidade do Minho.

A Eurotux efetua a gestão de infraestruturas com dimensão relevante para o panorama nacional, servindo várias centenas de *Mbps* de conteúdo para a *Internet*.

Esta dissertação surgiu da necessidade de uma solução SIEM, composta por componentes *opensource* que permita o controlo e análise da infraestrutura atual, sem esquecer o fator de crescimento no volume de dados atual e futuro.

1.4 CONTRIBUTO

Espera-se que com esta dissertação, tenha sido possível demonstrar a possibilidade de implementação de sistemas SIEM com recurso a tecnologias *opensource*, sem nunca deixar o factor de performance de lado. Mostrando ainda, que mesmo em ambientes *Big Data* é possível tirar partido do *opensource*, mantendo todo o controlo da infraestrutura privado e removendo a necessidade de terceirizar este tipo de serviço, garantido assim, ainda mais privacidade e segurança.

1.5 ESTRUTURA DO DOCUMENTO

Além do capítulo introdutório no qual esta secção se insere, o documento é composto por mais quatro capítulos e dois apêndices onde se pode encontrar:

CAPÍTULO 2 Contém o estado da arte referente aos sistemas SIEM disponíveis, atualmente, no mercado, assim como um conjunto de tecnologias, *opensource*, que se esperam capazes de construir um sistema capaz de solucionar os problemas descritos neste capítulo.

CAPÍTULO 3 Apresenta uma abordagem ao problema e como é que as tecnologias apresentadas no Capítulo 2 se relacionam com o modelo proposto.

CAPÍTULO 4 Neste capítulo são discutidas as escolhas de algumas das soluções apresentadas no Capítulo 2, com recurso a avaliações praticas.

CAPÍTULO 5 Conclusão do trabalho realizado no âmbito deste documento e direcções a seguir para um trabalho futuro.

APÊNDICE A Contém instruções necessárias à configuração dos ambientes de teste e das soluções de transporte, para fins de avaliação.

APÊNDICE B Contém instruções necessárias à configuração dos ambientes de teste e das soluções de armazenamento, para fins de avaliação.

2

ESTADO DA ARTE

Nesta secção serão apresentados alguns sistemas SIEM existentes no mercado, assim como componentes, todos eles sobre licença *opensource* que interligados podem levar a criação de um SIEM moderno, capaz de coletar, processar e analisar informação em tempo-real, para ambientes de *Big Data*.

2.1 SIEM

Existem alguns sistemas SIEM no mercado que oferecem características bastante interessantes para assegurar o bom funcionamento da infraestrutura e protegê-la de variados ataques. Mas, o fator *opensource* não é predominante e os preços praticados são bastante elevados.

É necessário compreender que estes sistemas passaram por uma evolução histórica e atualmente estamos já a presenciar o desenvolvimento da terceira geração de sistemas dedicados à segurança da infraestrutura.

2.1.1 *1º Geração: IDS*

Numa primeira geração encontramos um Intrusion Detection System (IDS) nas suas variantes NIDS e HIDS. O NIDS, permite analisar o tráfego de rede e lançar alertas quando algum padrão no tráfego de rede for semelhante a um conjunto de padrões definidos previamente. Um HIDS permite configurar alertas baseados em modificações que ocorrem ao nível do SO, *e.g.*, alterações de ficheiros, sucessivos erros de autenticação, *etc*.

Em resumo, estes sistemas permitem que sejam lançados alertas caso seja detetada alguma variação aos padrões de segurança definidos pela entidade responsável.

2.1.2 *2º Geração: SIEM*

Nesta 2ª Geração, o IDS continuam a fazer parte, mas são acrescentadas algumas características, importantes, para que seja possível um maior controlo e conhecimento da infraestrutura, *e.g.*, monitorização em (quase) tempo-real, painéis de controlo (ou *dashboards*), *etc*.

Será apresentado duas entidades, *AlienVault*[1] e *Loggly*[2], que fornecem o mesmo tipo de serviço mas de diferentes formas e com diferentes características.

A *AlienVault* disponibiliza dois sistemas SIEM, um deles é pago, o Unified Security Management (USM) e o OpenSource SIEM (OSSIM). Sendo este último *opensource*.

O USM está dividido numa arquitetura a 3 camadas. Têm o servidor central, onde agrega e correlaciona as informações reunidas pelos sensores, assim como permite a gestão de alguns aspetos da infraestrutura, geração de relatórios e *dashboards* que mostram estatísticas e dados relevantes.

Os sensores são instalados na infraestrutura de forma a coletarem os *logs* e outras informações relevantes. Estes sensores funcionam também como NIDS eHIDS.

Existe ainda o *Logger*. A função do *Logger* é armazenar todos os eventos para análise histórica e forense, assim como para procura de dados de interesse.

Este USM oferece 3 tipos de licença, em que a mais baixa custa cerca de 3600$, permite apenas distribuições *single-tier* e apenas uma opção de distribuição, através de uma *appliance*[3] virtual. No plano intermédio, a única diferença será nos tipos de distribuição disponíveis, passando a incluir Amazon Machine Image (AMI) e *appliance* física. Outra diferença será também o preço da licença, 17500$. Para ambientes *multi-tier* e distribuídos existe também uma licença, mas o preço é apenas por consulta. Além de serem produtos relativamente caros, não oferecem soluções nos pacotes mais acessíveis, que hoje em dia são cada vez mais importantes, *e.g.*, o suporte a ambientes distribuídos.

O sistema, *opensource*, OSSIM é disponibilizado como caso de estudo para investigadores e estudantes, apesar de algumas empresas o usarem como o seu SIEM. Alguns mecanismos de análise e alerta são mantidos pela comunidade e como é um sistema suportao pela mesma empresa que detém o USM, as informações sobre relatórios de segurança e dados a ser utilizados pelo OSSIM só são lançadas algum tempo depois, criando um risco para a infraestrutura.

O *Loggly*, ao contrário dos anteriores, é um SIEM em *cloud*. Basicamente, os utilizadores instalam um agente nas máquinas que pretendem controlar e os seus dados são enviados para a *cloud*. Este sistema

1 Página *web*: `http://www.alienvault.com`
2 Página *web*: `https://www.loggly.com`
3 Equipamento projectado e criado para fornecer um recurso tecnológico específico

processa a informação, armazena-a, gera gráficos com as principais informações e gera alertas caso necessário. É também possível filtrar e procurar informação, assim como a criação de *dashboards* personalizáveis. Como veremos mais à frente, é possível criar uma solução em infraestrutura privada, com o mesmo nível de detalhe e informação. O que é bom neste tipo de SIEM, é que o cliente não tem a necessidade de criar uma infraestrutura ou expandir a atual, para configurar o sistema, pois a única coisa que tem a fazer é instalar os agentes nas máquinas que pretende monitorizar. Um entrave a utilização deste tipo de sistema poderá estar no próprio facto de se deixar de ter controlo sobre essa informação, uma vez que esta passa a ser processada por terceiros. Outro entrave poderá ser o preço. O *Loggly* oferece um sistema de planos em que incluí um plano grátis, mas onde é apenas permitido o envio de 200*MB/dia* e 7 dias de arquivo, algo impensável para o tipo de infraestruturas que se pretende tratar. Olhando para planos um pouco mais realistas, 35*GB/diários* e um arquivo de 90 dias ficaria por 4106*$/mês*, que é um valor impensável para muitas das empresas suportarem. Além disso, 35*GB/diários* é um valor muito baixo para um volume de dados, quando falamos em ambientes *Big Data*. Quando subimos o plano para os 150*GB/diários* e 30 dias de arquivo, o valor dispara para os 11100*$/mês*.

2.1.3 *3ª Geração: Next-Gen SIEM*

Esta terceira geração ou *next-gen* SIEM como são denominados, tiram partido da introdução de novas técnicas e ferramentas de *Big Data*, assim como do seu rápido desenvolvimento. Com estas ferramentas passa a ser possível um sistema capaz de informar sobre o estado da infraestrutura de uma forma mais assertiva, assim como coletar um grande número de dados e processá-los em (quase) tempo-real, de uma forma que até agora seria quase impraticável. Num futuro muito próximo, poderemos presenciar sistemas capazes de prever períodos onde existirá uma maior probabilidade de ataques, através de mecanismos de *machine learning* entre outras técnicas, permitindo canalizar recursos no reforço de determinados pontos estratégicos.

2.2 TRANSPORTE

Para uma gestão centralizada de toda a infraestrutura é necessário ter acesso à informação gerada por todas as máquinas que dela fazem parte. Para que isso aconteça, são necessários mecanismos que permitam o transporte, idealmente seguro, dessa informação para um local central, para ser depois processada e armazenada. Para este efeito existem diversos mecanismos disponíveis e alguns criados com o propósito de agregação centralizada deste tipo de informação. Pretende-se ainda que estes

mecanismos afetem o mínimo possível as máquinas onde estão configuradas, deixando os recursos livres para o verdadeiro intuito para o qual a máquina foi configurada.

2.2.1 *Rsyslog*

Um dos mecanismos mais conhecidos para o transporte deste tipo de informação é o *Rsyslog*[4]. O *Rsyslog* é um *software* usado em sistemas baseados em UNIX para o reencaminhamento de mensagens sobre uma rede Internet Protocol (IP). É um sistema baseado no protocolo *syslog*[5] e permite, *e.g.*, a filtragem baseada em conteúdo, *i.e.*, permite realizar algum processamento básico sobre a informação a transportar. Além de permitir o armazenamento da informação em ficheiros, permite também armazená-los em bases de dados como *MySQL*[6], por exemplo.

Apresenta alguns mecanismos de tolerância a faltas - "R" em *Rsyslog* representa *Reliable*[7]). Utiliza Transmission Control Protocol (TCP) como protocolo de transmissão. Se a conexão entre o cliente e o servidor falhar, este sistema pode ser configurado para guardar a informação em ficheiros *spool*[8] até a conexão ser restabelecida e a informação ser reenviada para o servidor principal. Além disso, é possível também configurar mecanismos de *backup*. Suporta ainda Transport Layer Security (TLS), *IPv6* e múltiplos formatos de *input* e *output*.

Este mecanismo tornou-se a solução de *logging*, por predefinição, de muitas distribuições UNIX atuais.

2.2.2 *Logstash*

O *Logstash*[9] é uma ferramenta criada para gerir eventos e pode ser usada para o transporte e processamento básico. Esta ferramenta possuí uma quantidade considerável de *plugins* que lhe permite ser bastante flexível.

4 Página *web*: `http://www.rsyslog.com`.
5 Protocolo desenvolvido em 1980 por *Eric Allman* para ser usado pela aplicação *sendmail*, como forma de anexar conteúdo de certas aplicações, a ficheiros. A fonte enviava uma mensagem com um máximo de 1*KB* para o destinatário sobre User Datagram Protocol (UDP).
6 Página *web*: `http://www.mysql.com/`.
7 Do Inglês *Fiável*.
8 Ficheiros onde são armazenados os dados até serem processados.
9 Página *web*: `http://logstash.net`.

Este sistema começou como um projecto *opensource* desenvolvido e suportado por uma pequena comunidade, mas devido as suas características foi adquirido pela *Elastic* (ver Secção 2.3.1), permitindo um desenvolvimento mais rápido e disponibilizando suporte aos seus utilizadores.

Como foi dito anteriormente, possuí um grande número de *plugins* e é possível criar novos, se necessário. Através dos *plugins* é possível processar informação de diversos *inputs* (*e.g.*, ficheiros, *stdin*, *irc*, *log4j*, *etc.*), retirar métricas (*e.g.*, contagem de códigos de erro HyperText Transfer Protocol (HTTP) de um servidor *Web*), enviar para ferramentas que permitam a visualização dos dados (*e.g.*, *Kibana*[10], *Graphite*[11], *etc.*) ou simplesmente enviar para armazenamento.

Um dos componentes interessantes do *Logstash* é o *Grok*[12]. O *Grok* é um *software* que permite a análise de ficheiros, neste caso, permite transformar os dados não estruturados de um ficheiro *log* em informação estruturada e normalizada para mais fácil acesso. Isto tudo sem ser necessário escrever linhas e linhas de expressões regulares.

O *Logstash* é desenvolvido em *JRuby* e é de fácil distribuição pois é executado sobre Java Virtual Machine (JVM), logo, pode ser configurado em sistemas baseados em UNIX e Windows. Este também pode ser apontado como um dos seus maiores problemas, porque um dos principais objetivos é o impacto mínimo dos agentes nas máquinas a controlar e, uma JVM pode necessitar de recursos, nomeadamente memória, em excesso. Para solucionar este problema, existe o *Lumberjack*[13], também designado por *Logstash-Forwarder*, um projecto baseado em *Go*, com menor impacto em CPU e memória. Ele permite agregar e encaminhar a informação para o servidor de central. Um dos requisitos para a configuração do *Lumberjack* são certificados Secure Sockets Layer (SSL), para a comunicação cifrada entre este e o *Logstash*.

Outro problema é não estar preparado para tolerância a faltas. É sugerido pela equipa de desenvolvimento, a utilização do *Logstash* em conjunto com o *Redis*[14], uma *key-value store*, para servir como mecanismo de *buffering*. O problema é que esta solução só é possível do lado do servidor, uma vez que implementar este tipo de solução nos agentes instalados nas máquinas é algo que não seria aceitável a nível de performance e impacto no sistema.

10 Página *web*: `http://www.elasticsearch.org/overview/kibana`
11 Página *web*: `http://graphite.wikidot.com`
12 Página *web*: `https://code.google.com/p/semicomplete/wiki/Grok`.
13 Página *web*: `https://github.com/elasticsearch/logstash-forwarder`
14 Página *web*: `http://redis.io/`.

2.2.3 *Fluentd*

À semelhança do *Logstash*, existe o *Fluentd*[15]. O *Fluentd* foi desenhado como um coletor de dados para *streams* de grandes volumes de dados. O *Fluentd* apresenta também uma grande quantidade de *plugins* que nos permite processar dados de diferentes *inputs*, fazer algum processamento básico e armazená-los em ficheiros ou enviar para *Elasticsearch*, HDFS, *MySQL*, *etc*.

Este sistema faz parte da *Treasure Data*[16] com a denominação de *Treasure Agent*. Nesta versão, comercial, a *Treasure Data* disponibiliza suporte comercial, monitorização e alarmísticas adicionais, assim como *plugins* adicionais.

Ao contrário do *Logstash*, o *Fluentd* possuí já alguns mecanismos de tolerância a faltas. Permite, *e.g.*, a utilização de ficheiros de *spool* ou da memória como mecanismos de *buffering*, até reconectar com um dos servidores. Outra característica extremamente importante do *Fluentd* é a alta disponibilidade, *i.e.*, é possível configurar dois ou mais servidores e caso algum destes falhe, os agentes iram detectar a falhe e passar a enviar os dados para o servidor seguinte.

O *Fluentd* foi escrito em *CRuby* e tem como requisito o *Ruby* 1.9.2 ou superior. A sua distribuição é feita à base de binários para sistemas UNIX (*e.g.*, RedHat Package Manager (RPM) e Debian Binary Package (DEB)), mas começam já a surgir os primeiros avanços para sistemas Windows, com um *Fluentd-Forwarder* à semelhança do *Logstash-Forwarder*, excrito em *Go* e que permite fazer o encaminhamento dos dados para um servidor *Fluentd*.

2.2.4 *Apache Kafka*

Quando se fala em sistemas de transporte, não é possível deixar de falar em sistemas de mensagens distribuídos. Como sistemas capazes de transportar grandes volumes de dados, estes sistemas são usados, *e.g.*, no mercado bolseiro e de apostas, onde existe uma necessidade por mecanismos de performance elevada e altamente tolerantes a faltas. Se um dos objetivos desta dissertação passa pela eficiência no transporte de eventos e *logs* em ambientes *Big Data*, é impossível deixar de fora este tipo de sistema.

O *Apache Kafka*[17] é um sistema de mensagens distribuído, desenhado para alta performance, implementado em *Scala*[18] e algum *Java*. Foi criado pelo *LinkedIn*[19] e depois disponibilizado publicamente,

15 Página *web*: `http://fluentd.org`.
16 Página *web*: `http://www.treasuredata.com`.
17 Página *web*: `https://kafka.apache.org`
18 Página *web*: `http://www.scala-lang.org`.
19 Página *web*: `https://www.linkedin.com`.

através de licença *opensource*, em 2011. A *LinkedIn* focou-se na criação de um sistema de alto desempenho para suportar fluxos de grandes volumes de dados, assim como o suporte à distribuição e processamento desses fluxos em (quase) tempo-real.

Possuí uma arquitetura pensada para *cluster*, onde cada máquina é denominada de *broker*. Nesta arquitetura existem dois tipos de actores, os produtores e os consumidores. Os produtores são os clientes que escrevem os dados para os *brokers*, enquanto que os consumidores lêem os dados dos *brokers*. Os dados são armazenados em tópicos e estes são divididos em partições, que são por sua vez, replicadas. Desta forma previne-se a perda de dados. Uma vez que existe um ambiente distribuído, existe a necessidade de um sistema que faça a coordenação do *cluster*. Esse sistema é o *Apache Zookeeper*[20].

Na versão à data de publicação deste documento, *Apache Kafka* 0.8.1.1, o *Zookeeper* é necessário para que os *brokers* armazenem uma informação do estado geral do *cluster* e para eleger um novo líder, caso seja detetada alguma falha no líder atual. É também necessário pelos consumidores para manter *offsets* das mensagens. A partir da versão 0.9, o *Zookeeper* deixará de ser utilizado pelos consumidores, baixando assim a carga no sistema de coordenação. Os consumidores passarão a utilizar tópicos especiais no próprio *Kafka*. O *Zookeeper*, além de correr sobre JVM, necessita também de uma infraestrutura ou de estar configurado nas mesmas máquinas que o *Kafka*, o que poderá representar por si só um problema, caso estejamos perante uma infraestrutura de pequenas dimensões ou sujeitos a baixos orçamentos para implementação.

Até a versão *0.8.x*, ainda não existem mecanismos implementados que permitam, *e.g.*, cifrar os dados enquanto estes são transmitidos entre os produtores e os consumidores. Desde Julho de 2014 que mais funcionalidades sobre segurança estão a ser discutidas e prevê-se que futuras versões incorporem já estas funcionalidades.

Para passar uma noção de uma infraestrutura que utiliza *Kafka* como mecanismo de transporte, ficam aqui os valores referentes à implementação do mesmo pela *LinkedIn* em 2014:

- +300 *brokers*
- +18000 tópicos
- +140000 partições
- 200000 milhões de messagens/dia
- 40*TB* de tráfego de entrada

20 Sistema de gestão e coordenação de aplicações distribuídas em ambientes *cluster* - Página *web*: `http://zookeeper.apache.org`.

- 160*TB* de tráfego de saída

Durante um pico de carga, pode chegar aos seguintes valores:

- 3,25 milhões de mensagens/segundo
- 5,5*GB*/segundo de tráfego de entrada
- 18*GB*/segundo de tráfego de saída

2.3 ARMAZENAMENTO

O armazenamento de dados é visto cada vez mais como um desafio, tanto pela grande quantidade de informação que é gerada diariamente, como pela sua distribuição espacial e temporal -*Big Data*.

O armazenamento é de extrema importância e é necessário que seja um sistema, principalmente, tolerante a faltas e que demonstre uma boa performance. Quando se fala em boa performance para este tipo de sistemas, é necessário distinguir entre uma boa performance em leituras aleatórias e no armazenamento de pequenos ficheiros ou o oposto, um sistema otimizado para armazenar grandes quantidades de dados, distribuídas por ficheiros de tamanho considerável e que demonstre uma boa performance em escrita e leituras sequenciais. Devido a estes fatores, torna-se cada vez mais importante analisar todas as ferramentas disponíveis, determinar as suas características e tentar adequá-las o melhor possível às necessidades.

Até agora as empresas optavam por sistemas de armazenamento de dados estruturados, que passavam principalmente por bases de dados relacionais, Structured Query Language (SQL). Mas, estes sistemas sentem já alguma dificuldade em responder às necessidades de hoje e as empresas tendem cada vez mais a procurar soluções de armazenamento de dados não estruturados, Non-Structured Query Language (NoSQL)). Um dos grandes problemas neste tipo de sistemas de armazenamento é a interrogação dos dados e, uma vez que na sua maioria, estes sistemas não a possuem como característica nativa, passa então a ser necessário mecanismos que permitam a indexação dos dados para futuras pesquisas.

2.3.1 *Indexação*

Sendo os dados armazenados de forma não estruturada, é necessário mecanismos que permitam criar indíces para ser mais fácil encontrar e pesquisar sobre a informação. Sem estes mecanismos, seria

necessário percorrer todos os dados até encontrar os valores pelos quais é demonstrado interesse. Isto é inviável para sistemas SIEM, em que estamos interessados em encontrar padrões e informações num curto espaço de tempo, para reagir rapidamente e manter a segurança da infraestrutura.

Para um sistema SIEM, existem intervalos de tempo em que é mais importante o acesso rápido à informação em volta de um determinado evento. Normalmente, estes intervalos de tempo são dias ou semanas e é neste intervalo que devemos manter todos os dados indexados. A partir daqui podemos arquivar a informação e caso seja necessário, processá-la novamente.

Assim surge o conceito de armazenamento a curto e longo-prazo. No armazenamento a curto-prazo, mantém-se a informação indexada para acesso rápido. A partir daqui entra-se no armazenamento a longo-prazo (ver Secção 2.3.2), onde a informação é arquivada e depois é possível processá-la, periodicamente, para retirar dados estatísticos ou realizar análise histórica para determinar se a infraestrutura foi compremetida por alguma vulnerabilidade divulgada recentemente.

De seguida serão apresentadas algumas relacionadas com a indexação: *Apache Lucene*, *Apache SOLR* e *ElasticSearch*.

Apache Lucene

A melhor definição do *Apache Lucene* é dada na página *web* do projecto[21]:

> *"Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform."*

Basicamente, o *Lucene*, é um motor de pesquisa, *i.e.*, é um *software* que constrói um índice sobre um texto, e apenas sobre texto, e responde a interrogações usando os índices construídos previamente.

Este motor de pesquisa foi desenvolvido por *Doug Cutting* em 1999 e passou a ser um projeto da fundação *Apache* em 2001.

O núcleo lógico do *Lucene* assenta sobre a ideia de um documento que contém palavras, isto permite que a Application Programming Interface (API) seja independente do formato dos dados. Assim, passa a ser possível indexar diferentes fontes de dados, como correio eletrónico, Portable Document Format (PDF), Documentos *Microsoft Word*[22], páginas *web*, bases de dados, *etc.*, desde que seja possível extrair informação textual destas fontes.

21 Página *web*: `https://lucene.apache.org`.
22 Processador de texto da *Microsoft*

Um documento consiste num ou mais campos do tipo `<key, value>` e, um ou mais documentos formam um índice. A indexação de um documento passa por 3 fases:

1. Criação do documento
2. Adicionar o documento ao índice
3. Análise do documento

A análise do documento é executada através de componentes especializados que fazem a análise e se necessário, transformam os dados. Estes componentes controlam a forma como o texto é partido em termos, que são depois usados para indexar corretamente o documento. Estes componentes podem ser usados para remover *stop words*[23] ou executar *stemming*[24].

A pesquisa requer que pelo menos um índice já tenha sido criado. Depois desta condição estar satisfeita é necessário interrogar o sistema através da sua API. Então o *Lucene* irá percorrer os índices e procurar a informação usando o mesmo componente especializado que usou para analisar os documentos durante o processo de indexação. Após isto, se forem encontrados resultados, estes serão extraídos dos índices e apresentados ao utilizador.

A API do *Lucene* é bastante utilizada como base para outros motores de pesquisa, o *Apache Solr* e *ElasticSearch* são exemplos disso.

Apache Solr

O *Apache Solr*[25] é uma plataforma *opensource*, que entre as suas principais características, permite pesquisa *full-text*, indexamento em (quase) tempo-real, alta fiabilidade, escalabilidade e tolerância a faltas. Permite a replicação automática de índices assim como *failover* e recuperação automática. O *Solr* está otimizando para suportar um grande volume de tráfego *web* e suporta algumas das maiores páginas *web* a nível global, como a *Netflix*, *CNET* e *AOL*.

Criado por *Yonik Seeley*, em 2004, para dar a *CNET Networks*[26] a capacidade de pesquisa dentro da sua própria página *web*. Em 2007, a *CNET Networks* divulgou publicamente o *Solr*, doando-o à *Apache Foundation*. Desde 2010, que o *Solr* e o *Lucene* são desenvolvidos pela mesma equipa, mas continuam a ser distribuídos de forma independente. Isto permite que outras entidades, como é o caso da *Elastic* (ver Secção 2.3.1), desenvolvam produtos com base no *Lucene*.

23 Conjunto de palavras que são filtradas antes do processamento de dados de linguagens naturais, ou textos. Exemplos de *stop words* são: *de, o, a, com, ele, não, seu, etc*.
24 *Stemming* é o ato de reduzir uma palavra às suas origens, *e.g.*, redução de *gatos* e/ou *gatinhos* para *gato*.
25 Página *web*: `https://lucene.apache.org/solr`.
26 Rede de imprensa Americana, dedicada a tecnologia. Em 2008 foi comprada pela *CBS Interactive*.

Escrito em *Java* e executado por um *servlet*, como *Apache Tomcat* ou *Jetty*, permite que seja executado em vários *SO*, desde que estes estejam configurados para executar aplicações em *Java*, *i.e.*, possuam JVM. No núcleo do *Solr* podemos encontrar o *Lucene*, como foi referido na Secção 2.3.1, que permite a indexação e pesquisa. Possuí uma API Representational State Transfer (REST) recorrendo a documentos eXtensible Markup Language (XML) e JavaScript Object Notation (JSON) que lhe permite ser facilmente utilizada nas mais variadas linguagens de programação.

De seguida, serão dados alguns exemplos de como é possível utilizar a API do *Solr* para indexar ou pesquisar. Para começar, a indexação de um evento gerado por um IDS, recorrendo a API REST para JSON e ao utilitário *cURL*[27]:

```
$ curl 'http://<server IP>:8983/solr/update/json?commit=true' \
           --data-binary @ids_alert.json \
           -H 'Content-type:application/json'
```

Onde o conteúdo do ficheiro `ids_alert.json` é o seguinte:

```
[
  {
    "timestamp": "2014-11-15T23:32:11Z",
    "event_type": "alert",
    "src_ip": "10.16.1.11",
    "src_port": 49719,
    "dest_ip": "192.168.88.3",
    "dest_port": 443,
    "proto": "TCP",
    "alert_action": "allowed",
    "alert_gid": 1,
    "alert_signature_id": 30524,
    "alert_rev": 1,
    "alert_signature": "SERVER-OTHER OpenSSL TLSv1.1
                        heartbeat read overrun attempt",
    "alert_category": "Attempted Information Leak",
    "alert_severity": 2
  }
]
```

Como podem notar pelo documento *JSON* acima apresentado, não existe nenhum aninhamento, isto porque o *Solr* não aceita estruturas aninhadas.

Agora já é possível pesquisar, *e.g.*, executando o seguinte comando:

```
$ curl 'http://<server IP>:8983/solr/select?q=proto:TCP&wt=json&indent=true'
```

27 Biblioteca e ferramenta de linha de comandos que permite a transferência de dados através de vários protocolos, como HTTP e File Transfer Protocol (FTP). Página *web*: `http://curl.haxx.se`.

Como resposta ao pedido anterior, será devolvido o seguinte:


```
{
  "responseHeader":{
    "status":0,
    "QTime":2,
    "params":{
      "indent":"true",
      "wt":"json",
      "q":"proto:TCP"}},
    "response":{"numFound":1,"start":0,"docs":[
      {
        "timestamp": "2014-11-15T23:32:11Z",
        "event_type": "alert",
        "src_ip": "10.16.1.11",
        "src_port": 49719,
        "dest_ip": "192.168.88.3",
        "dest_port": 443,
        "proto": "TCP",
        "alert_action": "allowed",
        "alert_gid": 1,
        "alert_signature_id": 30524,
        "alert_rev": 1,
        "alert_signature": "SERVER-OTHER OpenSSL TLSv1.1
                            heartbeat read overrun attempt",
        "alert_category": "Attempted Information Leak",
        "alert_severity": 2}]
}}
```


Algo que é necessário ter em atenção é que antes de estes documentos serem indexados é necessário criar o esquema dos documentos e dar a conhecer ao *Solr*. Além disso e ao contrário da ferramenta que falaremos a seguir, é necessário realizar algumas alterações na configuração para que seja possível indexar documentos sem que seja necessário controlar os identificadores dos documentos, sendo estes gerados e controlados pelo próprio *Solr*. Estas configurações e alterações pode ser vistas no Apêndice B.2.

ElasticSearch

O *Elasticsearch*[28] é uma solução que permite a indexação e pesquisa de informação em (quase) tempo-real. É um projeto *opensource* e foi desenhado de raiz para ser usado em ambientes distribuídos onde a escalabilidade e fiabilidade são um requisito. Possui clientes em diversas linguagens, assim como

28 Página *web*: http://www.elasticsearch.org.

uma API e interrogações recorrendo a uma Domain-Specific Language (DSL), além de permitir a pesquisa *full-text* simples.

Criado por *Shay Banon* e tendo a sua primeira versão lançada em Fevereiro de 2010. *Shay Banon* criou a *Compass*[29] em 2004, mas percebeu, algum tempo mais tarde, que iria necessitar de reescrever grande parte do *software* para criar uma solução de pesquisa escalável. Dessa necessidade surgiu o *ElasticSearch*.

Uma vez que a solução está escrita em *Java* e corre sobre uma JVM, é possível correr a solução em ambientes *UNIX* e/ou *Windows*.

Como na solução anterior, no *ElasticSearch* podemos também encontrar o *Lucene* (ver Secção 2.3.1), que permite a indexação e pesquisa. Sobre o *Lucene* existe uma API *JAVA* e uma API REST com recurso a JSON.

De seguida, serão dados alguns exemplos de como é possível utilizar a API do *ElasticSearch* para indexar e/ou pesquisar. Para começar, a indexação de um evento gerado por um IDS, recorrendo à API REST e ao utilitário *cURL*:

```
$ curl -XPOST 'http://<server IP>:9200/ids/alert?pretty=true' \
        -d @ids_alert.json
```

Ao executar um pedido *POST* em vez de um pedido *PUT*, será gerado um *id* automaticamente.

O conteúdo do ficheiro `ids_alert.json` é o seguinte:

```
{
  "timestamp": "2014-11-15T23:32:11Z",
  "event_type": "alert",
  "src_ip": "10.16.1.11",
  "src_port": 49719,
  "dest_ip": "192.168.88.3",
  "dest_port": 443,
  "proto": "TCP",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 30524,
    "rev": 1,
    "signature": "SERVER-OTHER OpenSSL TLSv1.1
                  heartbeat read overrun attempt",
    "category": "Attempted Information Leak",
    "severity": 2
```

29 Motor de pesquisa em *Java* construído sobre *Apache Lucene*.

```
  }
}
```

Ao contrário do documento apresentado para o *Solr*, este já contém aninhamento na sua estrutura e essa é uma característica nativa do *ElasticSearch*.

Com a execução do pedido *POST* acima, vamos obter a seguinte resposta, se não existir qualquer problema com a indexação:

```
{
  "_index" : "ids",
  "_type" : "alert",
  "_id" : "AUwCvq2BZLNwk9TkUBUe",
  "_version" : 1,
  "created" : true
}
```

Uma vez que já existe um documento indexado, já é possível realizar uma pesquisa para demonstrar a *API* de pesquisa. Fazendo novamente uso do *cURL*, executa-se o seguinte comando:

```
$ curl -XGET 'http://localhost:9200/ids/_search?q=*:*&pretty=true'
```

Obtendo como resposta:

```
{
  "took" : 95,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "ids",
      "_type" : "alert",
      "_id" : "AUwCvq2BZLNwk9TkUBUe",
      "_score" : 1.0,
      "_source":{
        "timestamp": "2014-11-15T23:32:11Z",
        "event_type": "alert",
        "src_ip": "10.16.1.11",
        "src_port": 49719,
        "dest_ip": "192.168.88.3",
```

```
        "dest_port": 443,
        "proto": "TCP",
        "alert": {
          "action": "allowed",
          "gid": 1,
          "signature_id": 30524,
          "rev": 1,
          "signature": "SERVER-OTHER OpenSSL TLSv1.1
                        heartbeat read overrun attempt",
          "category": "Attempted Information Leak",
          "severity": 2
        }
      }
    } ]
  }
}
```

Ao contrário do *Solr*, no *ElasticSearch* não é necessário apresentar o esquema dos documentos a indexar, uma vez que ele o detecta automaticamente. Mas, para um maior controlo dos dados indexados e os seus tipos, recomenda-se a criação de um esquema e a sua configuração no *ElasticSearch*.

2.3.2 *Arquivo*

Quando se fala em *cloud* e *Big Data*, um dos nomes que surgem em mais artigos e notícias é o *Apache Hadoop*. O *Hadoop*[30] é uma *framework* para processamento distribuído e particionamento de grandes volumes de dados.

Foi criado pela *Yahoo!* em 2005, mas atualmente surge como um projeto da comunidade *Apache* e vem sendo adotado por inúmeras entidades, quando necessitam de tratar volumes massivos de dados. Atualmente existe já um ecossistema de tecnologias que foram desenvolvidas em volta dos dois principais projetos do *Hadoop*, o HDFS e o *Hadoop MapReduce*. Nesta secção iremos dar principal foco ao HDFS.

O HDFS é um sistema de ficheiros distribuído e tolerante a faltas. Foi desenhado para suportar grandes quantidades de dados, assim como otimizado para ficheiros de grandes dimensões e para operações de escrita/leitura sequenciais. Possui uma arquitetura *master/slave*, sendo o *master* denominado de *NameNode* e os *slaves* de *DataNodes*. O *NameNode* é responsável pela coordenação do particionamento/distribuição dos dados e pelo armazenamento dos metadados, enquanto que os *DataNodes* são responsáveis pelo armazenamento dos dados em si.

30 Página *web*: `http://hadoop.apache.org`.

O sistema de ficheiros está estruturado em blocos e um ficheiro é dividido em blocos de igual tamanho que são distribuídos por vários *DataNodes* do *cluster*. Tipicamente, o sistema assume 64*MB* ou 128*MB*, como tamanho padrão para os blocos. Daqui percebe-se que este sistema não foi desenvolvido com o objetivo de armazenar ou processar ficheiros de pequenas dimensões.

Para garantir a disponibilidade dos dados, cada bloco é replicado 3 vezes, por predefinição, e cada uma destas réplicas é colocada em *DataNodes* diferentes, assim garante-se que a falha de um dos *DataNodes* não implica a perda dos dados e inacessibilidade. Se o *NameNode* detetar que um bloco não cumpre o fator de replicação, escalona uma operação de replicação de um dos blocos para um dos *DataNodes* ativos.

No caso dos *NameNodes* é diferente, pois este é único e caso falhe, perde-se o acesso ao *cluster* (Single Point of Failure (SPOF)). A partir da versão 2.2.0, versão à data de escrita deste documento, o caso é diferente. Agora é possível a coexistência de dois *NameNodes*, em estado ativo e passivo, *i.e.*, um deles assume o estado passivo considerado *Hot Standby*, e caso o *NameNode* ativo falhe, realiza-se a transição do estado passivo para ativo e assim diminui-se consideravelmente o tempo em que o *cluster* está inacessível.

Uma das grandes vantagens do HDFS é ser escalável horizontalmente e ter a possibilidade de ser configurado em servidores de médio e baixo custo. Além disso integra idealmente com *Hadoop MapReduce*, permitindo que os dados sejam lidos e computados localmente, *i.e.*, nas máquinas onde estes estão armazenados.

No Capítulo 4.2.1 é possível observar alguns resultados sobre a performance e tolerância a faltas do HDFS.

2.4 PROCESSAMENTO

O processamento de dados é uma das etapas mais complexas desta dissertação, uma vez que o objetivo é que esse processamento seja suportado, quase por completo, do lado do servidor e não nos agentes, onde os dados são colectados.

Existem dois tipos de processamento essenciais para este tipo de sistemas, processamento em (quase) tempo-real e processamento em *batch*. Neste caso, o processamento em *batch* tem como principal objetivo o processamento de todos os dados guardados em arquivo de modo a retirar dados estatísticos assim como análise de padrões de tráfego, usabilidade, entre outros, de forma a criar limites e regras mais eficazes de controlo de segurança, assim como deteção de vulnerabilidades divulgadas recentemente.

No caso do processamento em (quase) tempo-real, é mais complexo, pois representa o processamento de grandes volumes de dados a serem lidos, formatados, normalizados, analisados e acima de tudo e mais importante correlacionados, tudo isto em (quase) tempo-real útil. Alguns mecanismos de transporte (ver Secção 2.2), permitem realizar algum processamento básico, como formatação, normalização e filtragem, mas como um dos objetivos é diminuir o impacto do processamento nas máquinas a controlar, isso não será uma hipótese. Contudo, poderá ser utilizado para modificações básicas, *e.g.*, a alteração de valores que podem vir a prejudicar a análise dos mesmos, como endereços IP, data do evento, *etc*.

2.4.1 *Batch*

O *Hadoop MapReduce* é um dos módulos do projeto *Hadoop* e permite o processamento em paralelo de grandes volumes de dados.

Em primeiro lugar, o *MapReduce* é um modelo de programação desenhado para o processamento de grandes volumes de dados em paralelo, dividindo a tarefa inicial num conjunto de tarefas mais pequenas. Inicialmente, uma função de mapeamento divide a tarefa num conjunto de tarefas mais pequenas e estas são executadas, gerando cada uma, um resultado. Depois, uma função de redução irá pegar em todos os resultados e processá-los de formar a retornar o resultado final.

O *Hadoop MapReduce* integra com o HDFS e permite processar os dados localmente, *i.e.*, usando o poder de computação de cada um dos *DataNodes*. É atráves do *Hadoop YARN*, uma *framework* para escalonamento de tarefas e gestão de recursos em *clusters*, que o *Hadoop MapReduce* é executado e as suas tarefas são escalonadas e distribuídas.

Latências difíceis de melhorar, relacionadas com operações (cópia, ordenação e iteração), tamanho do *input* e número de processadores disponíveis, fazem destas algumas das razões pelo qual o *Hadoop MapReduce* é tido sobretudo como um mecanismo de processamento em *batch* e, possivelmente o ideal para o tipo de análises históricas que se pretende para um armazenamento a longo-prazo.

2.4.2 *Stream*

Para o processamento em (quase) tempo-real, surge outro conceito, computação distribuída de fluxos de dados. Muitas empresas começaram a ter necessidade de realizar processamento em (quase) tempo-real de grandes volumes de dados, tanto para serem usados em sistemas de monitorização, em *machine learning*, computação contínua, *etc*. Empresas como a *Yahoo!* avançaram com a investigação

de soluções capazes de suportar as suas próprias necessidades. Das investigações realizadas por diversas entidades, resultaram algumas plataformas, *e.g.*, o *Storm* da *BackType* e o *S4* da *Yahoo!*, sendo este último entregue à fundação *Apache*, assim como tinha acontecido anteriormente com o *Hadoop*. Além das duas plataformas acima referidas, existe mais uma, designada por *Apache Spark* que foi desenvolvida na Universidade da Califórnia, *Berkeley*.

Apache Storm

O *Storm*[31] foi criado pela *BackType*, que mais tarde foi adquirida pela *Twitter*. A *Twitter* tomou então a iniciativa de tornar o *Storm opensource*. Muitas análises *online* comparam o *Storm* com o *Hadoop*, dizendo que o *Storm* fez pelo processamento em (quase) tempo-real o mesmo que o *Hadoop* fez pelo processamento em *batch*.

Este sistema foi desenhado para operar, não com dados estáticos, mas sim com fluxos de dados que se esperam contínuos. A utilização de *ZeroMQ* como sistema de mensagens, permite que as mensagens fluam diretamente entre tarefas, removendo intermediários. Existe um grande foco na tolerância a faltas e na gestão do *cluster*. O *Storm* transforma a informação em tuplos para tratamento interno e se algum tuplo for descoberto sem ter sido processado é automaticamente reenviado pelo *Spout*[32]. Se uma tarefa falhar, o *Storm* deteta e as mensagens são reencaminhadas para recomeçar rapidamente o seu processamento.

Outra característica importante, é o facto de correr quase qualquer tipo de linguagem de programação. Por defeito, já suporta *Clojure*, *Java*, *Ruby* e *Python*, mas é possível adicionar suporte para outras linguagens através do protocolo de comunicação do *Storm*.

Apache S4

O *S4*[33] é também, à semelhança do *Storm*, uma plataforma que permite o processamento em (quase) tempo-real de fluxos de dados contínuos. O *S4* foi desenvolvido pela *Yahoo!* em 2010, como foi dito anteriormente. Atualmente é um projeto da fundação *Apache* e é ainda usado nos servidores de produção da *Yahoo!*, onde processa centenas de interrogações por segundo.

É um sistema descentralizado e sem nenhum SPOF. É escalável horizontalmente, não existindo um limite predefinido no número de máquinas suportadas. O *S4* utiliza ainda outro projeto da fundação *Apache*, o *ZooKeeper*.

31 Página *web*: `http://storm-project.net`.
32 Fonte de entrada do fluxo de dados no *cluster Storm*.
33 Página *web*: `http://incubator.apache.org/s4`.

Apache Spark

O *Apache Spark* foi originalmente desenvolvido por *Matei Zaharia* no *AMPLab*[34], na Universidade da Califórnia, *Berkeley*, em 2009. O seu código foi aberto em 2010 e em 2011 doado à fundação *Apache*, que em 2014 tornou-o um projecto de alto nível, da sua fundação.

Na realidade, o *Spark* é uma *framework* que permite o processamento em *batch*, iterativo e em (quase) tempo-real. Já demonstrou em diversos casos ser mais eficiente do que o próprio *MapReduce*, nomeadamente, batendo o recorde mundial de ordenamento em larga escala, em 2014. Desta *framework*, fazem parte o *Spark Streaming* (responsável pelo processamento em (quase) tempo-real), *SparkSQL*, *MLLib* e *GraphX*.

Para executar o *Spark* numa infraestrutura é necessário um gestor de *cluster* e um sistema de armazenamento de dados distribuído. Para o primeiro, o *Spark* possuí uma ferramenta nativa, o *Spark Cluster*, mas também pode recorrer a outros gestores, *e.g.*, *Hadoop YARN* e *Apache Mesos*[35]. Para o sistema de armazenamento distribuído, é possível integrar o *Spark* com HDFS, *Apache Cassandra* ou outras fontes de dados do *Hadoop*, como o *Hive*[36] ou *HBase*[37].

O *Spark* foi desenvolvido na sua maioria em *Scala*, mas possui integração nativa com *Java*, *Python* e *R*, o que permite um nível de abstração muito elevado.

2.5 VISUALIZAÇÃO

Devido a quantidade de dados gerada atualmente, e futuramente, torna-se incomportável a análise textual de toda essa informação, recorrendo a diversas ferramentas, *e.g.*, *cat*[38], *tail*[39], *grep*[40], *etc*. Sobre este assunto, *Richie S. King* escreveu no seu livro *Visual Storytelling with D3* [4], o seguinte:

> *"Data, data, data. Piles and gob of it are amassing everywhere faster than ever before. And with this grand swelling of information has come a sudden rise in the need for a*

34 *Algorithms, Machines and People Lab.* - Laboratório com principal foco de investigação em Machine Learning (ML), *data mining*, bases de dados, *information retrieval*, processamento da linguagem natural e reconhecimento de voz. Mais informações em *https://amplab.cs.berkeley.edu*

35 *Framework* que permite a gestão de *clusters* e foi desenvolvida pela Universidade da Califórnia, *Berkeley*.

36 *Software* de *Data Warehouse* construído sobre o *Apache Hadoop*. Página *web*: `https://hive.apache.org`.

37 Base de dados não relacional e distribuída, construída sobre o *Apache Hadoop*. Página *web*: `http://hbase.apache.org`.

38 Comando *UNIX* que concatena e imprime ficheiros para o *standard output* ou outros.

39 Comando *UNIX* usado para retornar ao utilizador as últimas linhas de um ficheiro de texto. Em oposto, existe o comando *head*, para retornar as primeiras linhas.

40 Comando *UNIX* que permite a pesquisa de padrões, dados pelo utilizador, num ou mais ficheiros de texto.

discipline over two centuries old: Data visualization, the craft of communicating patterns and trends in raw data by transforming it into visual displays"

Com isto, verifica-se a necessidade de transformar os dados em formato textual para algo mais compreensível e de rápida análise - Gráficos. Com estes, será muito mais fácil verificar variações e outros padrões que podem estar de alguma forma relacionados com ataques ou outro tipo de anomalias na infraestrutura a observar. É ainda possível a construção de painéis, de agora em diante designados por *dashboards*, que nos permitirão ter acesso a múltiplos dados a partir de uma única página e com isto poupar tempo de pesquisa. Grande parte das soluções disponíveis, que serão discutidas mais a frente, permitem interagir com os gráficos ou tabelas de forma a controlar os dados observados através de diversas variáveis, quer sejam temporais, espaciais ou mais específicas, como códigos de erro, categorias, *etc*., desde que este tipo de informação esteja disponível nos dados sobre os quais estas *dashboards* são construídas.

Para o desenvolvimento/configuração destas *dashboards*, é possível recorrer a plataformas já existentes ou recorrer a bibliotecas para criação ou extensão das já existentes. Para esta dissertação e devido a sua popularidade, serão discutidas uma plataforma e uma biblioteca para a geração de gráficos, *Kibana* e *D3.js*, respectivamente.

2.5.1 *Kibana*

Foi desenvolvido pelo projecto *Logstash* (ver Secção 2.2.2), mas foi mais tarde absorvido pela *Elastic*, assim como aconteceu com o *Logstash*.

Construído em HyperText Markup Language (HTML), Cascading Style Sheets (CSS) e *JavaScript*. Recorre a algumas *frameworks* em *JavaScript*, nomeadamente *ExpressJS*[41] e *AngularJS*[42]. Esta aplicação corre sobre *NodeJS*[43] e, a partir da versão 4, versão à data de escrita deste documento, não é necessário configurar nenhum servidor *web*, uma vez que este é distribuído como um executável, tornando muito mais fácil a sua instalação e configuração.

Na Figura 1 é possível observar a utilização do *Kibana* 4 para a construção de uma *dashboard*, onde estão disponíveis um conjunto de informações sobre uma aplicação *web*.

41 *Framework* para criação de aplicações *web* para *NodeJS*.
42 *Framework opensource*, mantida pela *Google*, para desenvolvimento de aplicações *web*, com foco em aplicações *single-page*.
43 Ambiente multi-plataforma que permite a execução de aplicações escritas em *JavaScript*, do lado do servidor.

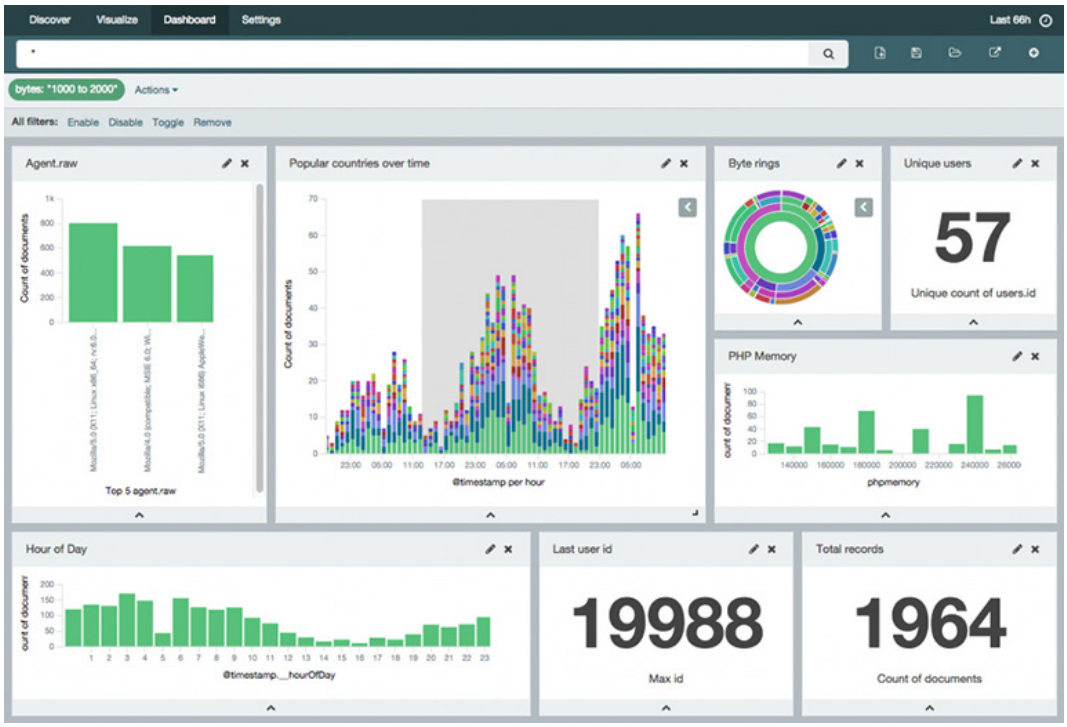

Figura 1.: *Dashboard* configurada com recurso ao *Kibana* 4[44].

Esta aplicação funciona sobre *ElasticSearch*, mas, e uma vez que nesta dissertação é também abordado o *Solr* como mecanismo de armazenamento a curto-prazo, existe então o *Banana*[45]. O *Banana* é baseado na versão 3 do *Kibana* e é desenvolvido e mantido pela *LucidWorks*[46].

Na Figura 2 é possível ver um caso de uso do *Banana* para a criação de uma *dashboard* com os dados de um servidor *web*.

44 Fonte: `http://blog.scottlogic.com/cprice/assets/Kibana4beta3.png`
45 Página *web*: `https://github.com/LucidWorks/banana`.
46 Empresa norte americana, focada no desenvolvimento de soluções de pesquisa e análise de dados com base em *Solr*, assim como na prestação de serviços comerciais e consultadoria na mesma área.

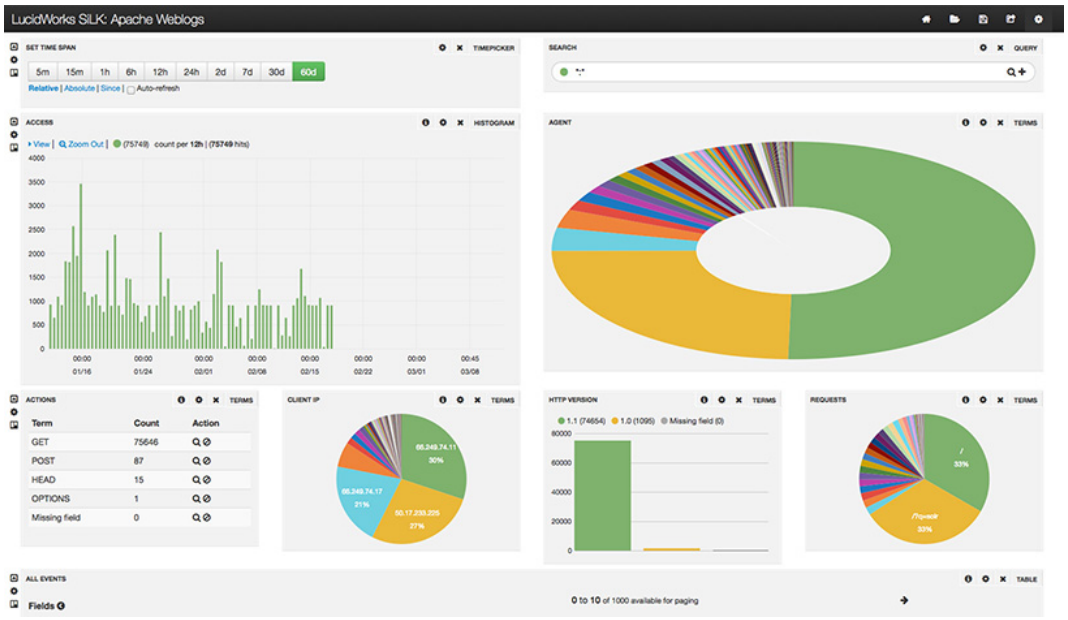

Figura 2.: *Dashboard* criada com recurso ao *Banana*[47].

2.5.2 *D3.JS*

O *D3.js*[48] é uma biblioteca *JavaScript* quer permite ligar dados, através da manipulação de documentos, ao Document Object Model (DOM) e, transformá-los e gerar diferentes elementos, recorrendo a HTML, Scalable Vector Graphics (SVG) ou CSS.

Ao contrário do *Kibana* e *Banana*, o *D3.js* não é uma plataforma com a facilidade de implementação e configuração. O *D3.js* é apenas uma biblioteca que permite a construção de plataformas como as apresentadas anteriormente ou até mesmo superiores, tanto a nível de qualidade como de adaptabilidade ao contexto das infraestruturas.

O seu desenvolvimento começou em finais de 2010 e no ano seguinte foi lançada, oficialmente, a primeira versão. Apesar de agora o *D3.js* ser mantido por uma comunidade, os seus principais contribuidores são *Michael Bostock*, *Jeffrey Heer* e *Vadim Ogievetsky*.

Nas Figuras 3 e 4 é possível ver alguns exemplos de *dashboards* construídas recorrendo a esta biblioteca.

47 Fonte: `http://data-informed.com/wp-content/uploads/2014/09/banana-dashboard-crop.png`

48 Página *web*: `http://d3js.org`.

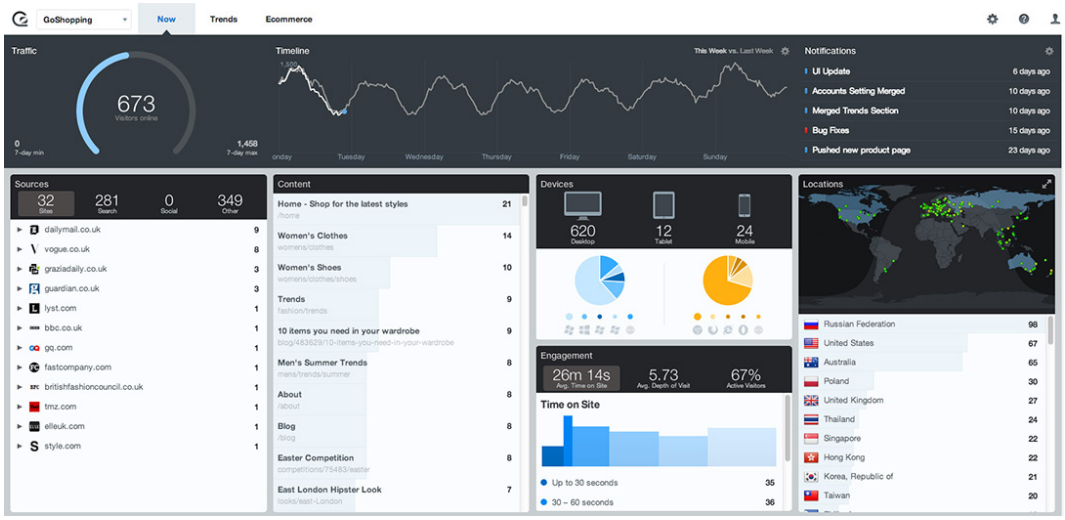

Figura 3.: *Dashboard NOW*, disponibilizada pela *GoSquared*[49].

Figura 4.: *Dashboard "The incredible rise of migrants' remittances"*[50].

49 Fonte: https://static.gosquared.com/images/screens/now-standard.png
50 Fonte: http://visualizing.org/full-screen/54850

3

ABORDAGEM AO PROBLEMA

Neste capítulo será apresentada a evolução de um modelo que se espera capaz de resolver os desafios enunciados neste documento. Assim como definir e discutir quais das soluções que foram apresentadas no Capítulo 2, se enquadram com o modelo proposto.

3.1 ARQUITECTURA

As próximas secções irão demonstrar a evolução, a partir de uma arquitectura que é ainda bastante utilizada atualmente, até um modelo que se espera capaz de suportar as necessidade atuais e futuras de uma solução deste género.

À medida que os modelos vão sendo apresentados, vai ser possível ganhar consciência de que será necessário, cada vez mais, apostar numa infraestrutura dedicada à gestão e análise deste tipo de eventos.

3.1.1 *Localização única*

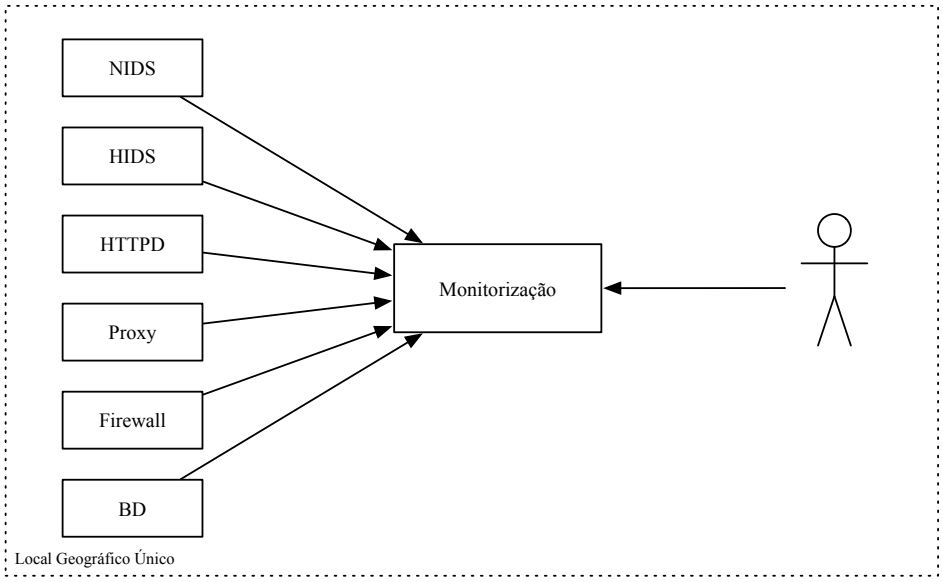

Figura 5.: Modelo simples e de localização geográfica única

Como mostra a Figura 5, todas as máquinas enviam os seus *eventos* para uma máquina central através de um mecanismo de transporte como enunciado no capítulo anterior, *e.g. Rsyslog*, que depois serão analisados por alguém, normalmente um administrador de sistemas. Por norma, estes dados são guardados em ficheiros de texto.

Este modelo não soluciona a distribuição geográfica, como torna impossível, para quem for responsável pelo controle e análise dos eventos, fazer o seu trabalho de forma eficiente à medida que se avança para ambientes de *Big Data*.

Neste caso, até poderia ser excluída a máquina central de monitorização, passado o responsável a percorrer todas as máquinas e analisar todas as máquinas, individualmente.

3.1.2 *Distribuição geográfica*

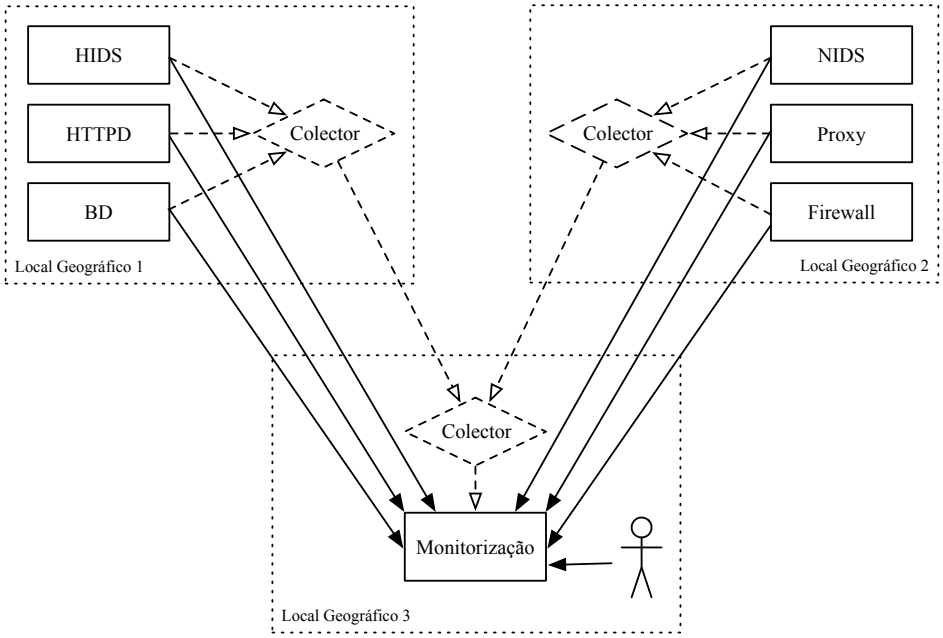

Figura 6.: Distribuição geográfica simples

Este modelo, como demonstra a Figura 6, já resolve, inicialmente, o problema da distribuição geográfica, assumindo o envio direto, ou não, dos dados para uma infraestrutura remota. Isto leva à centralização geográfica deste tipo de dados, o que é algo importante daqui para a frente, uma vez que possibilita que o responsável pelos dados não tenha tenha que se deslocar geograficamente. Um problema com a centralização é o aumento do volume de dados e tráfego de rede, sendo para isso necessários mecanismos capazes de o minimizar.

Existe a possibilidade da introdução de colectores de dados nas infraestruturas onde se pretende recolher dados. Estes colectores irão recolher os dados e enviar os mesmos em grupos de x eventos, para a infraestrutural central. Isto permite libertar alguns recursos de rede, mas também implica atrasos no processamento em (quase) tempo-real, caso esse processamento seja um requisito. Os colectores poderão ainda ter outras funções, como acompanhar o crescimento dos ficheiros para onde os dados são escritos e enviá-los, à medida que vão sendo escritos, para a infraestrutura central através de mecanismos como os que foram apresentados na secção de transporte (ver Secção 2.2) do capítulo anterior.

Na infraestrutura central, o armazenamento dos dados é feito, geralmente, em ficheiros de texto, em que cada evento, corresponde, geralmente a uma linha do ficheiro. Com o crescimento do volume de dados gerados, torna-se também necessário interrogar estes dados e relacioná-los. Para isso foram

introduzidos os motores de bases de dados SQL, como forma de armazenar, analisar e interrogar os dados. Para o problema enunciado neste documento, nenhuma destas soluções, ficheiros de texto ou SQL, será suficiente, sendo necessário desenhar uma nova solução para o armazenamento.

3.1.3 *Distribuição geográfica e armazenamento diferenciado*

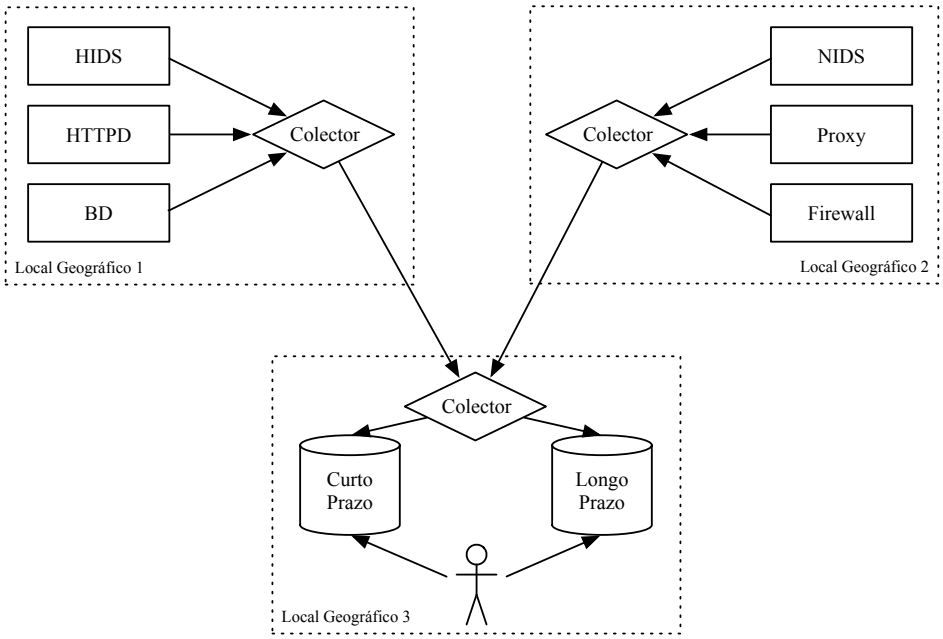

Figura 7.: Distribuição geográfica e armazenamento diferenciado

Como foi enunciado no capítulo anterior, existe a necessidade de separar o armazenamento em curto e longo-prazo, como se pode ver na Figura 7. Estes tipos de armazenamentos tem necessidades e propósitos diferentes, como se passa a explicar.

O armazenamento a curto-prazo pretende armazenar os últimos dados e mantê-los disponíveis durante um curto espaço de tempo. Dependendo das necessidades e do volume de dados, esta duração de armazenamento pode ir de algumas horas até dias, ou semanas, mas nunca ultrapassando o período de um mês. Com isto, pretende-se que estejam disponíveis os dados mais recentes e críticos para uma análise do estado atual da infraestrutura. Desta forma, encontramos um volume de dados inferior ao modelo apresentado anteriormente, o que permite uma maior rapidez e eficácia na análise e interrogação do dados. A diminuição do volume de dados, é também muito importante, uma vez que estes dados poderão estar sujeitos a grandes quantidades de interrogações e leituras aleatórias, ou seja, diminuindo o volume de dados, diminuí o tempo de execução destas operações.

O armazenamento a longo-prazo, também designado por arquivo, pretende ser um armazenamento com períodos de retenção muito superiores ao armazenamento de curto-prazo. Estes períodos podem ir de meses a alguns anos. Algumas entidades, com necessidades específicas, podem necessitar que os seus dados sejam retidos para sempre, precisando então de um mecanismo de armazenamento escalável e distribuído. Os dados armazenados em arquivo serão utilizados para análise histórica, *bussiness intelligence* e análise forense.

Esta diferenciação no armazenamento também pode levar a algumas complicações, nomeadamente, na visualização dos dados, processamento em (quase) tempo-real e *batch*. Algo que também é necessário ter em consideração é a possibilidade, se necessário, de escalar o colector de dados central.

3.1.4 *Distribuição geográfica e Escalabilidade*

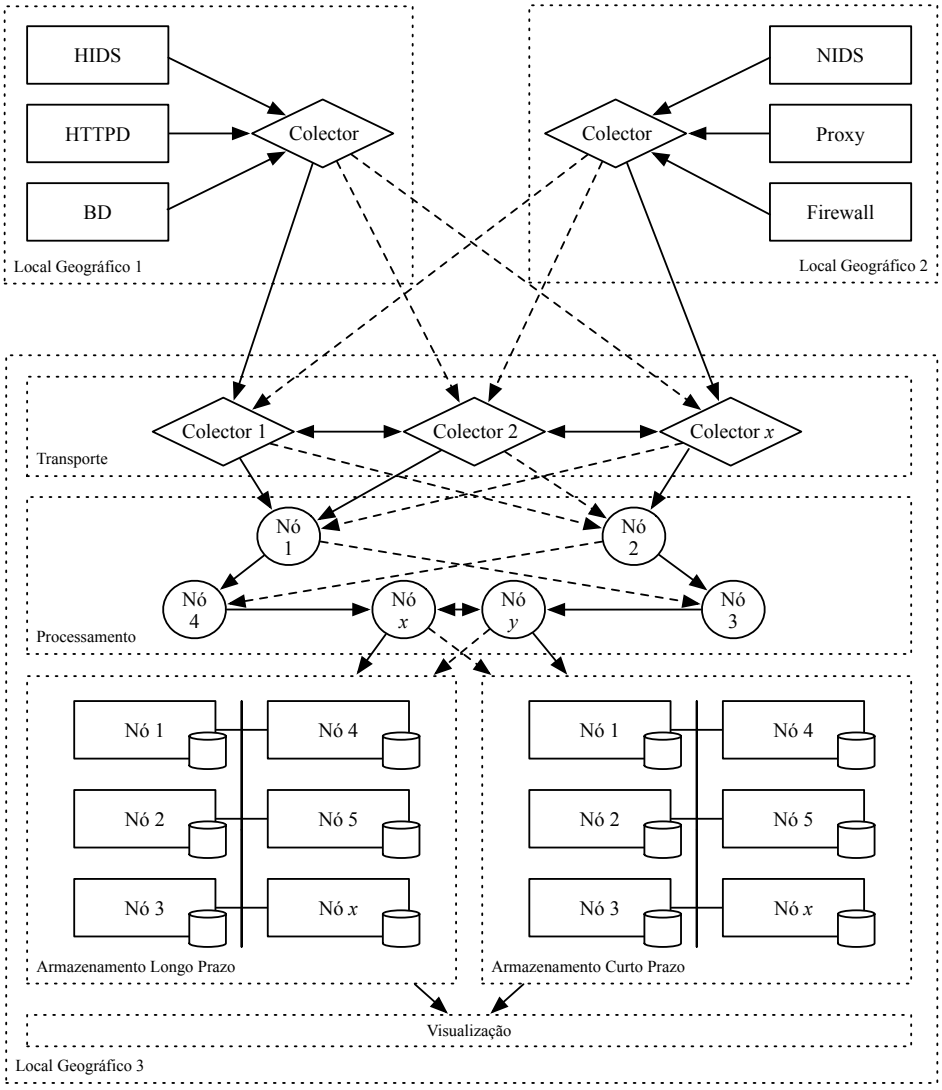

Figura 8.: Distribuição geográfica e Escalabilidade

Analisando a evolução dos modelos, será necesário uma infraestrutura quase que exclusiva ao suporte de todas as características necessárias a este tipo de solução. Como também é possível verificar, por este modelo de distribuição geográfica e escalabilidade (ver Figura 8), este é também o modelo mais genérico para o problema que queremos tratar, abordando todos os aspectos, desde a recolha de dados até a sua visualização, não esquecendo a distribuição geográfica e escalabilidade do próprio sistema. A razão de alguns elementos estarem numerados com um x, fortifica essa escalabilidade, uma vez que é possível escalar até milhares de nós, limite esse que depende apenas das tecnológias a usar para cada uma das áreas.

Apesar de neste modelo, o processamento e o armazenamento a longo-prazo estarem separados e usarem diferente nós, existem algumas tecnologias de processamento que permitem co-existir nas mesmas máquinas utilizadas para o arquivo, sem qualquer comprometimento de performance, aproveitando a localidade dos dados para um acesso mais rápido aos mesmos, resultando em tempos de processamento menores.

Na próxima secção, serão introduzidas as tecnologias apresentadas no Capítulo 2 e como estas se poderão integrar em cada uma das áreas deste modelo, assim como o que podem trazer de relevante para a solução.

3.2 INTEGRAÇÃO DAS TECNOLOGIAS

Nesta secção. será discutido como se poderá integrar as tecnologias apresentadas no Capítulo 2, nas áreas necessárias para o desenvolvimento e configuração do modelo apresentado na secção anterior (ver Figura 8).

Neste capítulo ainda não serão avaliadas as tecnologias em causa. Será feita sim, uma análise muito subtíl sobre o que cada uma das tecnologias poderá trazer para cada uma das áreas, de como será possível as interligar e da sua natureza de adaptação a uma solução distribuída e escalável.

Em resumo, pretende-se demonstrar que tecnologias serão avaliadas e para que áreas (transporte, armazenamento, processamento e visualização.)

3.2.1 *Transporte*

Aqui pretende-se introduzir os mecanismos que permitem o transporte dos dados entre as máquinas que pretendemos analisar, ou manter sob vigilância, e a insfraestrutura que estará responsável por analisar e armazenar esses dados.

No Capítulo 2, foram introduzidos quatro mecanismos de transporte que permitem satisfazer essa necessidade. Estas tecnologias são o *Rsyslog*, *Logstash*, *FluentD* e *Apache Kafka*.

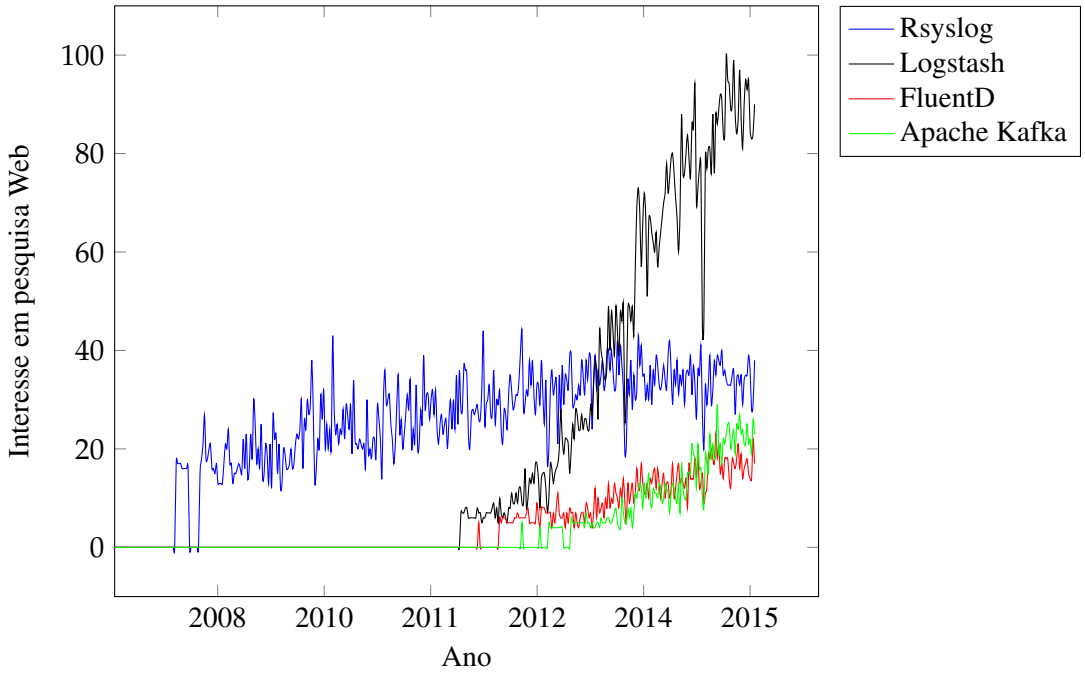

Figura 9.: Interesse nas soluções *Rsyslog*, *Logstash*, *FluentD* e *Apache Kafka*, recorrendo ao motor de busca *Google*[1].

Na Figura 9, é possível observar a evolução da pesquisa destas soluções, no motor de busca *Google*[2], ao longo dos últimos anos.

Para cada uma das tecnologias, será feita uma pequena abordagem aos requisitos da solução e se estes são satisfeitos, nomeadamente para questões relacionadas com distribuição geográfica, escalabilidade, alta disponibilidade, segurança, processamento básico e facilidade de configuração.

Distribuição Geográfica

Todas as soluções são capazes de operar de forma distribuída, sendo que o desenvolvimento destas ferramentas teve como requisito essa distribuição.

Escalabilidade e Alta disponibilidade

O *Rsyslog* não possuí nativamente estas características, para tal é necessário recorrer a ferramentas externas, tais como *iptables CLUSTERIP*[3] + *pacemaker*[4] ou Linux Virtual Server (LVS)[5]

1 Fonte: `https://www.google.com/trends/explore`.
2 Página *web*: `http://www.google.com`.
3 Extensão do *software iptables* que permite configurar um conjunto de nós que partilham o mesmo endereço IP e Media Access Control (MAC) sem a necessidade de um balanceador de carga, fazendo este o balanceamento de carga, estaticamente.
4 Gestor de recursos para *clusters*. Permite adicionar características de alta disponibilidade, entre outras.
5 O LVS permite a configuração de servidores escaláveis e altamente disponíveis apartir de um conjunto de nós, que fazem parte de um *cluster*. Possuí ainda características de balanceamento de carga, entre outras.

Numa instalação *Logstash*, para escalar e atingir alta disponibilidade é necessário recorrer a ferramentas não nativas do próprio *Logstash*. Ou seja, é possível escalar horizontalmente através da instalação de novas máquinas, configurandas com a solução, e configurar depois um balanceador de carga em frente a essas máquinas. Além do balanceador de carga, será necessário configurar também um mecanismo de *buffering*, geralmente uma *message queue*. Cada instância do *Logstash* permite armazenar, no máximo, até 20 mensagens e não possuí outro mecanismo de *buffering*, nativo.

O *Fluentd* permite ser escalável tanto vertical, como horizontalmente, recorrendo para isso a dois *plugins* distintos:

- ***in_multiprocess***: Escala verticalmente, tirando partido de Central Processing Unit (CPU) multinúcleo. Por predefinição, o *Fluentd* é executado apenas por um dos núcleos.
- ***out_forward/out_secure_forward***: Escalar horizontalmente. Recorre à criação de um conjunto de servidores, configurados com *Fluentd*, e à configuração dos vários servidores disponíveis nos clientes. Estes *plugins* permitem o balanceamento de carga entre os servidores e um mecanismo de *failover* automático.

É ainda possível configurar estes *plugins* em simultâneo, resultando numa solução bastante escalável.

Para escalar o *Kafka*, e uma vez que este foi desenvolvido sob uma ideia de *pub/sub*, basta aumentar o número de *brokers* e consumidores. Neste contexto, os consumidores serão agentes responsáveis por consumir os dados dos *brokers* e fazerem a ligação a fase seguinte. Já os *brokers* são os responsáveis por receber as informações dos colectores e armazená-la até esta ser consumida.

Segurança

Todas as solução permitem o envio dos dados através de TCP, mas apenas três das soluções permitem o envio de dados cifrados, recorrendo a TLS/SSL, que são o *Rsyslog*, *Logstash* e *Fluentd*. Apesar de o *Apache Kafka* não possuir essa característica, já está em desenvolvimento.

Processamento Básico

À excepção do *Kafka*, todas as soluções possuem *plugins* que permitem realizar algum processamento básico sobre os eventos, nomeadamente, adição, remoção ou alteração de valores, filtragem das mensagens, *etc*. O *Logstash* e o *Fluentd* possuem o conjunto mais diversificado de *plugins*, que permite alargar as competências das soluções.

Configuração

As soluções mais fáceis de configurar, são sem dúvida, o *Fluentd* e o *Logstash*. Para este resultado foi avaliado o processo desde a instalação até executar a solução. Apesar de o *Rsyslog* e *Kafka* serem

de fácil instalação, possuindo também binários de distribuição como as outras duas soluções, apresentam uma configuração mais complexa. De todas as soluções, a que apresenta o melhor processo de confguração é o *Fluentd*, uma vez que além do processo simples de configuração de apenas um agente e um coletor, é possível também escalar e configurar para alta disponibilidade, muito facilmente e sem grandes riscos.

3.2.2 *Armazenamento*

Para esta área foram apresentadas três soluções distintas para os dois tipos de armazenamento, a longo e a curto-prazo.

Para o armazenamento a longo-prazo foi apenas apresentada uma solução, pois é a mais relevante e é a que apresenta maior desenvolvimento e aceitação nos últimos anos, o HDFS.

Quanto ao armazenamento a curto-prazo, foram apresentadas duas soluções, o *Solr* e o *Elasticsearch*, ambas baseadas no *Lucene*.

As diferenças mais evidentes entre ambas as soluções são a facilidade de configuração e as configurações necessárias para escalar e manter a alta disponibilidade do sistema. Quanto as restantes características, ambas as soluções são bastante semelhantes, uma vez que são ambas baseadas na mesma tecnologia. Apesar disso, o *Elasticsearh* tem demonstrado um enorme desenvolvimento, comparativamente ao *Solr*. Esse rápido desenvolvimento tem levado a um aumento do interesse pelo *Elasticsearch*, que fez com que o número de pesquisas, no motor de busca *Google*, ultrapassa-se o *Solr* no início de 2014 e continuasse a gerar cada vez mais interesse, ao contrário do *Solr*. Estes dados são facilmente confirmados pela análise da Figura 10.

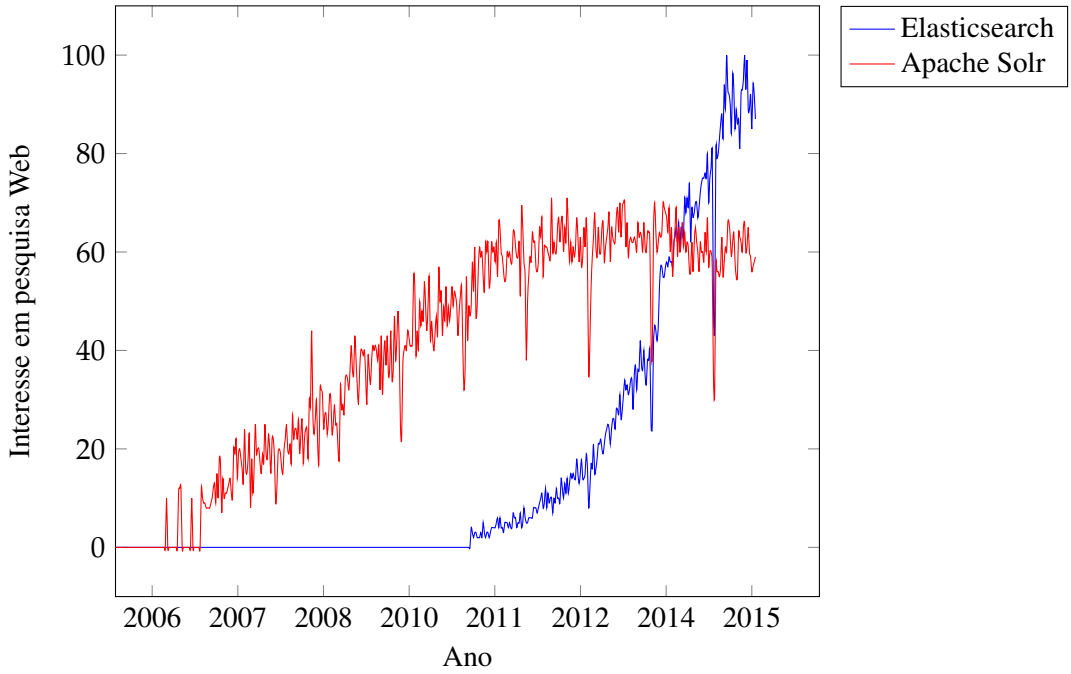

Figura 10.: Interesse nas soluções *Elasticsearch* e *Apache Solr*, recorrendo ao motor de busca *Google*[6].

3.2.3 *Processamento*

No capítulo anterior foram introduzidos dois tipos de processamento, em *batch* e em *stream*, ou (quase) tempo-real. Foram apresentadas quatro tecnologias distintas, sendo que uma delas, o *Hadoop MapReduce*, é apenas utilizada para processamento em *batch*, sendo que as restantes, *Storm*, *S4* e *Spark Streaming*, estão focudas em resolver problemas associados com o processamento em (quase) tempo-real, apesar que o *Spark* possui uma componente capaz de substituir o *Hadoop MapReduce* no processamento em *batch*.

No gráfico seguinte é possível ver o interesse demonstrado ao longo do tempo, através de pesquisas *web*, para duas das soluções para processamento em (quase) tempo-real. É de salientar que o *S4* não é apresentado no mesmo gráfico devido ao seu baixo interesse em pesquisa.

6 Fonte: https://www.google.com/trends/explore.

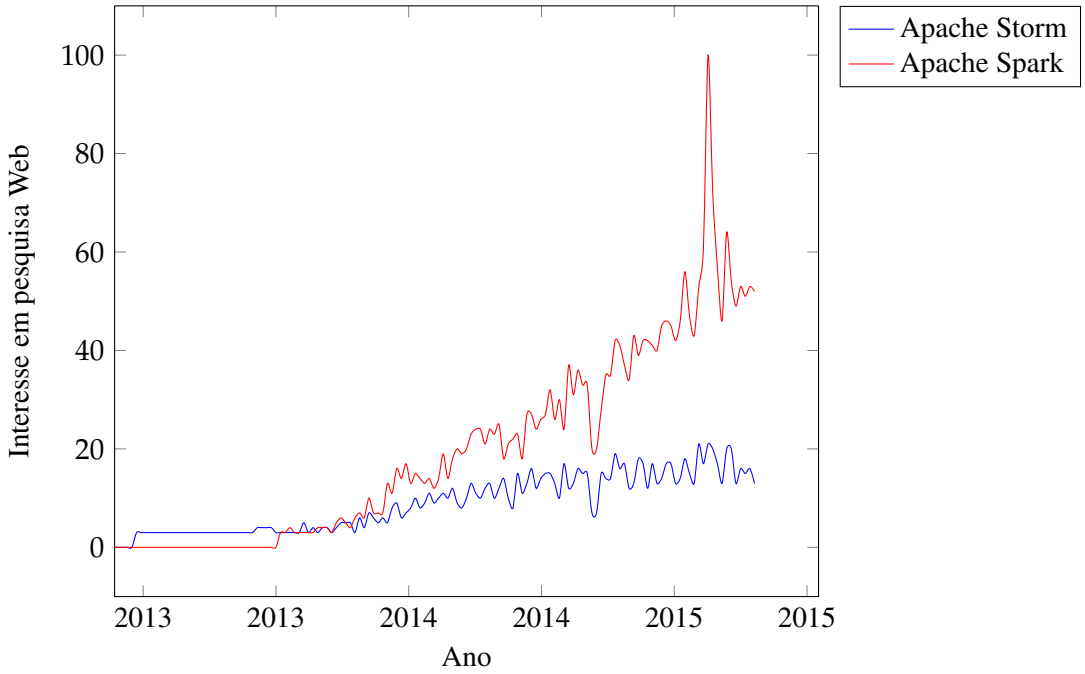

Figura 11.: Interesse nas soluções *Apache Storm* e *Apache Spark*, recorrendo ao motor de busca *Google*[7].

Apesar de o *Spark* poder ser executado na mesma infraestrutura que o HDFS, esta não será uma escolha a ter em conta para esta fase, uma vez que o *Spark* consegue o seu aumento de desempenho em relação ao *Hadoop MapReduce* através da execução em memória, mas a partir do momento que não seja possível colocar todos os dados a processar em memória, o desempenho irá degradar-se, tornando assim, o *Hadoop MapReduce* uma solução mais viável, tanto ao nível da quantidade de dados a tratar como nos requisitos de *hardware*, que serão menores, especialmente a nível de memória.

3.2.4 *Visualização*

A visualização dos dados será uma importante parte desta dissertação, uma vez que permitirá olhar para todos os dados da forma mais simples possível. Para isso foram apresentadas duas soluções, *Kibana* e *D3.js*. Como é possível verificar pela análise da Figura 12, o *Kibana* é uma ferramenta que tem revelado um crescente interesse e que demonstra também grande desenvolvimento.

7 Fonte: https://www.google.com/trends/explore.

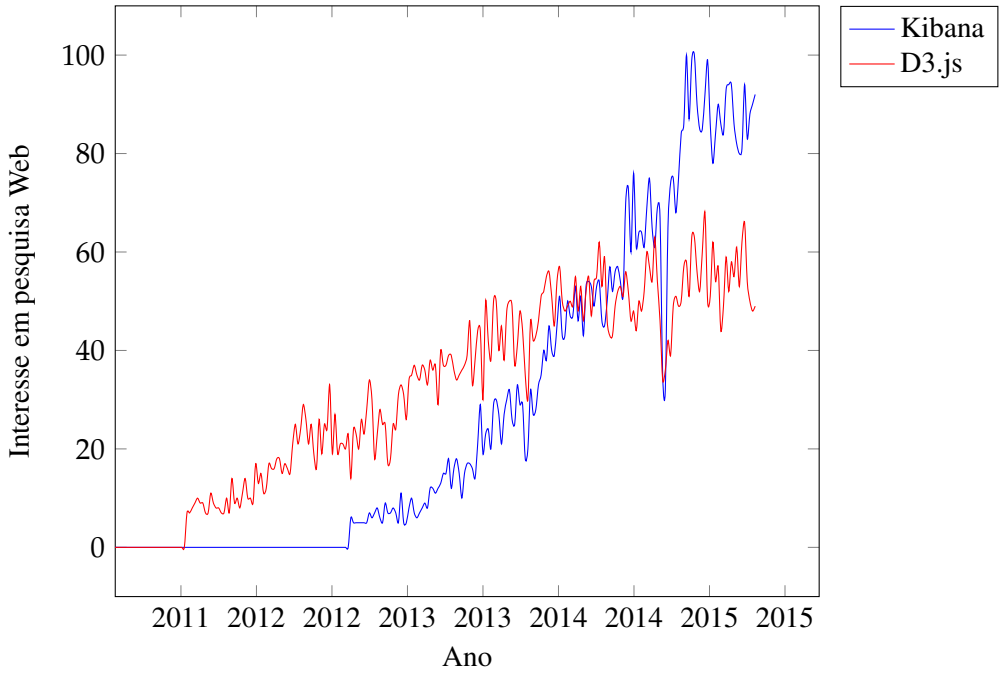

Figura 12.: Interesse nas soluções *Kibana* e *D3.js*, recorrendo ao motor de busca *Google*[8].

Neste caso não será necessário realizar uma análise mais profunda entre as duas soluções porque é possível o uso das duas como complementares, ou seja, caso o *Kibana* não disponibilize algum tipo de gráfico, será sempre possível usarmos o *D3.js* para acrescentar funcionalidades ao *Kibana*. Por esta razão, o próximo capítulo não contará com uma secção destinada a avaliar estas duas soluções.

8 Fonte: https://www.google.com/trends/explore.

4

AVALIAÇÃO

Este capítulo mostra o comportamento e desemepenho, em termos práticos e de avaliação, das soluções apresentadas na Secção 3.2 e apresenta uma breve conclusão sobre qual das várias soluções apresentadas é a melhor escolha para cada uma das áreas do modelo apresentado na Secção 3.1.4.

4.1 TRANSPORTE

Uma vez que uma grande parte das comparações entre soluções já foi feita na Secção 3.2.1, este espaço esta reservado à comparação dos resultados praticos da avaliação de duas soluções, o *fluentd* e *logstash*. A escolha destas duas soluções para avaliação, foi feita por serem de fácil configuração e por possuírem uma quantidade considerável de *plugins* para leitura/escrita de dados, assim como filtros que permitem a manipulação dos dados, *i.e.*, processamento básico.

Considera-se o *Apache Kafka* uma solução a ter em conta quando se lida com infraestruturas de grande escala e ambientes *Big Data* e, concerteza, uma solução que terá um desenvolvimento acelerado e uma adopção rápida por entidades que necessitem de mecanismos capazes de transportarem uma grande quantidade de eventos para processamento em quase tempo-real. Nesse sentido, é uma solução tida em conta como uma evolução natural deste tipo de sistemas e por isso, é apontada como trabalho futuro, desta dissertação (ver Secção 5.1).

Após a instalação/configuração dos dois ambiente de teste, ver Apêndices A.1 e A.2, e terminados as avaliações de desempenho, retirou-se as informações necessárias para o desenho das Figuras 13, 14 e 15. Estas representações gráficas permitem uma comparação, rápida, de ambas as soluções recorrendo a alterações no número de *threads* e no número de eventos gerados por segundo.

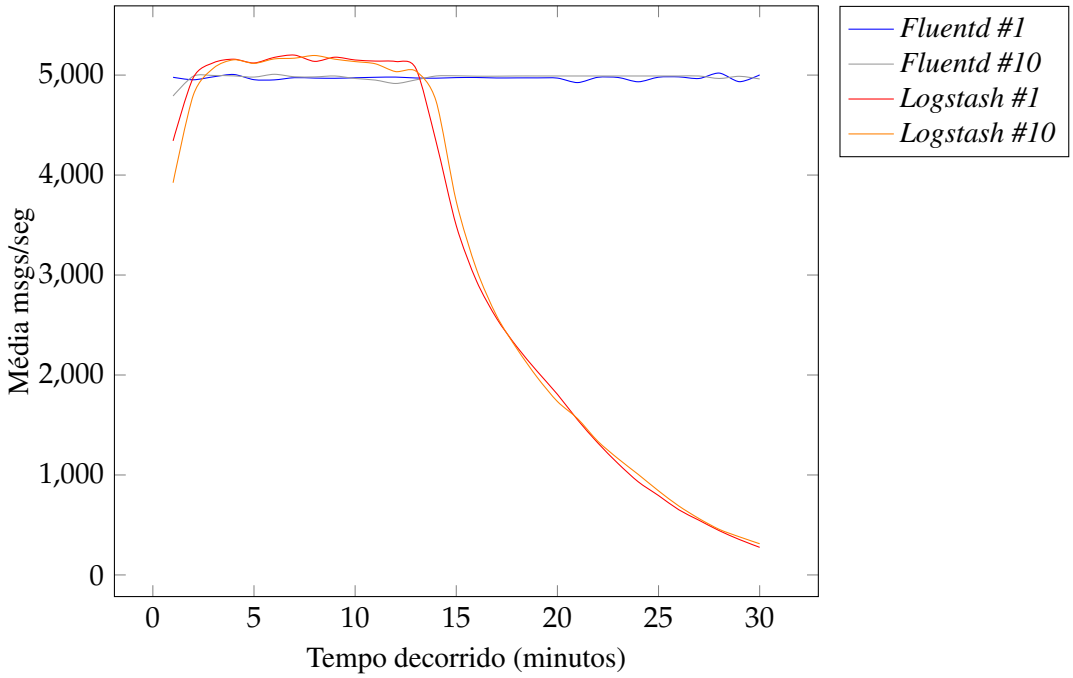

Figura 13.: Comparação de desempenho entre *Fluentd* e *Logstash*, durante 30 minutos, para uma geração de 5000 *eventos/seg*.

Na Figura 13, verifica-se que a alteração do número de *threads* não provoca nenhumas alterações vísiveis no desempenho de ambas as soluções. O *fluentd* mantém sempre o seu desempenho a rondar as 5000 *msgs/seg*, enquanto que o *logstash* perde desempenho, de forma significativa a partir dos 15 minutos decorridos. Esta perda de desempenho por parte do *logstash* é extremamente preocupante quando queremos atingir elevado desempenho e verifica-se que essa redução não é temporal, uma vez que a partir do momento em que começa a reduzir, em nenhum ponto, no intervalo de tempo das avaliações, demonstrou recuperação. Este problema pode derivar do facto de o *logstash* correr sobre JVM, o que leva a graves problemas de gestão de memória e desempenho, quando as configurações da JVM não são adequadas à máquina ou quando a máquina não possui um elevado número de recursos físicos.

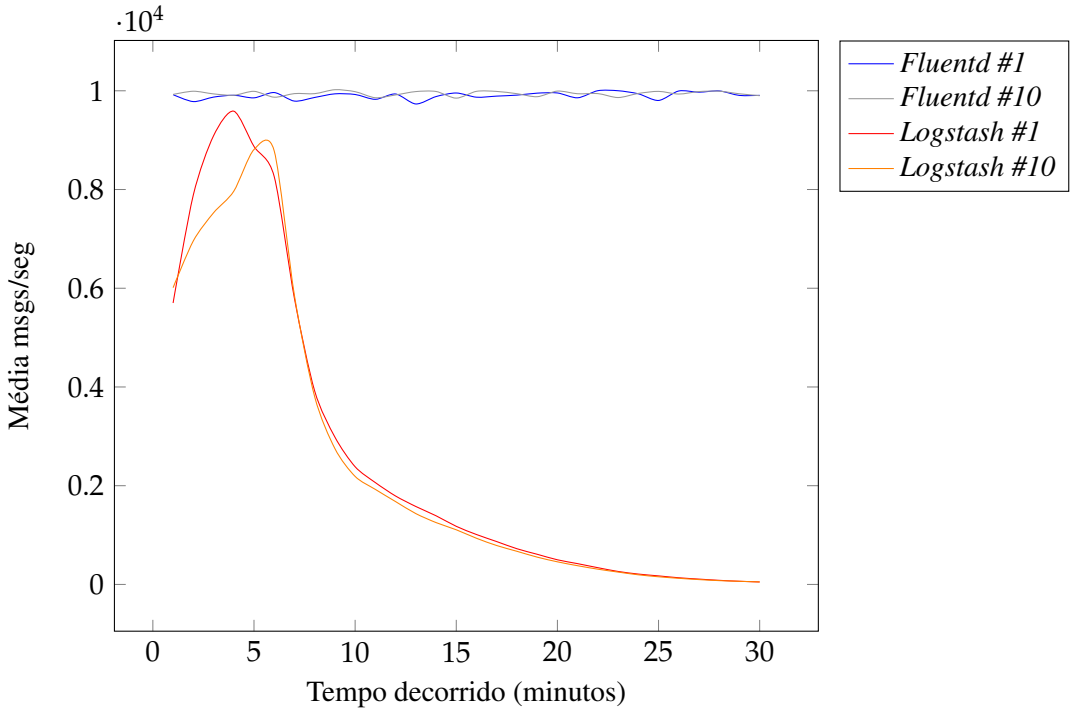

Figura 14.: Comparação de desempenho entre *Fluentd* e *Logstash*, durante 30 minutos, para uma geração de 10000 *eventos/seg*.

A semelhança da Figura 13, a Figura 14 mostra um comportamento semelhante, com o *fluentd* com um desempenho a rondar o mesmo número de eventos gerados por segundo, enquanto que o *logstash* mantém o mesmo desempenho e a mesma quebra, desta vez entre os 5 e os 10 minutos decorridos. Isto poderá ter acontecido devido a duplicação do número de eventos gerados por segundo, que levou a que o *logstash* entrasse em sobrecarga mais cedo.

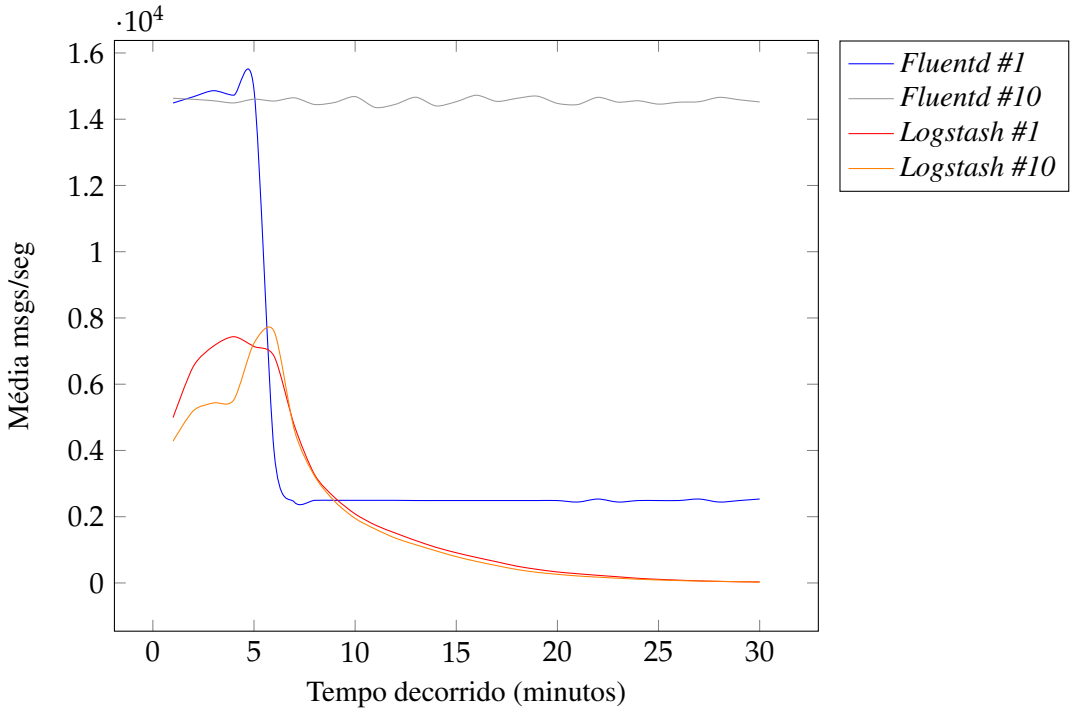

Figura 15.: Comparação de desempenho entre *Fluentd* e *Logstash*, durante 30 minutos, para uma geração de 50000 *eventos/seg*.

Na Figura 15, já é possível verificar um comportamento diferente do *fluentd*. Quando configurado com uma *thread*, o *fluentd*, demonstrou uma queda acentuada de desempenho a partir dos 5 minutos. Após essa queda, continuou a manter um nível regular de eventos transportados por segundo, acabando mesmo por voltar a ultrapassar, 9 minutos decorridos, o *logstash* que também já demonstrava quebra no desempenho desde os 6 minutos. Assim que o número de *threads* é actualizada para 10, verifica-se que o *fluentd* já não apresentou nenhuma quebra de desempenho ao longo dos 30 minutos, mas apresentou um limite, *i.e.*, apenas transportava aproximadamente 15000 *eventos/seg*, dos 50000 que estavam a ser gerados. O *logstash* manteve o seu comportamento, já demonstrado nas Figuras 13 e 14.

Após alguma pesquisa pela página *web* da solução, foi possível descobrir que outros utilizadores relatavam o mesmo problema, quando configuravam duas instâncias do *logstash*. Uma das formas de contornar este problema, seria a utilização de outros mecanismos que permitam o envio dos dados para um servidor *logstash* central, nomeadamente, do *logstash-forwarder* ou do *Beaver*.

A partir da análise feita as Figuras 13, 14 e 15, e ainda as comparações feitas, entre soluções, na Secção 3.2.1, conclui-se que para o mecanismo de transporte a incorporar a aquitectura proposta neste documento (ver Secção 3.1.4), deverá ser tido em conta o *fluentd*, até que este apresente restrições, nomeadamente a nível de desempenho, que o impeçam de ser considerado uma solução válida.

4.2 ARMAZENAMENTO

4.2.1 *Longo-Prazo*

A avaliação do HDFS divide-se em três partes de extrema importância para um sistema de armazenamento de dados a longo prazo: desempenho nas leituras, escritas e a tolerância a faltas. É sobre estes três tópicas que esta subsecção está divida.

A configuração do *cluster* usado para os testes desta solução podem ser visto no Apêndice B.3.

Leituras

Para estes testes, foram gerados cerca de 28GB de dados aleatórios, divididos em múltiplos ficheiros que rondavam, em tamanho, os KB e os MB, poucas vezes atingindo as dezenas de MB.

Estes testes correram sobre máquinas virtuais e estas estavam limitadas em termos de espaço, sendo esta a razão para a quantidade de dados reduzida. Este valor é baixo, mas está ainda a ser tido em conta um fator de replicação de dois, configurado para o HDFS.

Foi utilizada uma *framework* para a execução dos testes, a Yahoo! Cloud Serving Benchmark (YCSB)[1]. Esta *framework* disponibiliza vários *bindings* para bases de dados *NoSQL*, nomeadamente, *Cassandra*, *HBase*, *DynamoDB*, *Redis*, etc., mas não para HDFS. Nesse sentido, foi necessário desenvolver um *binding* que recorre aos serviços REST disponibilizados pelo HDFS, o *WebHDFS*. O *plugin* foi desenvolvido em torno do *WebHDFS*, porque outras soluções aqui apresentadas, como as soluções de transporte, utilizam *plugins* para comunicação com o *HDFS*, mas atráves da sua API REST - *WebHDFS*.

Outra nota de salientar, é que o HDFS está otimizado para leituras sequênciais e de preferência em ficheiros de grande tamanho e não em ficheiros como os que são utilizados neste teste. A utilização de ficheiros de tamanho reduzido serviu também para validar a fraca performance do *Hadoop*, nesta situação, remetendo esta solução para armazenamento em longo-prazo, apenas.

1 https://github.com/brianfrankcooper/YCSB/wiki

Para popular o HDFS com os dados de teste, foi utilizado um *script* em *Java*. Foram necessários 67042 segundos, aproximadamente 18 horas e 30 minutos para completar o processo. Este *script* funcionava da seguinte forma:

while *existem ficheiros resultantes de* `ls -lah /path/to/random_data_set/` **do**
    if *tamanho total ocupado no HDFS menor que 28GB* **then**
        escreve ficheiro para HDFS;
        escreve em ficheiro local, a *path* do ficheiro no HDFS;
        // este ficheiro local serviu como listagem de chaves para o YCSB
    else
        termina a execução do ciclo;
    end
end

Antes de serem executados os testes, definiu-se um *workload*, onde consta a execução de 100000 operações de leitura sobre um universo total de 1131396 chaves. Chaves essas, que neste caso, representam os ficheiros no HDFS.

Threads	*ops/seg*	Latência (ms)					Leituras	
		Méd	Mín	Máx	95th	99th	Sucesso	Insucesso
1	32.335	30.902	8.866	6367.711	51	128	100000	0
5	28.974	171.892	9.373	109768.341	179	-	100000	0
10	20.858	477.127	12.225	100767.46	-	-	99999	1
15	9.258	1611.574	11.404	489248.623	-	-	99964	26
20	122.966	125.147	11.831	200383.467	254	601	99787	213
25	225.230	110.054	10.498	4497.752	292	548	98621	1379

Tabela 1.: Resultado da primeira execução do teste de leituras aleatórias sobre HDFS

Threads	*ops/seg*	Latência (ms)					Leituras	
		Méd	Mín	Máx	95th	99th	Sucesso	Insucesso
1	37.944	26.328	8.618	15650.508	44	79	100000	0
5	119.072	41.854	9.333	2772.944	78	149	99999	1
10	51.643	192.403	7.596	28272.89	332	-	99998	2
15	25.281	588.532	10.286	91173.016	-	-	99910	10
20	171.420	115.742	9.572	8351.702	302	877	99999	1
25	166.725	148.598	10.801	11084.732	374	965	100000	0

Tabela 2.: Resultado da segunda execução do teste de leituras aleatórias sobre HDFS

Olhando para as Tabelas 1 e 2, é possível reparar que existe uma grande variação no *throughput*. Essa variação poderá ser devido a problemas no *binding* desenvolvido para o YCSB ou, por exemplo, devido ao *cluster* estar assente em tecnologias de virtualização.

É possível verificar um aumento de performance da primeira para a segunda execução. Este aumento deveu-se à alteração do *binding* utilizado pelo YCSB. Percebe-se então que estes resultados podem estar então limitados pela qualidade do *binding*.

A alteração do *binding* levou a:

- Diminuição do número de insucessos na leitura de ficheiros.
- Melhoramentos na latência.

Uma nota importante na comparação dos dois resultados, é que a primeira execução levou a um *throughput* na ordem das 220 operações por segundo, superior ao atingido na segunda execução. Mas, é também visível que esta diferença levou a um número superior de insucessos nas leituras durante a primeira execução.

Apesar desta solução, não permitir um elevado desempenho em leituras aleatórias de ficheiros, sabemos que esse não é o propósito para o qual a solução foi criada, nem o objectivo da utilização do mesmo no âmbito desta dissertação. Isto deve-se ao facto, do HDFS servir como sistema de arquivo e os seus dados serão armazenados em ficheiros de volume elevado, que corresponderão a horas ou dias de eventos. Estes ficheiros serão apenas lidos para a geração de relatórios mensais/anuais ou caso seja necessário algum tipo de análise de vulnerabilidade ao passado da infraestrutura. O mais importante é possuir uma capacidade elevade de ingestão de informação e uma boa tolerância a faltas.

Escritas

Nesta avaliação foi utilizado usado o *Fluentd*, uma vez que foi a ferramenta escolhida para o problema do transporte. Para a geração de eventos, métricas e envio de eventos para o HDFS, foi necessário instalar um conjunto de *plugins*:

- `fluent-plugin-dummydata-producer`[2] - Geração de eventos, para fins de simulação.
- `fluent-plugin-flowcounter`[3] - Métricas sobre o número de eventos e quantidade de *bytes* gerados e processados.

2 https://github.com/tagomoris/fluent-plugin-dummydata-producer
3 https://github.com/tagomoris/fluent-plugin-flowcounter

- `fluent-plugin-webhdfs`[4] - *Ouput* dos dados para HDFS.

Durante os testes foram introduzidas variações no número de eventos produzidos por segundo. Os testes foram executados apenas recorrendo a uma instância do *Fluentd*. Os resultados dos testes podem ser observados na tabela seguinte:

Gerador (*msgs/seg*)		*Throughput*					
Definido	Real	*msgs/seg*			*bytes/seg*		
		Méd	Mín	Máx	Méd	Min	Máx
500	504	504.2	496.7	504.9	126.7k	124.9k	127.1k
1000	1010	1.0k	1.0k	1.0k	253.6k	251.8k	254.1k
10000	8700	6.0k	6.0k	6.1k	1.5M	1.5M	1.5M
20000	14400	8.4k	8.3k	8.5k	2.1M	2.1M	2.1M
40000	21300	10.4k	10.3k	10.5k	2.6M	2.6M	2.6M
60000	25200	11.3k	11.1k	11.4k	2.8M	2.8M	2.9M
80000	27800	11.7k	11.6k	11.7k	2.9M	2.9M	3.0M
100000	29700	12.0k	11.9k	12.1k	3.0M	3.0M	3.1M

Tabela 3.: Resultados dos testes de escrita para HDFS

Nestes testes é possível reparar que *throughput* real do gerador tende a estabilizar nos 30 mil eventos por segundo e o envio de eventos estabiliza nos 12 mil eventos por segundo. Como foi referido anteriormente, estes valores são para apenas uma máquina.

Tolerância a Faltas

ResourceManager HA

O *ResourceManager* é o componente que a semelhança do *NameNode* faz a gestão do *MapReduce*, sendo responsável pelo escalonamento das tarefas, quais as tarefas que estão a correr e se existem falhas, *etc*. Este componente pertence ao *YARN*, ou *MapReduce v2*, mas ainda não possui um mecanismo de High-Availability (HA), na versão analisada. A partir da versão 2.4, publicada depois da execução desta avaliação, possui já um mecanismo de HA recorrendo a uma arquitectura de nó Activo/Passivo e ao *ZooKeeper* para fazer a gestão dos nós.

Apesar do *ResourceManager* ser SPOF, na versão 2.2.0, existe tolerância a nível dos *NodeManagers*, ou seja, dos processos que executam as tarefas em cada um dos *DataNodes*. Assim é possível reiniciar tarefas, ou seja, os *NodeManagers* estão comunicação periódica com o *ResourceManager* e caso

4 https://github.com/fluent/fluent-plugin-webhdfs

aconteça algum *timeout* durante essa comunicação, o *ResourceManager* assume que o *NodeManager* falhou e reatribui as tarefas a outro nó. Se a tarefa ainda estiver em fase de mapeamento, então, e após o *ResourceManager* reatribuir a tarefa, o novo *NodeManager* reexecuta a tarefa totalmente. No caso da tarefa já estar na fase de redução, o novo *NodeManager* reexecuta-a a redução.

NaneNode HA

NameNode HA, só começou a tornar-se uma realidade a partir da v2.* do *Hadoop*. Antes, o *NameNode* era SPOF e apesar de ser possível configurar um *NameNode* secundário ou de *backup*, não era possível uma transição entre nó Activo/Passivo eficiente. Nas versões anteriores, o *NameNode* de *backup* mantinha uma listagem das transações realizadas e caso o *NameNode* principal falhasse, este assumiria o seu lugar mas demoraria uma quantidade de tempo considerável até atingir um estado estável e que lhe permitisse assumir o controle.

Desde a versão 2.2.0, já é possível configurar uma arquitectura de nó Activo/Passivo. Para isso, ambos partilham de uma localização onde vão sendo escritas as transações e outras informações importantes que vão sendo lidas pelo *NameNode* passivo e assim permitir que este passe ao estado activo com o menor *downtime* possível, muitas das vezes de forma transparente para o utilizador.

Para a localização partilhada, é possível configurar a mesma através de Network File Storage (NFS) ou *QJM*. Apesar de ser mais fácil configurar através de NFS, este é SPOF e isso não é uma característica desejável para a solução requerida. Por essa razão, aconsenlha-se a configuração da localização através do Quorum Journal Manager (QJM), que funciona como um "jornal" distribuído entre algumas máquinas do *cluster* e que são denominadas de *JournalNodes*.

É possível configurar o modo de *failover* para manual ou automático. Para o modo manual, as ferramentas de HA do *Hadoop* disponibilizam um comando para executar a transição entre estados, nos *NameNodes*. Para executar essa transição, o comando é:

```
/opt/hadoop/bin/hdfs haadmin -failover failing\_namenode standby\_namenode
```

Para a execução deste *failover* manual é sempre necessário alguém ou algum mecanismo que inicie esta operação, caso contrário o *NameNode* passivo não passará ao estado ativo e em resultado, deixará de haver acesso ao *cluster*. O mecanismo para executar essa operação de transição pode ser um *script* ou, idealmente, uma ferramenta que permita a coordenação distribuída e tolerante a faltas. Com essas características em mente, o *Hadoop* permite o *failover* automático com recurso ao mecanismo de coordenação *Apache ZooKeeper*, que é também utilizado pelo *Apache Kafka*, como pode ser visto na Secção 2.2.4. O *ZooKeeper* permite então, através de um agente controlador de *failover* - *ZooKeeper Failover Controller* - em execução em cada um dos *NameNodes*, verificar a disponibilidade de cada um, assim como proceder a transição do estado, caso seja necessário. Para isto é ainda necessário um

conjunto de servidores *ZooKeeper*, idealmente num número igual ou superior a 3 e ímpar, para evitar empates no mecanismo de eleição.

Para análise, foi tido em conta a reação dos clientes às falhas do *NameNode*, o tempo de recuperação, ou seja, o tempo necessário até o *cluster* estar novamente acessível, assim como a reacção do próprio *cluster* ao novo estado.

Para falarmos sobre a reacção do cliente à falha de um *NameNode*, é necessário que este esteja configurado para HA, ou seja, que possua mecanismos capazes de detetar que o *NameNode* falhou, atráves de *heartbeats*, por exemplo, e passar a realizar os seus pedidos ao novo *NameNode*. Se o cliente colocar um pedido de leitura ao *NameNode* e este falhar, não há problema, pois basta que o cliente redirecione o pedido ao novo *NameNode*. No caso de uma escrita, se o *NameNode* falhar antes da escrita para ficheiro ter terminado e o cliente não conseguir comunicar ao novo *NameNode*, o ficheiro não será reconhecido no sistema.

Os clientes preparados para apenas um *NameNode*, sem mecanismo de HA, falham ou entram em modo de espera até o *NameNode* estar novamente disponível e passar ao estado activo. Existem clientes preparados para este tipo de situação, *e.g.* no caso se escritas, se as tentativas falharem um número pré-definido, então o cliente passa a armazenar os dados em ficheiros locais até o *NameNode* estar novamente disponível.

Uma nota importante, os clientes apenas utilizam o *NameNode* para conhecer a localização dos blocos que constituem os ficheiros que pretender ler ou escrever e para comunicar que um ficheiro foi escrito e fechado. O resto das operações é feita diretamente entre os clientes e os *DataNodes*.

O tempo de recuperação, a partir da versão 2.*, é relativamente rápido. As operações de escrita realizadas no *NameNode* em estado activo são também escritas para a localização partilhada, permitindo assim manter o *NameNode* em estado passivo actualizado. Desta forma, o tempo necessário para a transição do estado passivo para activo, passa pela detecção da falha que em modo automático ocorre por *timeout*, 5 segundos por predefinição, o *Zookeeper* proceder a eleição de um novo *NameNode* e executar o pedido de transição de estado. Para verificar este comportamento, foram introduzidas falhas baseadas em terminar, forçosamente, o processo responsável pelo *NameNode* - `kill -9 <namenode_pid>`. Durante as várias tentativas, a detecção e transição de estado ocorria sem problemas e de uma forma rápida, nunca ultrapassando um intervalo entre 1 e 2 minutos. Estes valores podem estar influenciados devido a dimensão da infraestrutura usada, que além de virtual pode ser considerada insignificante quando comparada com infraestruturas de produção.

À exceção dos mecanismos necessários a detecção e transição de estados, o *cluster* reage facilmente a falha de um *NameNode*, uma vez que os *DataNodes* estão configurados e contêm mecanismos que permitem suportar falhas dos *NameNodes*.

Se todos os *NameNodes* falharem, então o *cluster* deixará de estar acessível e não existe nenhum mecanismo que permita mitigar esta situação.

DataNodes

De forma análoga ao que foi feito anteriormente para os *NameNodes*, será também feita uma análise baseada na reação dos clientes, no tempo necessário à recuperação e na reacção do *cluster*.

Uma vez que as operações de escrita e leitura são feitas diretamente para os *DataNodes*, a falha de um deles tem efeitos mais difíceis de mitigar. No caso de uma escrita e como foi dito anteriormente, se o ficheiro a ser escrito não for fechado no HDFS e comunicado ao *NameNode*, este ficheiro não será reconhecido pelo sistema. Resta ao cliente ter a capacidade de "superar" esta situação e tentar novamente, reenviando o pedido de escrita ao *NameNode*. Apesar desta informação, retirada da documentação oficial do *Hadoop*, foi possível constatar que um ficheiro continuava a ser escrito, mesmo após terem sido introduzidas falhas nos *DataNodes*. O ficheiro parava de ser escrito quando não restavam mais *DataNodes* disponíveis. Este resultado pode ter origem na forma como os pedidos e as operações de escrita estavam a ser feitas. Os pedidos estavam a ser feitos ao *WebHDFS*, serviço REST, e eram pedidos de operações de *append* a um ficheiro já existente. Isto permitia que o ficheiro não ficasse irreconhecível para o sistema.

Uma vez que os *DataNodes* e oo *NameNodes* estão em constante comunicação, o *NameNode* consegue detetar falhas, proceder a verificação dos fatores de replicação dos blocos e atuar consoante necessário. Esta atuação consiste em detetar blocos que não estejam replicados consoante o factor e agendar a sua replicação para um novo *DataNode*. Durante estas operações, o *NameNode* tem em atenção o balanceamento de carga entre os *DataNodes*.

Se forem executadas operações no *cluster*, *e.g.* eliminação de ficheiros, enquanto um dos *DataNodes* está indisponível, essas operações serão enviadas assim que o nó voltar a estar disponível e executadas se o nó possuir blocos envolvidos na operação.

Quanto ao tempo de recuperação, este depende da quantidade de dados que estariam armazenados nos *DataNodes* que falharam. Esse tempo está também dependente da carga a que o *cluster* está sujeito no momento da falha.

4.2.2 *Curto-Prazo*

Distribuição e Escalabilidade

Como já foi falado anteriormente, o *Solr* não foi criado com a intenção de ser uma solução de pesquisa distribuída, ao contrário do *Elasticsearch*. Por essa razão é bastante mais fácil criar, inicializar e configurar um *cluster* recorrendo ao *Elasticsearch* do que ao *Solr*.

Para criar um *cluster* de pesquisa distribuída recorrendo ao *Elasticsearch*, basta, após instalar a solução, editar o ficheiro de configuração `elasticsearch.yml` e descomentar o campo `cluster.name`. O *Elasticsearch* possui um mecanismo de *Auto-Discovery* que permite descobrir e configurar um *cluster* automaticamente, se as máquinas estiverem configuradas na mesma rede. Apesar de o *Solr* já ser distribuído com um *script* que facilita a criação de um *cluster* através de um conjunto de perguntas realizadas, via interface textual, ao responsável pela infraestrutura, ainda não é uma solução facilmente configurável e comparável com o *Elasticsearch*. A acrescentar a isto, para um ambiente de produção não é aconselhável a utilização da implementação interna do *Zookeeper*, presente nas distribuições actuais do *Solr*, o que obriga a esforço extra de configuração. Já o *Elasticsearch* possui um mecanismo próprio, o *Zen Discovery*, que dispensa esse tipo de configuração adicional.

Um dos componentes básicos do *Lucene* é o *shard*. Quando um indíce é criado, este é dividido e cada uma dessas divisões é denominada de *shard*. Estes *shards* serão depois distribuídos pelas máquinas do *cluster*, recorrendo a um algoritmo de balanceamento de carga. O número de *shards* a criar, pode ser definido no momento da criação do indíce, ou então será usado o valor por predefinição. Uma das vantagens do *Solr* é exactamente aqui, permitindo o aumento no número de *shards* de um indíce, *a posteriori*, enquanto que o *Elasticsearch* não suporta esta característica nativamente, sendo necessário recorrer ao re-indexamento dos documentos para um indíce com um número superior de *shards*. [7]

A *Loggly*, empresa enunciada no Capítulo 2, colocou sobre availação estas duas soluções e optou por utilizar o *Elasticsearch* como solução de indexamento. Segue o seguinte texto, uma publicação no *blog* da empresa [2], sobre os testes realizados a ambas as soluções:

> *"So we spun up large clusters of each system and proceeded to destructively test them both by overloading them with data and by forcibly shutting down nodes to see how the systems behaved under duress. Think Chaos Monkey, where we were the monkeys. With Solr, one of the biggest problems we found was that a lot of the new features that came out in SolrCloud were very immature, the Overseer/Queue management in particular. We had some real problems with stability running the "out of the box" SolrCloud distribution. In our testing, we saw total lock-ups in the cluster that could only be resolved with a complete reset. ES, on the other hand, sailed through this same testing without any*

unrecoverable failures. Yes, we could force ES to lose data. However, we understood exactly how that happened and were confident we could engineer our way out of those situations."

Esquema

Ambas as soluções apresentam um modelo *schemaless*, o que permite que se possa indexar novos tipos de campos sem necessitar de novas configurações, através de um mapeamento dinâmico do tipo de dados. O *Elasticsearch* possui um mapeamento dinâmico muito mais avançando do que o *Solr*, isto porque no caso do *Elasticsearch* não é sequer necessário realizar qualquer configuração adicional para tal, fá-lo-a nativamente. Já o *Solr*, precisa de alguma configuração adicional para o fazer e não possuí tanta liberdade como o *Elasticsearch*. Essa configuração adicional baseia-se na criação de um, ou vários, prefixos ou sufixos, e na atribuição de um tipo de dados. Por exemplo, ao adicionar a seguinte linha de configuração ao *Solr*,

```
<dynamicField name="*_i" type="integer" indexed="true"/>
```

e, ao adicionar um novo campo ao documento a indexar, *e.g. sensor_i*, este assumirá que o novo campo será um inteiro e deverá ser indexado. Caso o novo campo tenha o nome *sensor_str*, o *Solr* não irá reconhecer o tipo de dados, ao contrário do *Elasticsearch*.

Quanto à criação de esquemas, o *Elasticsearch* possuí a capacidade de inferir automaticamente o esquema dos documentos que queremos indexar, enquanto que no *Solr* será necessário configurar o esquema dos documentos através de um ficheiro denominado de *schema.xml*. O *Elasticsearch* também permite a configurações de esquemas, ou *mappings* como é denominado nesta solução, através de pedidos REST, onde é ainda possível a criação de *templates* para uso futuro e versionamento dos *mappings*, caso seja necessário proceder à alteração do esquema e posterior re-indexamento dos documentos. Recomenda-se a criação de esquemas *a priori*, passando a existir um controlo bastante superior sobre os dados e é ainda possível a utilização de filtros e outras operações durante o indexamento.

Como foi referido no Capítulo 2, quando foram apresentadas as soluções, ambas possuem comportamentos diferentes no que diz respeito ao aninhamento de dados nos documentos. O *Elasticsearch* permite esse aninhamento, ao contrário do *Solr*, que apenas permite estruturas sem aninhamento. Comparando, no *Elasticseark* é possível armazenar e indexar informação do tipo:

```
{
  time: {
    hour: "16",
    minute: "26",
```

```
    seconds: "12",
  },
  message: "HTTP STATUS 201 OK"
}
```

Enquanto que no *Solr* teremos algo do género:

```
{
  time_hour: "16",
  time_minute: "26",
  time_seconds: "12",
  message: "HTTP STATUS 201 OK"
}
```

No caso de se optar pelo *Elasticsearch* como solução e se o aninhamento for um requisito, será necessário ter em conta que as operações de actualização a valores que façam parte dos aninhamentos, serão mais lentas que as restantes.

Interrogação

As razões que levaram ao desenvolvimento destas soluções são diferentes e como tal, estas, aprensentam algumas diferenças significativas no que respeita a capacidade de interrogação sobre os dados. Enquanto o *Solr* foi construído com o propósito de resolver problemas relacionados com Information Retrieval (IR), o *Elasticsearch* foi desenvolvido para satisfazer as necessidades de uma plataforma extremamente escalável. Por esta razão, o *Solr* possui uma API muito mais complexa que o *Elasticsearch*. Mas, e no intuito desta dissertação, não é necessário uma API que permita operações demasiado complexas sobre os dados, uma vez que o mais importante será termos os dados acessíveis em (quase) tempo-real e, que a escalabilidade seja de grande facilidade.

Entre as duas soluções, a sintaxe usada para interrogar os dados é diferente. O *Solr* utiliza um conjunto de pares chave/valor e símbolos lógicos - “AND”, “OR”, etc. No segundo caso, *Elasticsearch*, a interrogação é feita através de um documento JSON que consiste também num conjunto de pares chave/valor.

```
curl -XGET http://localhost:9200/ids/_search -d '{
  "query" : {
    "term" : { "proto": "TCP" }
  }
}'
```

Figura 16.: Interrogação com recurso a *Term*, em *Elasticsearch*

```
curl -XGET http://localhost:9200/ids/_search -d '{
  "query_string" : {
    "query": "(proto:TCP OR proto:UDP)
              AND
              (dest_port:443 OR desp_port:80)"
  }
}'
```

Figura 17.: Interrogação com recurso a *Query String*, em *Elasticsearch*

```
curl 'http://localhost:8983/solr/select?q=(
            (proto:TCP OR proto:UDP)
            AND
            (dest_port:443 OR desp_port:80)
          )&wt=json&indent=true'
```

Figura 18.: Interrogação simples em *Apache Solr*

Através da análise das figuras, 16, 17 e 18, que contêm interrogações às duas soluções em avaliação, é possível verificar que o *Elasticsearch* permite também criar interrogações à semelhança do *Apache Solr*. Isto, deve-se ao facto de ambas as soluções serem baseadas no *Apache Lucene*. O *Elasticsearch* permite então interrogar através da sua DSL e da sintaxe de interrogação do *Apache Lucene*.

Comunidade

Apesar do *Elasticsearch* ser relativamente recente, ver figura 10, este apresenta um desenvolvimento e um interesse, por parte da comunidade, cada vez maior. Como o *Apache Solr* é uma solução mais antiga no mercado, possui uma base de utilizadores superior, mas que está a ser rapidamente ultrapassada, devido a adopção natural do *Elasticsearch* pelos novos utilizadores.

Pode-se então concluir em termos de comunidade, ambas as soluções possuem uma larga base de utilizadores activos e existe já uma grande quantidade de empresas que disponibilizam tanto o *Apache Solr* como o *Elasticsearch* como SaaS, assim como suporte comercial a ambas as plataformas.

Análise de desempenho

Para esta análise foi criado um ambiente de testes composto por três máquinas, duas para a instalação e configuração de uma das soluções e uma terceira, utilizada para geração e envio de pedidos.

O *software* utilizado para a geração de pedidos foi o *Vegeta*[5] que é uma ferramenta *open-source* desenvolvida para testes de carga sobre *HTTP*. Esta ferramenta permite uma geração de pedidos constante, essencial para uma avaliação concreta das soluções. Apesar da geração constante, o número de pedidos gerados depende também da capacidade de Input/Ouput (I/O) da máquina responsável pela geração dos mesmos pedidos.

No contexto desta dissertação, o mais importante é sem dúvida a análise de desempenho para pedidos de indexação de eventos, uma vez que é necessário uma solução que permita um elevado desempenho na ingestão de informação, ou seja, capaz de receber e tratar uma grande quantidade de pedidos de indexamento.

Após a execução de um conjunto de testes, que podem ser verificados nos Apêndices B.1 e B.2, resultaram as Tabelas 4, 5, 6 e 7, que resumem o conjunto dos resultados através da análise da latência média por pedido e a percentagem de sucesso no indexamento de novos eventos.

Nas tabelas seguintes, as linhas e colunas representam valores distintos, a colunas representam o número de pedidos gerados, pelo *Vegeta*, por segundo, enquanto que as linhas representam o número de *workers* iniciais, ou clientes, usados durante os testes.

Nas Tabelas 4 e 5 é medida a latência média dos pedidos, em milisegundos, durante dois intervalos de tempo distintos, 60 e 600 segundos ou 1 e 10 minutos, respectivamente. Para ambos os intervalos de tempo, o ElasticSearch (ES) demonstra uma latência média entre 1 - 2 *ms*, enquanto que o *Solr* mostra alguma inconsistência, com valores que rondam entre 1 - 150 *ms*. É possível verificar ainda que a latência aumentam, sempre que o número inicial de clientes aumenta de 1 para 10.

Perante as Tabelas 6 e 7, que refletem a percentagem de sucesso no processamento de pedidos, é possível verificar rapidamente que o ES obteve sempre uma taxa de sucesso de 100%, enquanto que o *Solr* possuí uma enorme variação nos resultados, assim como apresentou nos resultados de latência média, das Tabelas 4 e 5.

Na Tabela 6 é possível verificar, analogamente a Tabela 4, que assim que o número inicial de clientes sobe de 1 para 10, a taxa de sucesso, no *Solr*, baixa num valor compreendido entre os 10% e os 15%. Como resposta aos pedidos, é apresentando, em muitos casos, o código de erro HTTP 503. Isto significa que o servidor não estava disponível ou estava em sobrecarga e, uma vez que durante a execução dos testes, as máquinas e os serviços em análise estavam sob vigilância, conclui-se então que o *Solr* estava em sobrecarga.

A Tabela 7, mostra este problema de forma mais problemática, uma vez que o *Solr* conseguiu obter taxas de sucesso de 0% assim que o número de pedidos feitos ao serviço aumentou de 100 para 400 e, comportando-se da mesma forma tanto para um, como dez clientes. Quando o número de pedidos era

5 https://github.com/tsenart/vegeta

de 100 e o número de clientes iniciais era de 10, verificou-se uma descida na taxa de sucesso a rondar os 15%, analogamente a Tabela 6. Assim que o número de pedidos gerados aumentou, aumentaram também o número de *timeouts* e códigos de erro HTTP 500 - erro interno no servidor. Estes erros faziam-se acompanhar por duas mensagens de error distintas, "*Connection reset by peer*" e "*Too many open files*". Para o segundo caso, é fácil corrigir o problema, aumentando o limite de *file descriptors* por processo, recorrendo ao comando *UNIX* `ulimit`. No caso do primeiro erro, "*Connection reset by peer*", já é mais complicado, pois é resultante do próprio *Solr* que fecha as ligações, respondendo ao cliente, quando sabe que não têm capacidade para aguentar uma nova ligação.

		# pedidos / segundo					
		100		400		1600	
# workers	1	ES:	1.46595	ES:	1.16619	ES:	1.42665
		SOLR:	45.05040	SOLR:	61.43649	SOLR:	35.15187
	10	ES:	1.38613	ES:	1.16061	ES:	1.91196
		SOLR:	149.98593	SOLR:	133.03450	SOLR:	144.17273

Tabela 4.: Latência média, em *ms*, durante uma execução de 60 segundos a ambas as soluções de indexamento.

		# pedidos / segundo					
		100		400		1600	
# workers	1	ES:	1.69075	ES:	1.09480	ES:	1.24676
		SOLR:	28.66156	SOLR:	1.61990	SOLR:	1.85752
	10	ES:	1.89151	ES:	1.13722	ES:	1.21238
		SOLR:	141.04459	SOLR:	23.72315	SOLR:	12.47510

Tabela 5.: Latência média, em *ms*, durante uma execução de 600 segundos a ambas as soluções de indexamento.

		# pedidos / segundo					
		100		400		1600	
# workers	1	ES:	100%	ES:	100%	ES:	100%
		SOLR:	100%	SOLR:	100%	SOLR:	100%
	10	ES:	100%	ES:	100%	ES:	100%
		SOLR:	89.92%	SOLR:	89.35%	SOLR:	86.98%

Tabela 6.: Percentagem de sucesso nos pedidos durante uma execução de 60 segundos a ambas as soluções de indexamento.

# workers	# pedidos / segundo	100	400	1600
1	ES:	100%	100%	100%
	SOLR:	100%	0%	0%
10	ES:	100%	100%	100%
	SOLR:	84.10%	0%	0%

Tabela 7.: Percentagem de sucesso nos pedidos durante uma execução de 600 segundos a ambas as soluções de indexamento.

Depois da análise feita anteriormente, o ES parece ser a melhor escolha para o objectivo deste documento, por um conjunto de factores que são necessários a construção de um modelo capaz de suportar as necessidades atuais e futuras, quer em termos de ingestão de informação, quer de escalabilidade.

Apesar de não possuir um conjunto tão amplo de opções de interrogação como o *Solr*, o ES demonstra características muito superiores a nível de distribuição, esquema e desempenho. Além disso, possuí uma comunidade cada vez maior e mais participativa que aumenta a velocidade de desenvolvimento da solução, incluíndo as propriedades de interrogação, podendo mesmo ser possível ultrapassar as capacidades de interrogação do *Solr* em muito pouco tempo.

5

CONCLUSÃO

Foi necessário compreender os problemas na gestão de eventos e informações de segurança, antes de avançar para o desenvolvimento desta dissertação. Este processo foi complicado, uma vez que estamos a falar de uma área extremamente abrangente e com bastantes ramificações, na heterogeneidade dos eventos, volume, dispersão no espaço geográfico e temporal, processamento em tempo-real, *etc*. Todas estas áreas são de extrema importância, mas díficeis de contemplar num único documento. Dessa forma, compreendeu-se que o maior desafio, inicial, seria o desenho de um modelo que fosse capaz de suportar os requisitos de volume e dispersão atuais, mas que possua escalabilidade como característica, de forma a suportar necessidades futuras. Desse desenho, resultou o modelo representado pela Figura 8, que representa um modelo que se supõe preencher os requisitos necessários a um sistema deste tipo.

Obteve-se então, um conjunto de tecnologias *opensource* que em conjunto permitem já colectar os dados, efectuar algum processamento básico (*e.g.* filtrar, obter informações geográficas através do endereço IP, métricas, *etc.*) e armazenar os dados, para consulta rápida (ver Secção 2.3.1) ou para análise histórica (ver Secção 2.3.2). Para a escolha das soluções, houve um cuidado adicional com o seu factor de escalabilidade, desempenho e envolvimento da comunidade no continuo desenvolvimento das soluções, presente e futuro.

A medida que esta dissertação foi sendo desenvolvida, algumas tecnologias que estão expostas neste documento, estavam a ser configuradas na infraestrura de produção da empresa Eurotux Informática, *S.A.* (ver Secção 1.3). Essas soluções permitam, recolher os eventos gerados por um NIDS, onde passava todo o tráfego de rede, e enviá-los através do *fluentd* (ver Secção 2.2.3) para serem indexados pelo *ElasticSearch* (ver Secção 2.3.1). Isto permitia aos responsáveis pela monitorização da rede detectar eventos possivelmente maliciosos atráves de uma *dashboard* desenvolvida em *Kibana 4* (ver Secção 2.5.1), assim como realizar operações de interrogação através dessa mesma interface. A partir daqui tentou-se armazenar o trafégo de rede, para análises futuras e análise de trechos do mesmo, relacionados com os eventos gerados pelo NIDS, mas estavam a ser gerados ficheiros com um tamanho

de 2*GB* a cada minuto, sobrecarregando a máquina a cada tentativa de localizar um trecho desse tráfego.

Ficou então claro que para implementar um modelo deste género, é necessário uma infraestrutura dedicada ao mesmo, algo que muitas pequenas ou médias empresas não serão capazes de suportar. Será então, idealmente, uma solução direcionada a grandes empresas ou empresas que pretendam a comercialização deste tipo de soluções como SaaS.

5.1 TRABALHO FUTURO

Como foi dito na secção anterior, esta é uma área extremamente abrangente e muitas das características necessárias a um sistema SIEM para ambientes de *Big Data* ficaram por desenvolver, ficando então destinadas a trabalho futuro.

PROCESSAMENTO em (quase) tempo-real dos eventos, entre o transporte e a indexação e armazenamento. Este mecanismo de processamento, ver Secção 2.4.2, permitirá entre outras coisas, a normalização dos eventos, alarmística e correlacionamento.

NORMALIZAÇÃO dos eventos de segurança. Este é sem dúvida um ponto bastante importante na construção de um SIEM para o futuro. Como referido na Secção 1.2, o número elevado de fontes destes dados, leva também a um número elevado de formatos diferentes e que devem ser tidos em conta no processamento, indexação e armazenamento dos dados. Com isto, pretende-se desenvolver uma solução capaz de processar todos os eventos, nos seus diversos formatos, e normalizar os eventos com o objectivo de criar uma estrutura idêntica, independentemente da fonte.

ALARMÍSTICA recorrendo a algoritmos estátisticos que permitam a deteção de anomalias na infraestrutura, ainda durante o processo de processamento anterior a indexação e armazenamento.

CORRELACIONAMENTO dos dados em quase tempo-real, através de mecanismos de processamento em *stream*, ver Secção 2.4.2. Com isto pretende-se, *e.g.*, a detecção de padrões de ataque através do correlacionamento de dados de múltiplas *firewalls*, HIDS e NIDS.

ANÁLISE HISTÓRICA dos dados em arquivo, para a detecção de possíveis vulnerabilidades a que a infraestrutura possa ter sido exposta no passado, através de vulnerabilidades divulgadas ao

público, recentemente. Para esta análise histórica seria necessário guardar em arquivo todo o tráfego de rede da infraestrutura para que fosse, depois, possível processar esse tráfego em *batch*, ver Secção 2.4.1. Além disso, e recorrendo também ao processamento em *batch*, permitir a geração de relatórios semestrais e anuais sobre a infraestrutura.

TRANSPORTE dos eventos com recurso a novos paradigmas. Como foi referido na Secção 4.1, o *Apache Kafka* é certamente uma solução de transporte que fará parte da *stack* de um grande número de infraestruturas, num futuro próximo. Pelo facto de ser uma solução desenhada para ambientes de alto *throughput* e de ser altamente tolerante a faltas, é uma evolução natural nos mecanismos de transporte do modelo proposto na Secção 3.1.4.

BIBLIOGRAFIA

[1] Christian Abad, Jed Taylor, Cidgem Sengul, William Yurcik, Yuanyuan Zhou, and Ken Rowe. Log correlation for intrusion detection: A proof of concept.

[2] Jon Gifford. Why Loggly Chose Elasticsearch for Reliable, Scalable Log Management, 2014. URL https://www.loggly.com/blog/loggly-chose-elasticsearch-reliable-scalable-log-management/.

[3] Karen Kent and Murugiah Souppaya. Guide to computer security log management, 2006. URL http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf.

[4] Ritchie S. King. *Visual Storytelling with D3: An Introduction to Data Visualization in JavaScript*. Addison-Wesley Professional, 2014. ISBN 978-0-13-343965-6.

[5] Umesh Hodeghatta Rao and Umesha Nayak. *The InfoSec Handbook*. Apress Open, 2014.

[6] David Swift. A practical application of sim/sem/siem automating threat identification, 2006.

[7] Ryan Tabora. Think Big Analytics - Solr vs ElasticSearch, 2013. URL https://thinkbiganalytics.com/solr-vs-elastic-search/.

A

TRANSPORTE

A.1 FLUENTD

A.1.1 *Instalação e Configuração*

Ambiente de Testes

Para este ambiente de avaliação, foram usadas duas máquinas físicas, ligadas entre si através de um *router* e configuradas na mesma rede local. As máquinas tinham as seguintes características:

Cliente:

- Intel Core 2 Duo P8700 @ 2.53GHz
- 4GB RAM
- 160GB HDD
- Centos 7 x86_64 *Minimal Install*

Servidor:

- Intel Pentium Dual T2390 @ 1.86GHz
- 3GB RAM
- 250GB HDD
- Centos 7 x86_64 *Minimal Install*

Preparação inicial

Para efeitos de testes, foram instaladas algumas ferramentas úteis a configuração da máquina e foi ainda desactivado o Security-Enhanced Linux (SELinux) e a *firewall* do sistema, o *firewalld*.

Todas as máquinas de testes, possuíam todas as actualizações do sistema à data dos testes.

```
$ yum update -y
$ yum install net-tools vim wget git
```

```
$ sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux
$ chkconfig firewalld off
$ reboot
```

Ruby 2 & Bundler

Para instalar o *Ruby* foi utilizado o Ruby Version Manager (RVM)[1]. Foram seguidos os seguintes passos:

```
$ gpg --keyserver hkp://keys.gnupg.net --recv-keys 409
    B6B1796C275462A1703113804BB82D39DC0E3
$ \curl -sSL https://get.rvm.io | bash -s stable --ruby
$ source /etc/profile.d/rvm.sh
$ rvm install 2.0.0
$ rvm use 2.0.0 --default
$ ruby --version
$ gem install bundler --no-ri --no-doc
```

Fluentd

Apesar de ser possível utilizar os binários de instalação, *e.g.* RPM ou DEB, disponibilizados pela *Treasure Data*, foi feita uma instalação através da *source* do projecto. Os passos seguintes mostram como foi obtida a *source*, compilada e instalada nas máquinas:

```
$ git clone https://github.com/fluent/fluentd.git
$ cd fluentd
$ bundle install && bundle exec rake build
$ gem install pkg/fluentd-*.gem
```

Para inicializar uma directoria para o *Fluentd*, foi executada a seguinte linha:

```
$ fluentd --setup ./fluent
```

1 Ferramenta de linha de comandos que permite a instalação e gestão de ambientes de trabalho *Ruby*.

Da execução do comando anterior resultou uma directoria na localização actual, que continha uma 2ª directoria para *plugins* e um ficheiro exemplo de configuração, `fluent.conf`. Para verificar se a solução estava devidamente instalada, executou-se os seguintes comandos, assim como estão descritos na página do projecto[2]:

```
$ fluentd −c ./fluent/fluent.conf −vv &
$ echo '{"json":"message"}' | fluent−cat debug.test
```

Como resultado, o *Fluentd* imprimiu para o ecrã a seguinte informação:

```
2015-10-23 13:46:44 +0100 debug.test: {"json":"message"}
```

A.1.2 *Configuração e execução dos testes*

Antes de iniciar os testes de desempenho foi necessário um utilitário para gerar os eventos. Para este efeito, foi utilizado o *Dummer* e este pode ser encontrado, publicamente, em https://github.com/sonots/dummer. Esta ferramenta pode ser instalada através da fonte ou por intermédio do utilitário *gem*[3]. Após a instalação do *Dummer* foi criado um ficheiro de configuração com o seguinte conteúdo:

```
configure 'bench' do
  output "dummy.log"
  delimiter "\t"
  rate 10000
  tag type: :string, any: %w[sensor2.prod sensor3.web]
  field :msg_count, type: :integer, countup: true, format: "%04d"
  field :timestamp, type: :integer, countup: true, format: "2015-10-05
                                                            00:23:52.%09d"
  field :event_type, type: :string, value: "alert"
  field :src_ip, type: :string, any: %w[192.168.1.41 192.168.1.23 192.168.1.51]
  field :src_port, type: :integer, range: 10000..20000
  field :dest_ip, type: :string, value: "192.168.1.101"
  field :dest_port, type: :integer, value: 443
```

2 http://docs.fluentd.org/articles/install-from-source
3 Gestor de pacotes para *Ruby*.

```
  field :proto, type: :string, value: "TCP"
  field :alert_action, type: :string, value: "allowed"
  field :alert_gid, type: :integer, range: 1..10
  field :alert_signature_id, type: :integer, range: 10000..30000
  field :alert_rev, type: :integer, range: 1..10
  field :alert_signature, type: :string, value: "SERVER-OTHER OpenSSL
                                        TLSv1.1 heartbeat read overrun attempt"
  field :alert_category, type: :string, value: "Attempted Information leak"
  field :alert_severity, type: :integer, range: 1..3
end
```

Após a instalação e configuração do *Dummer*, foram configurados as duas instâncias do *Fluentd*, nas respectivas máquinas. As configurações usadas nesta avaliação, foram as seguintes:

Cliente:

```
<source>
  type tail
  path /path/to/dummy.left
  pos_file /path/to/_dummy_log.pos
  format none
  tag fluentd_bench
</source>

<match fluentd_bench>
  type forward
  flush_interval 0
  try_flush_interval 1
  buffer_chunk_limit 1m
  buffer_queue_limit 64
  num_threads 1
  <server>
    host <server ip>
    port 24224
  </server>
</match>
```

Servidor:

```
<source>
  type forward
  port 24224
</source>

<match fluentd_bench>
  type flowcounter_simple
  unit second
</match>
```

As configurações para o cliente foram adaptadas a partir de uma configuração idêntica e com o mesmo propósito, utilizada por um dos membros da comunidade *Fluentd* e, pode ser encontrada em https://github.com/fluent/fluentd-benchmark. Neste repositório existe uma ligação para um *plugin* de medição do fluxo de dados, *flowcounter-simple*[4], que foi também utilizado nesta avaliação. Para a instalação do mesmo, recorreu-se ao seguinte comando:

```
$ gem install fluent-plugin-flowcounter-simple --no-ri --no-rdoc
```

4 https://github.com/sonots/fluent-plugin-flowcounter-simple

Para iniciar os testes, executaram-se os seguintes comandos, pela mesma ordem com que são aqui apresentados:

Servidor:

```
$ fluentd −c /path/to/server.conf −−log /path/to/fluentd_server.log
```

Cliente:

```
$ fluentd −c /path/to/client.conf −−log /path/to/fluentd_client.log &
$ dummer −c /path/to/bench.conf
```

Uma vez se usou o *plugin* `flowcounter-simple`, o servidor irá apresentar, todos os minutos, o número de pedidos que estão a ser processados por minuto. Essa informação será idêntica a:

```
<timestamp> [info]: plugin:<plugin_name> count:288931 indicator:num unit:minute
```

Todas as alterações nas configurações, entre avaliações, foram realizadas manualmente e as alterações foram ao nível do número de *threads* e número de eventos gerados por segundo, no cliente. Estas configurações variaram entre:

 # Threads

- 1
- 10

 # Eventos/seg

- 5000
- 10000
- 50000

A.2 LOGSTASH

A.2.1 *Instalação e Configuração*

Ambiente de Testes

O ambiente de testes foi construído de forma análoga ao ambiente para a avaliação do *Fluentd*. Por essa razão o ambiente de testes pode ser consultado no Apêndice A.1.1.

Preparação inicial

A preparação inicial da máquina foi feita de forma análoga a preparação feita para o *Fluentd*. Essa preparação pode ser vista no Apêndice A.1.1

Oracle Java 7u71 JDK

Foi feito o download do RPM para a instalação do *Java*, a partir da página *web* da *Oracle*[5]. Após o *download* do pacote e do seu envio para a máquina de testes, foi necessário executar o seguinte conjunto de comandos para a configuração do *Java*:

```
$ rpm -Uvh jdk-7u71-linux-x64.rpm
$ alternatives --install /usr/bin/java java /usr/java/latest/jre/bin/java 200000
$ alternatives --install /usr/bin/javaws javaws /usr/java/latest/jre/bin/javaws 200000
$ alternatives --install /usr/bin/javac javac /usr/java/latest/bin/javawsc 200000
$ alternatives --install /usr/bin/jar jar /usr/java/latest/bin/jar 200000
$ echo export "JAVA_HOME=\"/usr/java/latest\"" >> ~/.bashrc
$ source ~/.bashrc
```

Logstash

De forma análoga ao que aconteceu com a instalação do *Fluentd*, ver Apêndice A.1.1, optou-se pela instalação do *Logstash* através da fonte, como demonstra o seguinte conjunto de passos:

5 http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

```
$ wget https://download.elastic.co/logstash/logstash/logstash-1.5.4.tar.gz
$ tar xzf logstash-1.5.4.tar.gz
```

Para verificar que o mesmo está instalado correctamente:

```
$ cd logstash-1.5.4
$ bin/logstash -e 'input { stdin { } } output { stdout {} }'
```

Se o *Logstash* iniciar correctamente, será apresentada uma mensagem de sucesso, aí, introduzindo uma mensagem e pressionando *return*, foi devolvida a seguinte mensagem:

```
2015-10-24T16:16:05.472Z 127.0.0.1 Logstash Teste
```

A.2.2 *Configuração e execução dos testes*

Como aconteceu na avaliação do *Fluentd*, ver Apêndice A.1.2, foi necessário utilizar o *Dummer* para a geração do eventos.

Após a instalação e configuração do *Dummer*, foram configurados as duas instâncias do *Logstash*, nas respectivas máquinas. As configurações usadas nesta avaliação, foram as seguintes:

Servidor:

```
input {
  tcp {
    mode => "server"
    codec => "line"
    port => 24224
  }
}

filter {
  metrics {
    meter => "events"
    add_tag => "metric"
    flush_interval => 60
    rates => [1]
  }
}

output {
  if "metric" in [tags] {
    stdout{
      codec => line {
        format => "rate: %{events.rate_1m}"
      }
    }
  }
  null {}
}
```

Cliente:

```
input {
  file {
    path => "/path/to/dummy.log"
    start_position => beginning
  }
}
```

```
output {
  tcp {
    mode => "client"
    host => "<server IP>"
    port => 24224
    codec => "line"
    workers => 1
  }
}
```

Para inicializar o teste, executaram-se os seguintes comandos, pela mesma ordem com que são aqui apresentados:

Servidor:

```
$ /path/to/logstash-1.5.4/bin/logstash -f server.conf
```

Cliente:

```
$ /path/to/logstash-1.5.4/bin/logstash -f client.conf &
$ dummer /path/to/bench.conf
```

Todas as alterações de configurações entre avaliações seguiram o mesmo padrão das avaliações realizadas ao *Fluentd* (ver Apêndice A.1.2).

B

ARMAZENAMENTO

B.1 ELASTICSEARCH

B.1.1 *Instalação e Configuração*

Ambiente de Testes

A máquina virtual onde foi configurado o ES, tinha as seguintes características:

- 2 vCPUs
- 4GB RAM
- 40GB HDD
- Centos 7 x86_64 minimal install

Além desta máquina, foi usada uma segunda máquina de onde eram lançados os pedidos. Esta segunda estava configura numa rede distinta da primeira, para simular um ambiente real e possuía as seguintes características:

- 1 vCPUs
- 2GB RAM
- 20GB HDD
- Centos 7 x86_64 minimal install

Preparação inicial

Para efeitos de testes, foram instaladas algumas ferramentas úteis a configuração da máquina e foi ainda desactivado o SELinux e a *firewall* do sistema, o *firewalld*.

Todas as máquinas de testes, possuíam todas as actualizações do sistema à data dos testes.

```
$ yum update -y
$ yum install net-tools vim wget
$ sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux
$ chkconfig firewalld off
$ reboot
```

Oracle Java 7u71 JDK

Foi feito o download do RPM para a instalação do *Java*, a partir da página *web* da *Oracle*[1]. Após o *download* do pacote e do seu envio para a máquina de testes, foi necessário executar o seguinte conjunto de comandos para a configuração do *Java*:

```
$ rpm -Uvh jdk-7u71-linux-x64.rpm
$ alternatives --install /usr/bin/java java /usr/java/latest/jre/bin/java 200000
$ alternatives --install /usr/bin/javaws javaws /usr/java/latest/jre/bin/javaws 200000
$ alternatives --install /usr/bin/javac javac /usr/java/latest/bin/javawsc 200000
$ alternatives --install /usr/bin/jar jar /usr/java/latest/bin/jar 200000
$ echo export "JAVA_HOME=\"/usr/java/latest\"" >> .bashrc
$ source .bashrc
```

ElasticSearch

O ES disponibiliza um repositório para sistemas baseados em *Red Hat*, o que facilitou a sua instalação.
A partir do serguinte conjunto de comandos, é possível verificar essa facilidade, que engloba a configuração do repositório, instalação, execução e teste do serviço:

```
$ rpm --import http://packages.elasticsearch.org/GPG-KEY-elasticsearch
$ cat > /etc/yum.repos.d/elasticsearch.repo <<_EOF_
```

1 http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

```
\ [elasticsearch-1.4]
\ name=Elasticsearch repository for 1.4.x packages
\ baseurl=http://packages.elasticsearch.org/elasticsearch/1.4/centos
\ gpgcheck=1
\ gpgkey=http://packages.elasticsearch.org/GPG-KEY-elasticsearch
\ enabled=1
_EOF_
$ yum install elasticsearch
$ /bin/systemctl daemon-reload
$ /bin/systemctl enable elasticsearch.service
$ /bin/systemctl start elasticsearch.service
$ curl -XGET <server IP>:9200
```

Vegeta HTTP load test tool

A instalação do *Vegeta* foi bastante simples, bastou apenas descarregar o binário, em arquivo `.tar.gz`, e descomprimir o mesmo como mostra o seguinte conjunto de comandos:

```
$ wget https://github.com/tsenart/vegeta/releases/download/v5.5.0/vegeta-linux-amd64.
    tar.gz
$ tar xzf vegeta-linux-amd64.tar.gz
$ chmod +x vegeta
$ ./vegeta
```

Pode ainda ser necessário atribuir permissões de execução ao binário, como foi o caso, daí o execução do comando `chmod`[2].

B.1.2 *Configuração e execução dos testes*

Antes de executar os testes, foi necessário criar três ficheiros, dois utilizados pelo *Vegeta* e um terceiro, em *shell script*, que foi usado para "comandar" e automatizar a execução dos testes.

2 Comando *UNIX* usado para a alteração de permissões de directórios e ficheiros.

Para o *Vegeta* foram criados os ficheiros `ids_body.json` e `es_target.txt`. O `ids_body.json` disponha de um evento, exemplo, gerado por um NIDS, num ambiente de produção[3]. O ficheiro continha o seguinte documento JSON:

```
{
  "timestamp": "2014-04-15T23:32:11.736275-0600",
  "event_type": "alert",
  "src_ip": "10.16.1.11",
  "src_port": 49719,
  "dest_ip": "192.168.88.3",
  "dest_port": 443,
  "proto": "TCP",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 30524,
    "rev": 1,
    "signature": "SERVER-OTHER OpenSSL TLSv1.1 heartbeat read overrun attempt",
    "category": "Attempted Information Leak",
    "severity": 2
  }
}
```

No caso do `es_target.txt`, esse era um ficheiro, denominado "*target*", para o *Vegeta*, ou seja, era constituído por um conjunto de valores que foram utilizadas durante a geração de pedidos. Estes valores incluíam o tipo de pedido HTTP, um *endpoint* para onde foram enviados os pedidos e a localização de um ficheiro com os dados que foram enviados. Este ficheiro era extremamente pequeno, apenas duas linhas:

```
POST http://<server IP>:9200/ids/alert
@/path/to/ids_body.json
```

O terceiro ficheiro, pode ser considerado um ficheiro auxiliar, utilizado apenas como auxiliar na execução automática dos testes. Neste ficheiro estava configurado a duração dos testes, o número de *workers* e o número de pedidos gerados por segundo. O ficheiro continha o seguinte:

```
#! /bin/bash
echo "[START ATTACKS]"
for d in 60s 300s 600s
do
    echo  "[DURATION - $d SECS]"
    for i in 100 200 400 800 1600
    do
```

3 Por motivos de segurança, os *IPs* foram alterados para outros, ditos, pseudo-aleatórios. Qualquer semelhança com *IPs* reais será pura coincidência.

```
        echo "[RATE - $i]"
        for w in 1 10 50 100 200
        do
            echo "[WORKERS - $w]"
            ./vegeta attack -duration=$d -rate=$i -workers=$w \
                -targets=es_target.txt > results_$d_$w_$i.bin
            sleep 10
        done
    done
done
echo "[END ATTACKS]\n"
```

Após toda a configuração acima, foi possível então executar os testes de desempenho e no final foram disponibilizados um conjunto de ficheiros com os resultados, que seguiam a nomenclatura `results_*_*_*.bin`. Para a extracção da informação textual, ou gráfica, dos dados escritos para estes ficheiros, foi necessário passá-los novamente ao *Vegeta*, desta vez com recurso à opção `report`. Por exemplo, para obter um sumário textual de um dos resultados, foi executado:

```
$ cat results_60_10_1600.bin | ./vegeta report
Requests [total]    96000
Duration [total, attack, wait]  1m4.995490909s, 1m4.994932458s, 558.451us
Latencies [mean, 50, 95, 99, max]  1.911958ms, 855.974us, 11.104415ms, 61.80339ms,
    61.80339ms
Bytes In [total, mean]   9024000, 94.00
Bytes Out [total, mean]   40704000, 424.00
Success  [ratio]    100.00%
Status Codes [code:count]   201:96000
Error Set:
```

Noutro caso, para obter um suporte gráfico, foi executado:

```
$ cat results_60_10_1600.bin | ./vegeta report -reporter=plot > results_60_10_1600.html
```

Depois do ficheiro `results_60_10_1600.html` ter sido gerado, foi possível abri-lo num *browser* e obter um *gráfico* como o da figura 19.

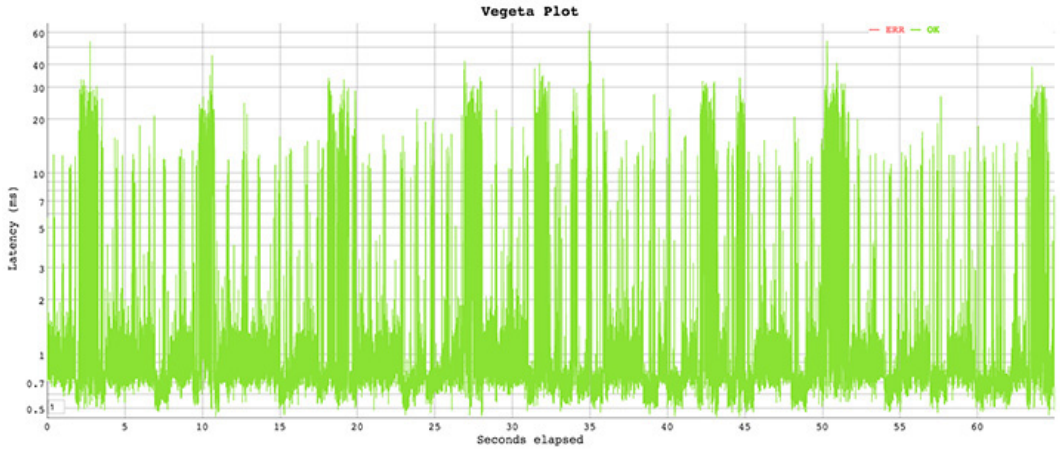

Figura 19.: Exemplo de um gráfico gerado pelo *Vegeta* apartir de um dos resultados.

Estes gráficos permitiram uma análise mais rápida dos resultados, em comparação com sumário textual acima representado.

B.2 APACHE SOLR

B.2.1 *Instalação e Configuração*

Ambiente de Testes

O ambiente de testes foi construído de forma análogo ao ambiente para a avaliação do ES. Por essa razão o ambiente de testes pode ser consultado no Apêndice B.1.1.

Preparação inicial

A preparação inicial da máquina foi feita de forma análoga a preparação feita para o ES. Essa preparação pode ser vista no Apêndice B.1.1

Oracle Java 7u71 JDK

A configuração do *Java* foi feita de forma análoga a configuração feita para o ES. Essa configuração pode ser vista no Apêndice B.1.1

Apache Solr

```
$ wget http://mirrors.fe.up.pt/pub/apache/lucene/solr/4.10.2/solr-4.10.2.tgz
$ tar xzf solr-4.10.2.tgz
$ mv solr-4.10.2 solr && mv solr /opt
```

Uma vez que se pretendia a execução e avaliação das soluções com o menor número de optimizações e configurações possíveis, foi utilizado a configuração mais simples que já é distribuída com o *Solr*, com algumas alterações. Estas alterações foram ao nível do esquema dos documentos e uma configuração adicional para que o *Solr* fizesse a gestão automática dos *ids* dos novos documentos, caso contrário estes já teriam de ser passados juntamente com o documento.

Para alterar o esquema foi necessário editar o ficheiro `schema.xml` e adicionar os campos, assinalar o campo a utilizar como chave única e ainda configurar os tipos de dados possíveis para os campos. O esquema utilizado para a avaliação é o seguinte:

```
<schema name="example" version="1.5">
  <fields>
    <field name="_version_" type="long" indexed="true"
           stored="true" required="true"/>
    <field name="id" type="string" indexed="true"
           stored="true" required="true" multiValued="false" />
    <field name="timestamp" type="date" indexed="true"
           stored="true" required="true" multiValued="false" default="NOW"/>
    <field name="event_type" type="string" indexed="true"
           stored="true" required="true" multiValued="false" />
    <field name="src_ip" type="string" indexed="true"
           stored="true" required="true" multiValued="false" />
    <field name="src_port" type="int" indexed="true"
           stored="true" required="true" multiValued="false" />
    <field name="dest_ip" type="string" indexed="true"
           stored="true" required="true" multiValued="false" />
    <field name="dest_port" type="int" indexed="true"
           stored="true" required="true" multiValued="false" />
    <field name="proto" type="string" indexed="true"
           stored="true" required="true" multiValued="false" />
    <field name="alert_action" type="string" indexed="true"
          stored="true" required="true"  multiValued="false" />
    <field name="alert_gid" type="int" indexed="true"
          stored="true" required="true" multiValued="false" />
    <field name="alert_signature_id" type="int" indexed="true"
          stored="true" required="true" multiValued="false" />
    <field name="alert_rev" type="int" indexed="true"
```

```
        stored="true" required="true" multiValued="false" />
    <field name="alert_signature" type="string" indexed="true"
        stored="true" required="true" multiValued="false" />
    <field name="alert_category" type="string" indexed="true"
        stored="true" required="true" multiValued="false" />
    <field name="alert_severity" type="int" indexed="true"
        stored="true" required="true" multiValued="false" />
  </fields>
  <uniqueKey>id</uniqueKey>
  <types>
    <fieldType name="string" class="solr.StrField" sortMissingLast="true" />
    <fieldType name="date" class="solr.TrieDateField" precisionStep="0"
        positionIncrementGap="0"/>
    <fieldType name="int" class="solr.TrieIntField" precisionStep="0"
        positionIncrementGap="0"/>
    <fieldType name="long" class="solr.TrieLongField" precisionStep="0"
        positionIncrementGap="0"/>
    <fieldType name="text" class="solr.TextField" positionIncrementGap="100" />
  </types>
</schema>
```

Ao contrário do ES, foi necessário algumas configurações adicionais para que o *Solr* gerisse os *ids* dos documentos automaticamente. Para tal, foi editado o ficheiro `solrconfig.xml`, de forma a incluir a seguinte configuração para o campo `id`:

```
<updateRequestProcessorChain>
  <processor class="solr.UUIDUpdateProcessorFactory">
    <str name="fieldName">id</str>
  </processor>
  <processor class="solr.LogUpdateProcessorFactory" />
  <processor class="solr.RunUpdateProcessorFactory" />
</updateRequestProcessorChain>
```

Após as configurações, foi possível executar o *Solr* através do seguinte comando:

```
$ java -jar start.jar -Dsolr.solr.home=solr
```

Para verificar que o *Solr* estava a correr correctamente, recorreu-se a *dashboard* que é disponibilizada através do *url* `http://<serverIP>:8983/solr/`. Esta é uma funcionalidade que o ES não possuí nativamente, mas sim recorrendo a *plugins* de terceiros.

Vegeta HTTP load test tool

A instalação do *Vegeta* foi feita de forma análoga a instalação feita para os testes com ES. Essa configuração pode ser vista no Apêndice B.1.1

B.2.2 *Configuração e execução dos testes*

A configuração e execução dos testes foi feita de forma análoga a que é demonstrada no Apêndice B.1.2, com duas excepções a assinalar. Os ficheiros criados para serem utilizados pelo *Vegeta* durante os testes tiveram de ser alterados para responder aos requisitos do *Apache Solr*.

Uma vez que o *Apache Solr* não aceita aninhamento, foi necessário alterar o conteúdo do ficheiro `ids_body.json` para:

```
{
  "timestamp": "2014-04-15T23:32:11.736275-0600",
  "event_type": "alert",
  "src_ip": "10.16.1.11",
  "src_port": 49719,
  "dest_ip": "192.168.88.3",
  "dest_port": 443,
  "proto": "TCP",
  "alert_action": "allowed",
  "alert_gid": 1,
  "alert_signature_id": 30524,
  "alert_rev": 1,
  "alert_signature": "SERVER-OTHER OpenSSL TLSv1.1 heartbeat read overrun attempt",
  "alert_category": "Attempted Information Leak",
  "alert_severity": 2
}
```

A segunda alteração, teve haver com o *endpoint* e o tipo de conteúdo a utilizar na geração dos pedidos. Logo, o ficheiro `solr_target.txt`, neste caso, teve de ser alterado para:

```
POST http://<server IP>:8983/solr/update/json?commit=true
Content-type:application/json
@/path/to/ids_body.json
```

B.3 APACHE HADOOP HDFS

B.3.1 *Instalação e Configuração*

Ambiente de Testes

Os testes foram realizados sobre uma infraestrutura virtual, composta por 5 máquinas, com as seguintes características:

Duas máquinas:

- 1 vCPU
- 2GB RAM
- 5GB HDD
- Fedora 19 x86_64

Três máquinas:

- 1 vCPU
- 2GB RAM
- 20GB HDD
- Fedora 19 x86_64

Preparação inicial

A preparação inicial das máquinas foi feita de forma análoga a preparação feita para o ES. Essa preparação pode ser vista no apêndice B.1.1.

Além das configuraçõess vistas no apêndice referido anteriormente, foi ainda necessário verificar que todas as máquinas possuiam um cliente e servidor de SSH, para isso executou-se o seguinte comando:

```
$ yum install -y openssh-clients openssh-server
```

Por motivos de segurança foi ainda necessário, a criação de um novo utilizador no sistema para uso exclusivo do HDFS.

```
$ useradd hadoop
$ passwd hadoop
```

Para tornar mais fácil conhecer as máquinas, foi adicionado ao ficheiro `/etc/hosts`, de todas as máquinas, as seguintes linhas:

```
10.10.6.3 HADOOP-NN1
10.10.6.4 HADOOP-NN2
10.10.6.5 HADOOP-DN1
10.10.6.6 HADOOP-DN2
10.10.6.7 HADOOP-DN3
```

Para activar o mecanismo de alta disponibilidade do HDFS foi necessário configurar também o acesso remoto, por chave, ao SSH. Para tal, foi gerado um conjunto chaves privada/pública e a chave pública foi copiada para as restante máquinas do *cluster*. Para tal, foi executado o seguinte em cada uma das máquinas dos *cluster*:

```
$ su - hadoop
$ ssh-keygen -t rsa
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop@HADOOP-NN1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop@HADOOP-NN2
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop@HADOOP-DN1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop@HADOOP-DN2
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop@HADOOP-DN3
$ chmod 0600 ~/.ssh/authorized_keys
$ exit
```

Oracle Java 7u71 JDK

A configuração do *Java* foi feita de forma análoga a configuração feita para o ES. Essa configuração pode ser vista no apêndice B.1.1. É necessário que se verifique se as variáveis de ambiente `JAVA_HOME` e `JRE_HOME` existem e estão correctamente definidas.

Apache Zookeeper

Uma vez que esta instalação foi apenas feita com o intuito de analisar o desempenho e tolerância a faltas, configurou-se apenas um servidor *ZooKeeper*, numa das máquinas que será usada para guardar os dados do HDFS - `HADOOP-DN1` - para assim testar a alta disponibilidade dos `Namenode`. Para este caso, a instalação foi relativamente simples, como demonstram os seguintes passos:

```
$ wget http://mirrors.fe.up.pt/pub/apache/zookeeper/stable/zookeeper-3.4.5.tar.gz
$ tar xzf zookeeper-3.4.5.tar.gz && mv zookeeper-3.4.5 /opt/zookeeper
```

```
$ cp -a /opt/zookeeper/conf/zoo_sample.cfg /opt/zookeeper/conf/zoo.cfg
$ mkdir /opt/zookeeper/data
$ vim /opt/zookeeper/conf/zoo.cfg
    dataDir=/opt/zookeeper/data
```

Apache Hadoop HDFS

Para configurar o `Hadoop`, bastou configurar o mesmo numa da máquinas, um `namenode`, e depois copiar a directória e configurações para as restantes máquinas do *cluster*. Como tal, as configurações que se seguem, foram executadas todas na máquina `HADOOP-NN1`.

Para a instalação inicial do *Hadoop*, foi executado o seguinte conjunto de comandos:

```
$ mkdir /opt/hadoop
$ cd /opt/hadoop/
$ wget http://mirrors.fe.up.pt/pub/apache/hadoop/common/stable/hadoop-2.2.0.tar.gz
$ tar -xzf hadoop-2.2.0.tar.gz && mv hadoop-2.2.0 hadoop
$ chown -R hadoop /opt/hadoop
$ cat >> ~./bashrc << _EOF_
\ export HADOOP_HOME=/opt/hadoop
\ export HADOOP_MAPRED_HOME=$HADOOP_HOME
\ export HADOOP_COMMON_HOME=$HADOOP_HOME
\ export HADOOP_HDFS_HOME=$HADOOP_HOME
\ export YARN_HOME=$HADOOP_HOME
\ export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
\ export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
\ _EOF_
$ source ~./bashrc
$ mkdir -p $HADOOP\_HOME/tmp
```

A partir daqui foi feita a configuração do *Hadoop*, recorrendo a um conjunto de ficheiros `.xml` presentes na directoria de configuração, `/opt/hadoop/etc/hadoop`. Foram feitas as seguintes alterações, nos ficheiros seguintes.

core-site.xml

```
<property>
  <name>fs.default.name</name>
```

```
  <value>hdfs://bench-hadoop</value>
</property>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/opt/hadoop/tmp</value>
</property>
```

hdfs-site.xml

```
<property>
  <name>dfs.nameservices</name>
  <value>bench-hadoop</value>
</property>
<property>
  <name>dfs.ha.namenodes.bench-hadoop</name>
  <value>HADOOP-NN1,HADOOP-NN2</value>
</property>
<property>
  <name>dfs.namenode.rpc-address.bench-hadoop.HADOOP-NN1</name>
  <value>HADOOP-NN1:8020</value>
</property>
<property>
  <name>dfs.namenode.rpc-address.bench-hadoop.HADOOP-NN2</name>
  <value>HADOOP-NN2:8020</value>
</property>
<property>
  <name>dfs.namenode.http-address.bench-hadoop.HADOOP-NN1</name>
  <value>HADOOP-NN1:50070</value>
</property>
<property>
  <name>dfs.namenode.http-address.bench-hadoop.HADOOP-NN2</name>
  <value>HADOOP-NN2:50070</value>
</property>
<property>
  <name>dfs.client.failover.proxy.provider.bench-hadoop</name>
  <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
</property>
<property>
  <name>dfs.ha.fencing.methods</name>
  <value>sshfence</value>
</property>
<property>
  <name>dfs.ha.fencing.ssh.private-key-files</name>
  <value>~/.ssh/id_rsa.pub</value>
</property>
<property>
  <name>dfs.replication</name>
  <value>2</value>
```

```
</property>
<property>
  <name>dfs.permissions</name>
  <value>false</value>
</property>
<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>

<!-- Configurar o failover automático e
     dar a conhecer a localização do servidor Zookeeper -->
<property>
  <name>dfs.ha.automatic-failover.enabled</name>
  <value>true</value>
</property>
<property>
  <name>ha.zookeeper.quorum</name>
  <value>HADOOP-DN1:2181</value>
</property>

<!-- Configuração do Quorum Journal Manager -->
<!-- Propriedade a configurar nos Namenodes -->
<property>
  <name>dfs.namenode.shared.edits.dir</name>
  <value>qjournal://HADOOP-DN1:8485;HADOOP-DN2:8485;HADOOP-DN3:8485/bench-hadoop</value>
</property>
<!-- Se esta máquina for um JournalNode,
     descomentar esta propriedade e comentar a propriedade anterior
<property>
  <name>dfs.namenode.shared.edits.dir</name>
  <value>/opt/journalnode/data</value>
</property>
-->
```

mapred-site.xml

```
<!-- Define a framework de gestão de recursos para YARN -->
<property>
  <name>mapred.framework.name</name>
  <value>yarn</value>
</property>
```

Uma vez que só interessava a configuração do HDFS, o *YARN* foi configurado com o mínimo e sem alta disponibilidade, como demonstra a seguinte configuração.

yarn-site.xml

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
  <name>yarn.resourcemanager.resource-tracker.address</name>
  <value>hadoop-NN1:8025</value>
</property>
<property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>hadoop-NN1:8030</value>
</property>
<property>
  <name>yarn.resourcemanager.address</name>
  <value>hadoop-NN1:8040</value>
</property>
```

Após a configuração anterior, bastou então copiar a directoria `/opt/hadoop` para as restantes máquinas, assim como a realização dos `exports` necessários, das váriaveis de ambiente.

```
$ su − hadoop
$ scp −r /opt/hadoop HADOOP−NN2:/opt/hadoop
$ scp −r /opt/hadoop HADOOP−DN1:/opt/hadoop
$ scp −r /opt/hadoop HADOOP−DN2:/opt/hadoop
$ scp −r /opt/hadoop HADOOP−DN3:/opt/hadoop
```

Após a cópia das configurações, foi ainda necessário alguns acertos nos ficheiros de configuração, nomeadamente para os `journalnodes` - `HADOOP-DN1`, `HADOOP-DN2` e `HADOOP-DN3`. Estas alterações estão em comentário no ficheiro `hdfs-site.xml`, acima listado.

Nas máquinas `HADOOP-NN1` e `HADOOP-NN2`, foi ainda necessário preencher o ficheiro `slaves` com o seguinte conteúdo:

```
HADOOP-DN1
HADOOP-DN2
HADOOP-DN3
```

Antes de arrancar o sistema, foi necessário inicializar o HDFS, para isso, foi executado o seguinte comando no `namednode` `HADOOP-NN1`:

```
$ su − hadoop
$ /opt/hadoop/bin/hdfs namenode −format
```

Depois de inicializado o HDFS, seguiu-se a execução dos seguintes comandos, pela ordem apresentada:

`[HADDOP-DN1, HADOOP-DN2, HADOOP-DN3]`

```
$ su − hadoop
$ /opt/hadoop/sbin/hadoop−daemon.sh start journalnode
```

`[HADDOP-DN1]`

```
$ su − hadoop
$ /opt/zookeeper/bin/zkServer.sh start
```

`[HADOOP-NN1, HADOOP-NN2]`

```
$ su − hadoop
$ /opt/hadoop/sbin/hadoop−daemon.sh start zkfc
```

`[HADOOP-NN1]`

```
$ su − hadoop
$ /opt/hadoop/sbin/hadoop−daemon.sh start namenode
$ /opt/hadoop/sbin/hadoop−daemons.sh start datanode
$ /opt/hadoop/sbin/yarn−daemon.sh start resourcemanager
$ /opt/hadoop/sbin/yarn−daemons.sh start nodemanager
$ /opt/hadoop/sbin/mr−jobhistory−daemon.sh start historyserver
```

Para verificar que serviços estão a correr, usar o utilitário da JVM, `jps`[4].

4 *Java Virtual Machine Process Status Tool*

B.3.2 *Configuração e execução dos testes*

Para avaliação desta solução, tirou-se partido de algum material utilizado durante a investigação e desenvolvimento do artigo *Evaluating Cassandra as a Manager of Large File Sets*[5]. Esse material foi disponibilizado pelos seus autores e será divulgado no decorrer desta Secção.

Popular o HDFS

Antes de popular o HDFS, foi necessário gerar os dados. Para esse efeito, foi utilizado um conjunto de ficheiros que possuíam o *ouput* resultante da execução do comando `ls -laR` numa directoria que teria sido gerada para o efeito. A partir dessa listagem, foi possível gerar a estrutura, com recurso a um *script Python* e a dois comandos *UNIX*, *mkdir*[6] e *dd*[7]. Após a geração dos ficheiros, foi então populado o HDFS. Os *scripts* de geração e população, apresentados a seguir, foram adaptações aos *scripts* utilizados durante o desenvolvimento do artigo acima referido.

***Script* para geração dos dados**:

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-

''' Script to generate dummy files '''
''' IMPORTs '''
import os,sys
import mmap
from subprocess import call,Popen,PIPE
from multiprocessing import Process
from datetime import datetime
import time

''' GLOBAL VARs '''
local_folder = '/path/to/RANDOM_DATA'
''' DEFs '''
## responsible for reading 'files' list and start a new process for each one
def read_filesList(filesNames):
  process_id = 0
  process_list = []
  i = 0
  for fileName in filesNames:
    if os.path.isdir(fileName) == True:
      print "[INFO] Finded DIR (not processed):", fileName
    else:
      print "[INFO] File to process:", fileName
```

5 *Evaluating Cassandra as a Manager of Large File Sets* - http://repositorium.sdum.uminho.pt/bitstream/1822/34119/1/912.pdf

6 Utilitário *UNIX* para a criação de directorias.

7 Utilitário *UNIX* que permite a cópia de dados entre um *input* e um *output*

```
      p = Process(target=process_file, args=(process_id, fileName,))
      process_id += 1
      p.start()
      i+=1
      process_list.append(p)
  return process_list

# optimized for processing files with origin in ls -laR command
def process_file(process_id, fileName):
  f = os.open(fileName, os.O_RDONLY)
  fmap = mmap.mmap(f, 0, prot=mmap.PROT_READ)
  readline = fmap.readline

  folder_name = readline()
  folder_name = (str.replace(folder_name, ":", "")).strip(' \t\n\-r\.\/')

  if(folder_name==""):
    folder_name="/"

  readline() # total
  readline() # blank line

  line = readline()
  while line:
    striped_line = line.strip(' \t\n\r')

    if striped_line == "":
      folder_name = readline()
      folder_name = (str.replace(folder_name, ":", "")).strip(' \t\n\.')

      readline() # blank line
      readline() # total
    else:
      line_data = line.split()
      line_size = len(line_data)

      if striped_line.startswith("-"):
        file_name = line_data[8]
        file_size = line_data[4]

        full_local_path = str(local_folder + folder_name + "/" + file_name)
        if(os.path.exists(str(local_folder + folder_name))):
          call(['dd', 'if=/dev/zero',
                      str('of='+full_local_path),
                      str('bs='+file_size),
                      'count=1',
                      'status=none'])
        else:
          call(['mkdir','-p',str(local_folder + folder_name)])
          call(['dd', 'if=/dev/zero',
                      str('of='+full_local_path),
                      str('bs='+file_size),
                      'count=1',
                      'status=none'])
```

```
      if striped_line.startswith("d"):
        dir_name = line_data[8]
        full_local_path = str(local_folder+folder_name+"/"+dir_name)
        call(['mkdir','-p',full_local_path])

    line = readline()

''' INVOCATION '''
if len(sys.argv) == 1 or len(sys.argv) > 2:
  print "ERROR - Usage: python random_files.py <path_to_ls-laR_output_files>" +
        " | Example: python random_files.py outs/"
  sys.exit(-1)

if len(sys.argv) == 2:
  input_file = sys.argv[1]

print str("[START] Start generating files!")

filesList = []

base_name = os.path.basename(input_file)
orig_folder = os.path.dirname(input_file)
dir_list = os.listdir(orig_folder)

for entry in dir_list:
  if entry != base_name and entry.startswith(base_name):
    filesList.append(orig_folder + "/" + entry)

if len(filesList) == 0:
  filesList.append(input_file)

start_time = datetime.now()

minutes = 0;
process_list = read_filesList(filesList)
test = True
ended_processes = 0
while test:
  time.sleep(600)
  minutes += 10
  for i in range(0,len(process_list)):
    if not(process_list[i].is_alive()):
      print str("[INFO] " + process_list[i].name + " terminates! ")
      ended_processes += 1
    else:
      print str("[INFO] " + process_list[i].name + " still running! ")

  print str("[INFO] Time: " + str(minutes) + " minutes passed!")

  if ended_processes == 4:
    test = False

end_time = datetime.now()
total_time = end_time - start_time
print str("[END] Stopped generating files! Total time: " +
```

```
          str(total_time.seconds) +
          " seconds!")
```

Script para popular o *HDFS*:

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-

''' Script to store dummy files on HadoopHDFS '''

''' IMPORTs '''
import os,sys
import mmap
from multiprocessing import Process
from datetime import datetime
from pywebhdfs.webhdfs import PyWebHdfsClient

''' GLOBAL VARs '''
local_folder = '/path/to/RANDOM_DATA'
hdfs_folder = 'benchs/'
keys_fd = os.open('/paht/to/hdfs.keys', os.O_CREAT|os.O_WRONLY)

''' DEFs '''
## responsible for reading 'files' list and start a new process for each one
def read_filesList(filesNames, hdfs):
  process_id = 0
  i = 0
  for fileName in filesNames:
    if os.path.isdir(fileName) == True:
      print "[INFO] Finded DIR (not processed):", fileName
    else:
      print "[INFO] File to process:", fileName
      p = Process(target=process_file, args=(process_id, fileName, hdfs,))
      process_id += 1
      p.start()
      p.join()
      i+=1

  os.close(keys_fd)
  print "[INFO] keys_file created..."

# optimized for processing files with origin in ls -laR command
def process_file(process_id, fileName, hdfs):
  f = os.open(fileName, os.O_RDONLY)
  fmap = mmap.mmap(f, 0, prot=mmap.PROT_READ)
  readline = fmap.readline

  folder_name = readline()
  folder_name = (str.replace(folder_name, ":", "")).strip(' \t\n\-r\.\/')

  if(folder_name==""):
    folder_name="/"

  readline() # total
```

```
  readline() # . path
  readline() # .. path

  line = readline()
  while line:
    striped_line = line.strip(' \t\n\r')
    if striped_line == "":
      folder_name = readline()
      folder_name = (str.replace(folder_name, ":", "")).strip(' \t\n\.')
      readline() # total
      readline() # . path
      readline() # .. path
    else:
      line_data = line.split()
      line_size = len(line_data)
      if striped_line.startswith("-"):
        file_name = line_data[8]
        full_local_path = str(local_folder + folder_name + "/" + file_name)
        if os.path.exists(full_local_path):
          local_file = os.open(full_local_path, os.O_RDONLY)
          file_data = os.read(local_file, int(line_data[4]))
          hdfs.create_file(str(hdfs_folder + folder_name + /" + file_name),
                           file_data, overwrite=True)
          os.write(keys_fd, str(folder_name + "/" + file_name + "\n"))
          os.close(local_file)
      if striped_line.startswith("d"):
        dir_name = line_data[8]
        hdfs.make_dir(str(hdfs_folder+folder_name+"/"+dir_name))
    line = readline()

''' INVOCATION '''
if len(sys.argv) == 1 or len(sys.argv) > 2:
        print "ERROR - Usage: python create_files_HDFS.py" +
        " <path_to_ls-laR_output_files> | " +
        " Example: python create_files_HDFS.py outs/"
        sys.exit(-1)

if len(sys.argv) == 2:
        input_file = sys.argv[1]

# Client WebHDFS
hdfs = PyWebHdfsClient(host='hadoop-NN2',port='50070', user_name='hadoop')

# file list to be stored on HDFS
filesList = []

base_name = os.path.basename(input_file)
orig_folder = os.path.dirname(input_file)
dir_list = os.listdir(orig_folder)

for entry in dir_list:
  if entry != base_name and entry.startswith(base_name):
    filesList.append(orig_folder + "/" + entry)

if len(filesList) == 0:
```

```
  filesList.append(input_file)

start_time = datetime.now()
read_filesList(filesList, hdfs)
end_time = datetime.now()

total_time = end_time - start_time
print str("[INFO] Total time: " + str(total_time.seconds) + " seconds!")
```

Configuração do YCSB

Uma vez que o YCSB ainda não disponha de nenhum *binding* para *WebHDFS*, foi necessário desenvolver o mesmo, como foi referido no Capítulo 4.2.1. Uma vez que o *binding* apresenta alguma extensão, optou-se por não ser incluído neste documento, mas encontra-se disponível, em *opensource*, num repositório criado para o efeito[8]. Depois do *binding* desenvolvido, foi necessário editar o ficheiro de configuração do YCSB para este reconhecer o novo módulo. Para isso editou-se o ficheiro *pom.xml*, que se encontra na raiz do projecto YCSB, e foi adicionada a seguinte linha:

```
<modules>
  <module>hdfs</module>
</modules>
```

Foi então necessário criar um ficheiro de *workload* para configurar as características dos testes. O ficheiro foi criado, debaixo da directoria *workiload*, com a seguinte configuração:

```
recordcount=1131396
operationcount=100000
workload=com.yahoo.ycsb.workloads.File_CoreWorkload
readallfields=true

readproportion=1
updateproportion=0
scanproportion=0
insertproportion=0

requestdistribution=zipfian
maxscanlength=100
scanlengthdistribution=uniform
```

Para automatizar a execução dos testes, criou-se então um *script*, em Bourne Again Shell (Bash), com o seguinte conteúdo:

8 *YCSB WebHDFS Binding* - https://github.com/BrunoAndrade/ycsb-webhdfs-binding

```
#!/bin/bash

export YCSB_HEAP_SIZE=2048
WORKLOAD_HDFS=${WORKLOAD:-"workloads/workload_hdfs"}

CLIENTS="1 5 10 15 20 25"

for CLIENT in $CLIENTS
do
  bin/ycsb run hdfs -P ${WORKLOAD_HDFS} \
                    -p filesystem.base_folder="benchs" \
                    -p keys_file="/path/to/hdfs.keys" \
                    -p use_file_columns=false \
                    -threads $CLIENT > results_hdfs/Hdfs_benchFS-$CLIENT.out
done
```