



**UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA DE COMPUTAÇÃO**

PABLO FAVERO DE SIQUEIRA

**SISTEMA DE MONITORAMENTO DE ARQUIVOS DE LOG
DE APLICAÇÃO BASEADO NA PLATAFORMA ELASTIC
STACK**

**NATAL-RN
2020**

Pablo Favero de Siqueira

Sistema de monitoramento de arquivos de log de aplicação baseado na plataforma Elastic Stack

Trabalho de Conclusão de Curso na modalidade Monografia, submetido como parte dos requisitos necessários para conclusão do curso de Engenharia de Computação pela Universidade Federal do Rio Grande do Norte

Orientador: Prof. Dr. Carlos Manuel Dias Viegas

Natal-RN
2020

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Siqueira, Pablo Favero de.

Sistema de monitoramento de arquivos de log de aplicação baseado na plataforma Elastic Stack / Pablo Favero de Siqueira. - 2020.

0f76.: il.

Monografia (Graduação) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Curso de Engenharia de Computação, Natal, 2020.

Orientador: Dr. Carlos Manuel Dias Viegas.

1. Elastic Stack - Monografia. 2. Monitoramento de Arquivos de Log - Monografia. 3. Elasticsearch - Monografia. 4. Logstash - Monografia. 5. Kibana - Monografia. I. Viegas, Carlos Manuel Dias. II. Título.

RN/UF/BCZM

CDU 004

Pablo Favero de Siqueira

Sistema de monitoramento de arquivos de log de aplicação baseado na plataforma Elastic Stack

Trabalho de Conclusão de Curso na modalidade Monografia, submetido como parte dos requisitos necessários para conclusão do curso de Engenharia de Computação pela Universidade Federal do Rio Grande do Norte

Aprovado em 03 de dezembro de 2020:

Prof. Dr. Carlos Manuel Dias Viegas – Orientador
UFRN

Prof. Dr. Danilo Curvelo de Souza – Examinador interno
UFRN

M. Sc. Aguinaldo Bezerra Batista Jr – Examinador externo
Centro de Lançamento da Barreira do Inferno (CLBI)

Natal-RN
2020

AGRADECIMENTOS

Aos meus pais e a minha irmã por todo o apoio dentro de casa durante todo o período da minha graduação.

A todos os meus colegas e professores da UFRN que me ajudaram de alguma forma ao longo da graduação.

Ao meu orientador, Prof. Dr. Carlos Manuel Dias Viegas pelo direcionamento e auxílio concedido na elaboração deste trabalho.

E também para toda a equipe que conviveu comigo durante o meu período estagiando no MPRN, me auxiliando em várias tarefas diárias e compartilhando diversos conhecimentos, em especial para o coordenador geral do estágio, Rivaldo Xavier da Silva Júnior, e também para o coordenador de infraestrutura e redes, Josemberg Pessoa Borges, que foi quem sugeriu a criação desse projeto e forneceu os dados que foram utilizados neste trabalho.

RESUMO

Devido à constante popularização do acesso à Internet e a sistemas de informática, muitas empresas e instituições vêm se aproveitando desse contexto para implementar a digitalização de seus serviços em busca de uma maior agilidade e acessibilidade em suas atividades através de diversas soluções computacionais. Para manter esses novos sistemas em plenas condições de funcionamento é essencial que haja um monitoramento do funcionamento das aplicações envolvidas. Essa tarefa pode ser realizada através da leitura de arquivos de *log* que são arquivos de texto gerados automaticamente pelas aplicações contendo informações de seu funcionamento como usuários que acessam o sistema, alertas de erros de execução, etc. A leitura desses arquivos pode ser difícil devido, principalmente, a fatores como a grande quantidade de texto gerado e sua linguagem que pode não ser de fácil compreensão para um humano. Para facilitar essa tarefa, este trabalho propõe a criação de um sistema de monitoramento que irá ler esses arquivos, tratar seus dados e exibi-los através de painéis de visualização (*dashboards*) que são páginas com gráficos, tabelas e elementos visuais interativos que possibilitam ao usuário uma leitura mais fácil e rápida das informações de *log* em tempo real. Esse sistema é feito com as aplicações do grupo *Elastic Stack* que são: *Beats* e *Logstash* (leitura e tratamento de dados), *Elasticsearch* (armazenamento de dados) e *Kibana* (visualização de dados). Todas as ferramentas utilizadas nesse trabalho são gratuitas e de código aberto (*open source*) além de possibilitarem uma fácil adaptação para vários tipos diferentes de sistemas operacionais.

Palavras-chave: Monitoramento, Arquivos de Log, Elasticsearch, Logstash, Kibana, Dashboards.

ABSTRACT

Due to the constant popularization of the access to the Internet and computer systems, many companies and institutions are taking advantage of that context to implement the digitalization of their services to achieve better efficiency and accessibility in their activities through many computer solutions. To keep these systems working properly it is essential to monitor the working of the involved applications. That task can be made by reading log files, they are text files automatically generated by the applications that contain information about their functioning like the users' access to the system and execution errors warnings. Reading those files can be hard because of some factors like the big quantity of generated text and its language that can be difficult to comprehend by a human. To make this task easier, this work proposes the creation of a monitoring system that will read those files, organize their data and show them using dashboards that are pages with graphics, tables and other interactive visual elements which allow the user an easier and faster reading of the log information in real-time. This system is made with the applications from the Elastic Stack group, they are: Beats and Logstash (reading and manipulation of data), Elasticsearch (data storage) and Kibana (data visualization). All the tools used in this project are freeware and open source and they also allow an easy portability between different operational systems.

Keywords: Monitoring, Log Files, Elasticsearch, Logstash, Kibana, Dashboards.

LISTA DE ILUSTRAÇÕES

Figura 1 - Fluxograma de funcionamento do sistema de monitoramento.....	17
Figura 2 - Documento armazenado no <i>Elasticsearch</i>	20
Figura 3 - Busca por <i>index</i> invertido no <i>Elasticsearch</i>	20
Figura 4 - Hierarquia dos componentes do servidor <i>Apache Tomcat</i>	23
Figura 5 - Comandos para iniciar o <i>Elasticsearch</i>	27
Figura 6 - Mensagem de sucesso do <i>Elasticsearch</i>	27
Figura 7 - Comandos para iniciar o <i>Kibana</i>	28
Figura 8 - Página inicial do <i>Kibana</i>	28
Figura 9 - Propriedades do <i>Tomcat</i> no <i>Windows</i>	29
Figura 10 - Página inicial do servidor <i>Apache Tomcat</i>	29
Figura 11 - Documento criando usuário do <i>Tomcat</i>	30
Figura 12 - Mensagem de sucesso do <i>Jolokia</i>	30
Figura 13 - Código da função <i>input</i> recebendo <i>Acessos.csv</i>	31
Figura 14 - Código nomeando campos do acesso	32
Figura 15 - Código convertendo data e hora	32
Figura 16 - Código manipulando o <i>IP</i>	32
Figura 17 - Trecho do arquivo de dicionário de comarcas	33
Figura 18 - Código da função <i>translate</i> para criar a comarca	33
Figura 19 - Trecho do arquivo de dicionário de municípios.....	33
Figura 20 - Código da função <i>translate</i> para criar o município.....	33
Figura 21 - Trecho do arquivo de dicionário de coordenadas	33
Figura 22 - Código da função <i>translate</i> para criar a coordenada	33
Figura 23 - Código criando o campo <i>location</i> no padrão do <i>Kibana</i>	34
Figura 24 - Código removendo campos que não serão mais utilizados	34
Figura 25 - Código para a saída dos dados de acessos.	34
Figura 26 - Código da função <i>input</i> recebendo <i>catalogina_prod.out</i>	36
Figura 27 - Código da função <i>dissect</i>	36
Figura 28 - Código da função <i>date</i>	36
Figura 29 - Trecho do arquivo de dicionário de categorias de alertas.....	37
Figura 30 - Código classificando alertas de tipo " <i>INFO</i> "	37
Figura 31 - Código classificando alertas de tipo " <i>SEVERE</i> "	37
Figura 32 - Código classificando erros de usuários	38
Figura 33 - Código classificando alertas de tipo " <i>WARNING</i> " e retirando campos ..	38
Figura 34 - Código de saída dos dados de alertas.....	39
Figura 35 - Código da função <i>input</i> recebendo <i>req_log.txt</i>	39
Figura 36 - Código de filtro <i>Grok</i>	40
Figura 37 - Código de função <i>date</i>	40
Figura 38 - Código removendo e renomeando campos	40
Figura 39 - Código criando campos de comarca e município	40
Figura 40 - Código removendo e convertendo campos.....	41
Figura 41 - Código de saída dos dados de requisições.	41
Figura 42 - Código de função <i>input</i> recebendo dados da porta 5044	42
Figura 43 - Código manipulando os dados de métricas	42
Figura 44 - Código enviando dados de métricas	43

Figura 45 - Módulo <i>Jolokia</i> do <i>Metricbeat</i>	43
Figura 46 - Executando o <i>Metricbeat</i> com <i>Jolokia</i> habilitado	44
Figura 47 - <i>Template</i> padrão para o <i>index</i> de acessos	44
Figura 48 - Exemplo de execução do <i>Logstash</i>	45
Figura 49 - <i>Index pattern</i> para o painel de acessos	46
Figura 50 - Documentos de "acessos-v1" descobertos.....	46
Figura 51 - Visualização do título da aplicação	47
Figura 52 - Visualização do logo da instituição	47
Figura 53 - Visualização da métrica de total de acessos	48
Figura 54 - Visualização da tabela de acessos por nome e matrícula	48
Figura 55 - Visualização da tabela de acessos por município.....	48
Figura 56 - Visualização do gráfico de <i>top 10</i> comarcas	49
Figura 57 - Visualização do mapa de acessos no RN.....	49
Figura 58 - Visualização do título "Alertas do Servidor"	50
Figura 59 - Visualização da métrica de total de alertas.....	50
Figura 60 - Visualização da tabela de total de alertas por categoria.....	50
Figura 61 - Visualização da tabela de total de mensagens	51
Figura 62 - Visualização do gráfico de tipos de alertas	51
Figura 63 - Visualização do título "Erros de Acesso de Usuários"	51
Figura 64 - Visualização da métrica de total de erros de acesso de usuários.....	52
Figura 65 - Visualização da tabela de usuários com erros.....	52
Figura 66 - Visualização do gráfico <i>top 10</i> de usuários com erros.....	52
Figura 67 - Visualização do gráfico de categorias de erros de acesso de usuários.	53
Figura 68 - Visualização do título "Requisições"	53
Figura 69 - Configuração convertendo de <i>bytes</i> para <i>gigabytes</i>	54
Figura 70 - Visualização da métrica de soma de <i>gigabytes</i> enviados	54
Figura 71 - Visualização da métrica de total de requisições	54
Figura 72 - Visualização da tabela de requisições por <i>IP</i>	54
Figura 73 - Visualização da tabela de requisições por objeto	55
Figura 74 - Visualização do gráfico de tipos de requisição	55
Figura 75 - Visualização do gráfico de comarcas e municípios.....	56
Figura 76 - Visualização do título "Métricas <i>JMX</i> "	56
Figura 77 - Visualização do logo do <i>Apache Tomcat</i>	56
Figura 78 - Configuração convertendo de ms para h	57
Figura 79 - Visualização da métrica de <i>uptime</i> em horas.....	57
Figura 80 - Visualização da métrica do total de eventos	57
Figura 81 - Visualização da métrica de nome e versão do servidor	57
Figura 82 - Visualização do gráfico da memória <i>heap</i>	58
Figura 83 - Visualização do gráfico da memória <i>non heap</i>	58
Figura 84 - Visualização do gráfico de carga da <i>CPU (CPU Load)</i>	58
Figura 85 - Painel de acessos.....	59
Figura 86 - Painel de alertas	60
Figura 87 - Painel de erros de acesso de usuários	60
Figura 88 - Painel de requisições	61
Figura 89 - Painel de métricas <i>JMX</i>	61
Figura 90 - Pesquisa pelo console	62
Figura 91 - Estatísticas da busca feita no console	63
Figura 92 - Exemplo de documento encontrado na busca do console.....	63
Figura 93 - Painel de acessos de março de 2019	64

Figura 94 - Painel com acessos que não são de Natal ou Parnamirim	65
Figura 95 - Seleção de nome de usuário	65
Figura 96 - Painel de acessos feitos apenas pelo usuário selecionado	66
Figura 97 - Consulta por município e nome	67
Figura 98 - Painel de acessos de Mossoró com usuários de sobrenome Silva	67
Figura 99 - Opções de visualização	68
Figura 100 - Inspeccionando a visualização	68
Figura 101 - Trecho da tabela exportada em formato .csv.....	69
Figura 102 - Compartilhando busca	69
Figura 103 - Trecho do arquivo .csv de busca salvo	69
Figura 104 - Lista de objetos salvos.....	70

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i>
CPU	<i>Central Processing Unit</i>
CSV	<i>Comma Separated Value</i>
ELK	<i>Elasticsearch, Logstash e Kibana</i>
GED	Gerenciamento Eletrônico de Documentos
GUI	<i>Graphical User Interface</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IP	<i>Internet Protocol</i>
JMX	<i>Java Management Extensions</i>
JSON	<i>Javascript Object Notation</i>
JSP	<i>Java Server Pages</i>
JVM	<i>Java Virtual Machine</i>
KQL	<i>Kibana Query Language</i>
MBean	<i>Managed Java Object</i>
MPRN	Ministério Público do Rio Grande do Norte
NDJSON	<i>Newline Delimited JSON</i>
ONU	Organização das Nações Unidas
RAM	<i>Random-Access Memory</i>
REST	<i>Representational State Transfer</i>
RN	Rio Grande do Norte
SQL	<i>Structured Query Language</i>
URL	<i>Uniform Resource Locator</i>
WAR	<i>Web Application Resource</i>

SUMÁRIO

LISTA DE ILUSTRAÇÕES	7
LISTA DE ABREVIATURAS E SIGLAS.....	10
1 INTRODUÇÃO.....	13
1.1 MOTIVAÇÃO.....	14
1.2 OBJETIVOS.....	15
1.3 ORGANIZAÇÃO DO TRABALHO.....	16
2 APRESENTAÇÃO DAS FERRAMENTAS	17
2.1 ELASTIC STACK.....	18
2.1.1 Logstash.....	18
2.1.2 Beats.....	19
2.1.3 Elasticsearch.....	19
2.1.4 Kibana.....	21
2.2 FERRAMENTAS PARA A OBTENÇÃO DE MÉTRICAS.....	21
2.2.1 Servidor Apache Tomcat.....	22
2.2.2 Agente Jolokia.....	23
2.3 FERRAMENTAS AUXILIARES.....	24
2.4 VERSÕES UTILIZADAS.....	25
3 DESENVOLVIMENTO DO PROJETO.....	26
3.1 PREPARAÇÃO DO AMBIENTE.....	26
3.1.1 Iniciando o Servidor Elasticsearch.....	27
3.1.2 Iniciando o Kibana.....	28
3.1.3 Iniciando o Servidor Apache Tomcat.....	28
3.1.4 Iniciando o Agente Jolokia.....	30
3.2 DESCRIÇÃO DOS PAINÉIS E CONFIGURAÇÃO DO LOGSTASH.....	31
3.2.1 Painel de Acessos.....	31
3.2.2 Painéis de Alertas e Erros de Acesso de Usuários.....	35
3.2.3 Painel de Requisições.....	39
3.2.4 Painel de Métricas.....	41
3.3 CONFIGURAÇÃO E EXECUÇÃO DO METRICBEAT.....	43
3.4 PREPARAÇÃO DO KIBANA.....	44
3.5 EXECUÇÃO DO LOGSTASH.....	45
3.6 CRIAÇÃO DO INDEX PATTERN NO KIBANA.....	45
3.7 DESCOBERTA DE DADOS NO KIBANA.....	46
3.8 CRIAÇÃO DE VISUALIZAÇÕES NO KIBANA.....	47
3.8.1 Painel de Acessos.....	47
3.8.2 Painel de Alertas do Sistema.....	50
3.8.3 Painel de Erros de Acesso de Usuários.....	51
3.8.4 Painel de Requisições.....	53
3.8.5 Painel de Métricas.....	56
3.9 CRIAÇÃO DOS PAINÉIS DE VISUALIZAÇÃO NO KIBANA.....	59
4 UTILIZAÇÃO DOS PAINÉIS E DE FERRAMENTAS AUXILIARES	62
4.1 DESCOBERTA DE DADOS.....	62
4.2 CONSOLE.....	62
4.3 FILTROS.....	63
4.4 INTERAÇÃO COM AS VISUALIZAÇÕES.....	65
4.5 CONSULTAS.....	66

4.6 EXPORTAÇÃO E COMPARTILHAMENTO DE DADOS.....	68
5 CONSIDERAÇÕES FINAIS.....	71
REFERÊNCIAS	72

1 INTRODUÇÃO

A cada ano que se passa, o acesso à Internet e a sistemas de informática vem se popularizando no mundo todo. Com isso, muitas empresas e instituições vêm aderindo à digitalização de seus serviços em busca de uma melhor eficiência em seus negócios.

Esse cenário de maior uso de sistemas informatizados vem também crescendo no âmbito dos órgãos públicos brasileiros onde, segundo dados da Secretaria de Governo Digital do Ministério da Economia, em setembro de 2020, a digitalização de serviços voltados para a população em geral já abrange aproximadamente 60% dos cerca de 3,7 mil serviços públicos disponibilizados pelo portal do governo federal. É estimado que o governo economize cerca de 2,2 bilhões de reais por ano com a digitalização de seus serviços. Em 28 de abril de 2020 foi assinado o decreto número 10.332 que estabelece a Estratégia de Governo Digital 2020-2022 que cria a meta de digitalizar 100% dos serviços governamentais até 2022 (AGÊNCIA BRASIL, 2020).

Em 2020, segundo ranking da ONU, o Brasil é o país da América Latina com melhor disponibilidade de serviços digitais, além de figurar na lista dos vinte países com maior desenvolvimento de governo eletrônico. Isso é devido a investimentos que visam à agilização e desburocratização de serviços públicos que também causam uma diminuição no número de fraudes e erros humanos, podendo gerar uma economia estimada de R\$ 38 bilhões nos cofres públicos em cinco anos (LIMA, 2020).

Dentro desse cenário é necessário salientar que para implementar todas essas novas ferramentas digitais é preciso haver o suporte de um grande grupo de funcionários capacitados para seu desenvolvimento e manutenção. Dentre as tarefas realizadas por essas pessoas, uma é fundamental para manter os sistemas funcionando de maneira adequada que seria a de monitorar o funcionamento das aplicações. Um dos pilares dessa função está em analisar relatórios gerados pelos programas, os chamados arquivos de *log*, que trazem informações sobre o que ocorre durante a operação de variados tipos de sistemas como banco de dados, sistemas operacionais, aplicativos em geral, etc. Cada registro gravado no arquivo de *log* mostra informações que podem ser sobre eventos, processos ou mensagens que informam o que ocorre dentro dos programas, por exemplo, se foi feita uma

gravação ou leitura em um banco de dados ou se o usuário fez um *login* na aplicação. Todas essas informações são gravadas em formato de texto que pode ser lido de forma direta pelo usuário ou por outra aplicação (ROCK CONTENT, 2020).

Apesar do responsável pelo sistema poder ler o conteúdo de arquivos de *log* de forma manual, nem sempre esse trabalho é simples, pois muitas vezes esses arquivos tendem a ter muitas linhas de texto a serem lidas ou podem apresentar uma linguagem de difícil compreensão. Devido a esse problema que surgem várias soluções para facilitar o processo de análise de mensagens de *log*, sendo assim, este trabalho apresenta a criação de um sistema que monitora constantemente informações de *log* geradas por uma aplicação e exibe essas informações de forma simplificada para facilitar a análise feita pelo administrador.

Este trabalho propõe a criação de um sistema de monitoramento que irá ler arquivos de *log*, tratar seus dados e exibi-los através de painéis de visualização (*dashboards*) que são páginas com gráficos, tabelas e elementos visuais interativos que possibilitam ao usuário uma leitura facilitada e rápida, em tempo real, das informações de *log* da aplicação. O sistema proposto utilizará as aplicações gratuitas do grupo *Elastic Stack* que são: *Beats* e *Logstash* (leitura e tratamento de dados), *Elasticsearch* (armazenamento de dados) e *Kibana* (visualização de dados).

1.1 MOTIVAÇÃO

Um dos órgãos públicos que implementou uma solução para digitalizar algumas de suas atividades foi o Ministério Público do Rio Grande do Norte (MPRN), onde foi desenvolvido o “Sistema de Gestão de Usuários” que consiste em uma aplicação de gerenciamento eletrônico de documentos (GED).

Aplicações do tipo GED consistem em tecnologias responsáveis por facilitar principalmente o controle, armazenamento, compartilhamento e recuperação de dados pertencentes a uma instituição. Esse tipo de sistema permite que usuários trabalhem com documentos de forma totalmente digital e remota (DIGIX, 2020).

O Sistema de Gestão de Usuários do MPRN é utilizado em todo o estado do Rio Grande do Norte e auxilia diariamente uma grande quantidade de funcionários, por isso seu funcionamento é fundamental para as atividades da instituição.

Para garantir um melhor funcionamento desse sistema, esse trabalho propõe uma ferramenta de monitoramento de mensagens de *log* que tem o objetivo principal de se facilitar a análise dos dados gerados pela aplicação, em especial dados sobre erros durante a execução da aplicação e dados sobre os usuários que acessam o sistema. Essa ferramenta de monitoramento se faz necessária devido a grande quantidade de dados gerados pelo sistema que se tornam difíceis de analisar através de uma leitura direta dos arquivos de *log* gerados.

1.2 OBJETIVOS

Como objetivo geral, este trabalho propõe a criação de um sistema de monitoramento de dados de *log* da aplicação chamada de “Sistema de Gestão de Usuários” do MPRN.

O sistema de monitoramento consiste em um conjunto de ferramentas que têm o objetivo de extrair e tratar dados de *log* da aplicação e a partir deles gerar painéis de visualização, também conhecidos como *dashboards*, que exibem toda a informação captada através de um conjunto de elementos visuais interativos como gráficos, mapas e tabelas. Esse sistema tem a função de simplificar a leitura dos dados de *log* gerados pelo sistema.

Os dados analisados são coletados de três arquivos de texto gerados pela aplicação que tratam de acessos, alertas e requisições ao sistema e de uma ferramenta que obtém dados de métricas do servidor que hospeda a aplicação.

Com exceção dos dados de alertas que necessitam de dois painéis, é criado um painel para cada tipo de dado, totalizando cinco painéis que estão descritos a seguir:

- **Acessos de Usuários:** Informações relacionadas a quem acessou o sistema, incluindo dados de localização por comarca e município baseados pelo número *IP*;
- **Alertas do Sistema:** Categorização de mensagens de alerta criadas pelo sistema com o intuito principal de identificar as mensagens de erro da aplicação e encontrar erros incomuns;
- **Erros de Acesso de Usuários:** Uma subdivisão do *dashboard* anterior focando nos erros relacionados a acesso de usuários, fazendo uma

classificação desses erros e identificando quais usuários presenciaram esses erros;

- Requisições e acessos: Análise das requisições feitas para o sistema bem como a identificação do endereço *IP* e da localização do autor da requisição;
- Métricas *JMX*: Visualização de métricas do servidor da aplicação obtidas por um agente *JMX*.

Os dados de alertas do sistema são representados por dois painéis, pois além de visualizar os alertas em geral, o administrador da aplicação do MPRN também considera muito importante a análise mais detalhada de alertas de erros de acesso de usuários, sendo assim, para dar um maior destaque a esse tipo de alerta e evitar que um painel único fique muito poluído visualmente foi decidido por separar o conteúdo em dois painéis.

Os outros tipos de dados coletados possuem um painel exclusivo associado para cada um a fim de não misturar muitas informações diferentes em uma mesma página, o que poderia dificultar a compreensão do usuário que for visualizar a tela.

1.3 ORGANIZAÇÃO DO TRABALHO

Além deste capítulo de introdução, este trabalho está organizado em mais quatro capítulos. O capítulo 2 apresenta a descrição das ferramentas utilizadas e como funciona o sistema criado. O capítulo 3 mostra o desenvolvimento do projeto para cada cenário enquanto que o capítulo 4 detalha a análise dos dados utilizando os painéis criados e algumas ferramentas adicionais. Por fim, no capítulo 5, são apresentadas as considerações finais sobre o trabalho.

2 APRESENTAÇÃO DAS FERRAMENTAS

Todo o projeto utiliza ferramentas gratuitas de código aberto (*open source*). A seguir, na Figura 1, é apresentado o esquema de funcionamento completo do sistema de monitoramento e a função das principais ferramentas utilizadas.

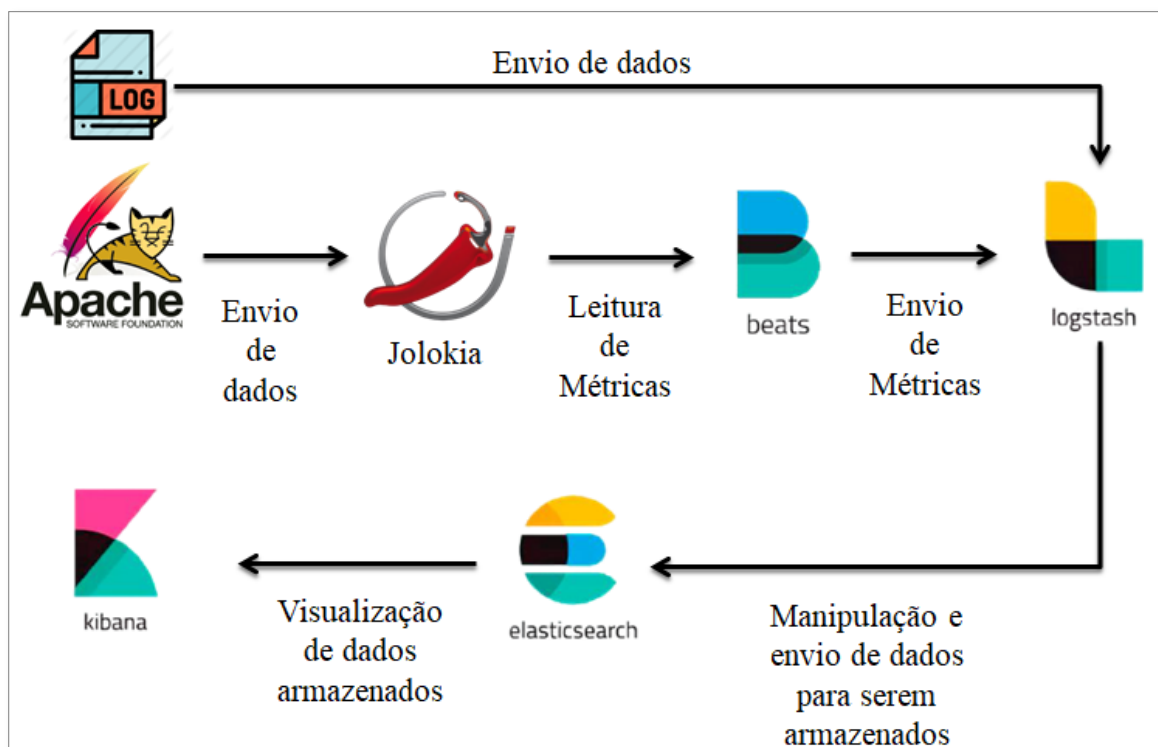


Figura 1 – Fluxograma de funcionamento do sistema de monitoramento.

A estrutura central do projeto é baseada no conjunto de aplicações "*Elastic Stack*", composta aqui pelas ferramentas Beats, Logstash, Elasticsearch e Kibana que realizam todas as atividades desde a coleta de dados até a criação de painéis de visualização.

Além da *Elastic Stack*, algumas ferramentas são necessárias para obter os dados de métricas do sistema que vêm diretamente do servidor *Apache Tomcat* que hospeda a aplicação do MPRN. Esses dados são captados pelo agente *Jolokia* que os envia para o *Beats*. Os outros dados provêm de três arquivos de texto com dados de *log* gerados automaticamente pela aplicação.

Todas essas ferramentas estão introduzidas com detalhes nas seções a seguir.

2.1 ELASTIC STACK

O conjunto de ferramentas que compõem o *Elastic Stack*, também conhecido como *ELK Stack*, são *Logstash*, *Beats*, *Elasticsearch* e *Kibana*. Esse conjunto surgiu a partir do *Elasticsearch* que é um motor de busca e armazenamento de dados, mas com a demanda de se armazenar arquivos de *log* foi criado o *Logstash* para processar esses arquivos, fazer o tratamento dos dados e enviá-los para um servidor *Elasticsearch*. Para facilitar a visualização desses dados foi criado o *Kibana* que é o responsável por acessar o banco de dados e gerar relatórios e visualizações simplificadas. Já os *Beats* surgiram posteriormente apenas para receber e enviar dados de algum tipo específico, nesse trabalho é utilizado o *Metricbeat*, subcategoria dos *Beats*, que é especializado em receber dados de métricas (ELASTIC, 2020).

2.1.1 Logstash

A obtenção, a preparação e a ingestão de dados são as atividades realizadas pelo *Logstash*. Ele lê os dados a partir de uma fonte que, no caso desse projeto, são arquivos de texto ou dados enviados pelo *Metricbeat*. Esses dados recebidos também são organizados e preparados com o uso de filtros que os deixam da maneira desejada para o armazenamento (LOGSTASH, 2020).

Após preparar os dados, o *Logstash* os envia em formato de texto, nesse caso, para um servidor *Elasticsearch* que os armazena. Além disso, de forma opcional, os dados podem ser gravados em um arquivo de texto que auxilia na detecção de problemas que podem ocorrer durante sua preparação.

Mais detalhes sobre o formato dos dados enviados são mostrados na seção 2.1.3 (*Elasticsearch*), enquanto que na seção 3.2 há um maior foco na configuração do *Logstash* e como é feito o tratamento dos dados por ele.

2.1.2 Beats

Os *Beats* são uma coleção de agentes que funcionam como coletores de dados que apenas recebem e enviam dados, recomendados para tratar um fluxo contínuo de grande quantidade de informações, utilizados para monitorar, na maioria dos casos, aplicações e serviços (BEATS, 2020).

Nesse projeto é utilizado o *Metricbeat*, que é um *beat* específico para a obtenção de métricas que são medidas de desempenho, como uso de recursos da máquina entre outras, nesse caso, obtidas de um agente *Jolokia* (mais detalhes na seção 2.3) que obtém métricas do servidor *Apache Tomcat*, detalhado na seção 2.2, que hospeda a aplicação do MPRN.

O *Metricbeat* obtém as métricas e as envia para o *Logstash* que faz um tratamento dos dados e os envia para serem armazenados no servidor *Elasticsearch*.

2.1.3 Elasticsearch

O *Elasticsearch* foi criado como uma ferramenta de busca construída com base na biblioteca *Apache Lucene* que é uma biblioteca de busca por texto de alta performance que exigia uma programação na linguagem *Java* para integrá-la diretamente a alguma aplicação.

Como a *Lucene* era considerada muito complexa, o *Elasticsearch* surgiu como uma ferramenta que permitia que se trabalhasse de forma mais simples com essa biblioteca, utilizando um código em *Java* e usando a *Lucene* internamente para classificação e busca dos dados. Esse processo é feito através de uma *API* em estilo *RESTful*.

API corresponde á um código que faz a comunicação entre dois softwares diferentes enquanto que *RESTful* compreende é um tipo de arquitetura de comunicação que utiliza o protocolo *HTTP* para fazer a comunicação entre as partes.

Em resumo, o *Elasticsearch* pode ser descrito como sendo um repositório de documentos que podem ter seus elementos indexados e buscados, uma ferramenta de busca distribuída com análise em tempo real além de ser capaz de

escalar até centenas de servidores e *petabytes* de dados estruturados e não estruturados (GORMLEY e TONG, 2014, p. 7).

Ao invés de armazenar informações no formato de linhas e colunas, o *Elasticsearch* armazena dados em documentos no formato JSON (ver Figura 2) onde cada documento é uma coleção de campos que são pares chave-valor que contém os dados.

```
{
  "_index" : "alertas",
  "_type" : "_doc",
  "_id" : "OtDuOHMBGuVZdYpMH314",
  "_score" : 1.0,
  "_source" : {
    "Aviso" : "INFO",
    "Mensagem" : "INFO [main] org.apache.catalina.startup.VersionLoggerListe
ner.log Server version: Apache Tomcat/9.0.12",
    "@timestamp" : "2019-02-18T13:58:36.146Z",
    "Categoria" : "Versão do Servidor"
  }
}
```

Figura 2 - Documento armazenado no *Elasticsearch*.

Para realizar a busca por texto em documentos, o *Elasticsearch* cria uma tabela com as palavras relevantes existentes e associa a elas uma lista com todos os documentos que possuem a palavra em questão, dessa maneira, o usuário busca por palavras e o *Elasticsearch* retorna os documentos da lista associada à palavra. Esse método é chamado de busca por *index* invertido (ver Figura 3).

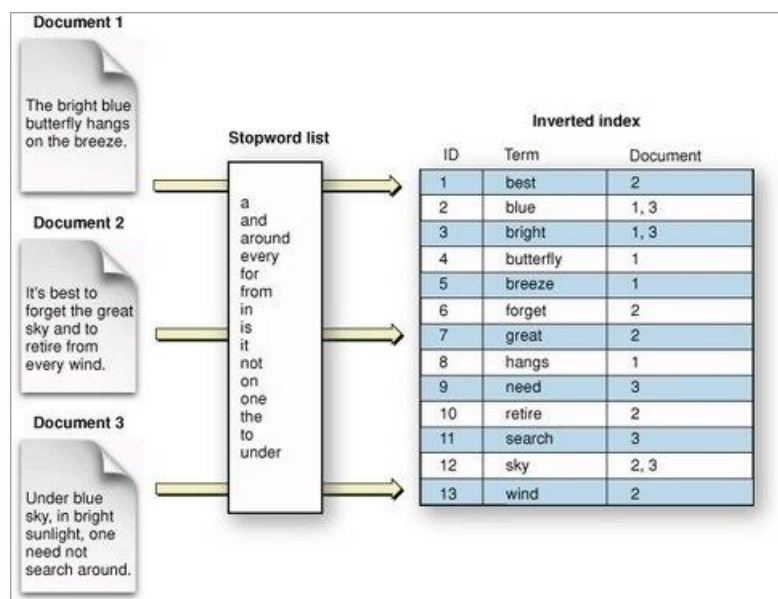


Figura 3 - Busca por *index* invertido no *Elasticsearch*.

Fonte : <https://mentormate.com/blog/what-is-elasticsearch/>

É utilizado um *index* para armazenar vários documentos e para facilitar as buscas, além disso, o Elasticsearch pode criar automaticamente subdivisões para o *index* chamadas de *shards* que são distribuídos em diversos nós gerenciados automaticamente pelo servidor. Cada *shard* armazena uma parte dos documentos do *index* tornando possível a distribuição de atividades entre eles.

Os documentos armazenados não precisam ter uma estrutura pré-determinada, cada documento pode ter diferentes campos mesmo se servirem ao mesmo propósito ou serem armazenados no mesmo *index* (KALYANI e MEHTA, 2017, p.21824-21826).

2.1.4 Kibana

O *Kibana* é uma interface de usuário (*GUI*) que permite a visualização e gerenciamento de dados contidos no servidor *Elasticsearch* de forma simples e intuitiva. Com ele é possível criar gráficos, tabelas, mapas, histogramas e vários outros tipos de visualizações que auxiliarão na análise de dados (KIBANA, 2020).

Nesse projeto, o *Kibana* é utilizado para acessar os dados que estão armazenados no *Elasticsearch* e para criar os painéis de visualização (*dashboards*).

2.2 FERRAMENTAS PARA A OBTENÇÃO DE MÉTRICAS

Como visto anteriormente, na Figura 1, os dados de métricas do sistema não são obtidos diretamente pelos componentes da *Elastic Stack*, eles são obtidos pelo agente *Jolokia* que os extrai do servidor *Apache Tomcat* que hospeda a aplicação do MPRN.

As seções a seguir introduzem essas duas ferramentas que trabalham em conjunto para enviar dados de métricas para a *Elastic Stack* que faz o tratamento desses dados e cria o painel de métricas.

2.2.1 Servidor Apache Tomcat

A aplicação monitorada nesse projeto, o “Sistema de Gestão de Usuários” do MPRN, encontra-se hospedada em um servidor *Apache Tomcat*. Devido a questões de segurança de informação, esse trabalho utiliza um servidor de testes similar ao encontrado no MPRN.

O servidor *Apache Tomcat* é uma aplicação *web* de *container* baseada em *Java* que foi criada para rodar aplicações *web* de *servlet* e *Java Server Pages* (JSP) (VUKOVIC e GOODWILL, 2011, p.1).

O *servlet* é uma classe que recebe requisições, processa-as e retorna uma resposta, por exemplo, ele pode receber dados digitados por um usuário em um formulário HTML, pode pesquisar informações em bancos de dado e também criar páginas *web* dinamicamente (BAELDUNG, 2018).

O JSP é uma tecnologia padrão do *Java* que permite a escrita de páginas dinâmicas e associadas com dados para aplicações *web* *Java*. Ele é construído com base no *servlet* e as duas tecnologias, geralmente, trabalham juntas. Do ponto de vista de codificação, a diferença entre os dois é que em *servlets* é escrito código *Java* que recebe marcações incorporadas do lado do cliente (como HTML) enquanto que no JSP é iniciado um *script* ou marcação que então recebe marcações JSP incorporadas que conectam a página ao *back-end Java* (TYSON, 2019).

Uma instância, ou servidor, do *Tomcat* consiste no agrupamento de *containers* de aplicação que são organizados em uma hierarquia bem definida onde o servidor está no seu nível mais alto. Apenas uma instância do servidor pode existir dentro de uma máquina virtual *Java* (JVM), isso garante maior segurança, pois vários servidores diferentes podem ser executados na mesma máquina física de forma independente utilizando portas diferentes, ou seja, se um deles travar, o funcionamento dos outros não é afetado (VUKOVIC e GOODWILL, 2011, p.3-4).

A máquina virtual *Java* (JVM) interpreta um código binário *Java* (chamado *bytecode*) para o processador de um computador poder executar instruções de um programa escrito em linguagem *Java*. Ela foi criada para permitir que aplicações baseadas em linguagem *Java* sejam executadas em qualquer ambiente sem que haja a necessidade de reescrita ou de uma nova compilação do código original (ROUSE, 2019).

Os componentes que compõem a arquitetura do *Apache Tomcat* se organizam de acordo com a hierarquia mostrada abaixo, na Figura 4.

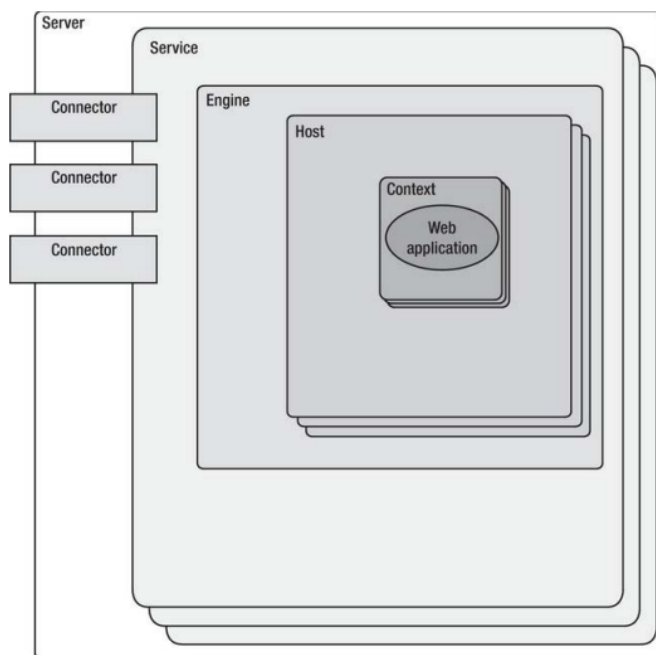


Figura 4 - Hierarquia dos componentes do servidor *Apache Tomcat*.

Fonte - (VUKOVIC e GOODWILL, 2011, p.5)

O servidor (*server*), que é o elemento que engloba todos os outros, pode possuir um ou mais contêineres de serviço (*service*) que, por sua vez, possuem uma coleção de um ou mais conectores (*connectors*) que compartilham um único motor (*engine*). Os conectores definem a classe que manipula requisições e respostas de e para uma chamada da aplicação cliente. Todas as requisições recebidas pelos conectores são gerenciadas pelo motor que é quem possui, em cada instância, diversos elementos de *hosts* que hospedam uma ou mais aplicações *web* representadas pelo elemento de contexto (*context*) (VUKOVIC e GOODWILL, 2011, p.3-6).

2.2.2 Agente Jolokia

O agente *Jolokia* é utilizado apenas para a criação do painel de métricas, onde ele é o responsável pela obtenção das métricas *JMX* provenientes do servidor *Apache Tomcat* de testes criado para este trabalho. O *Jolokia* é uma aplicação *web* que é hospedada pelo servidor e iniciada junto da iniciação do servidor para obter

as métricas de funcionamento dele, mais detalhes sobre como ele é executado estão na seção 3.1.4 (Iniciando o agente *Jolokia*).

Nesse caso, as métricas são informações relacionadas ao desempenho do servidor, como carga de CPU, tempo de execução, uso de memória. A seção 3.2.4 (Painel de Métricas) dá mais detalhes sobre as métricas obtidas.

A tecnologia Java *JMX* é uma ferramenta que permite monitorar e gerenciar aplicações Java. A *API* do *JMX* geralmente é acessada apenas pelo código *Java*, mas existem diversas ferramentas que tornam esses dados acessíveis ao usuário, uma delas é o agente *Jolokia* que cria uma ponte entre *JMX* e o protocolo *HTTP* com o benefício de alguns recursos, como políticas de segurança e recursos de requisições em massa (SORIANO, 2018).

O *Jolokia* faz a comunicação através de métodos *GET* ou *POST* do protocolo *HTTP* onde respostas e requisições são representadas no formato *JSON*. O protocolo de comunicação do *Jolokia* suporta as operações de leitura e escrita de atributos *JMX*, execução de operações *JMX*, busca por nomes de *MBean* através de padrão e listagem de meta-dados de *MBean* como atributos suportados, operações e notificações (HUSS, 2019).

O *MBean* é um objeto que representa um dispositivo, aplicação ou qualquer recurso que precise ser gerenciado. Ele é composto por um grupo de atributos de leitura e/ou escrita, um grupo de operações invocáveis e uma autodescrição (ORACLE, 2019).

2.3 FERRAMENTAS AUXILIARES

Para auxiliar o projeto é utilizado o editor de texto *Microsoft Visual Studio Code* que é usado para a escrita de códigos e visualização de arquivos de texto diversos.

A execução das ferramentas da *Elastic Stack* é feita através de linha de comando utilizando o console do *Windows Power Shell* que é uma ferramenta inclusa em alguns sistemas operacionais *Windows*.

Todas as visualizações das interfaces do *Kibana*, *Apache Tomcat* e status do *Elasticsearch* são feitas através de um navegador *web*. Nesse projeto é utilizado o *Mozilla Firefox*.

2.4 VERSÕES UTILIZADAS

Todas as ferramentas utilizadas nesse projeto são obtidas gratuitamente no site oficial de cada uma, a seguir, são apresentadas as versões de cada ferramenta. Para se reproduzir esse trabalho, é recomendada a utilização das mesmas versões aqui citadas ou de versões posteriores. Todo o trabalho foi realizado em um sistema operacional *Windows 10* de 64 *bits*.

A versão das ferramentas auxiliares (editor de código, console e navegador) não influi no projeto, mas é interessante utilizar suas versões mais atualizadas.

Para a reprodução deste trabalho em sistemas diferentes do *Windows* é preciso procurar as versões existentes que sejam compatíveis com o sistema operacional que for utilizado.

Versões das ferramentas utilizadas:

- *Logstash* 7.6.2 (LOGSTASH 7,2020);
- *Metricbeat* 7.6.2 (METRICBEAT 7,2020);
- *Elasticsearch* 7.4.2 (ELASTICSEARCH 7,2019);
- *Kibana* 7.4.2 (KIBANA 7,2019);
- *Jolokia* 1.6.2 versão *WAR-Agent* (JOLOKIA,2019);
- *Visual Studio Code* 1.45.1 (VISUAL, 2020) ;
- *Servidor Apache Tomcat* 8.5.54 (APACHE, 2020);
- *Mozilla Firefox* 80 (MOZILLA, 2020).

3 DESENVOLVIMENTO DO PROJETO

Para a criação dos painéis de visualização que compõem esse projeto de monitoramento são seguidas as seguintes etapas:

1. Preparação do ambiente (execução do *Elasticsearch*, *Kibana* e *Tomcat*);
2. Configuração do *Logstash* e do *Metricbeat* além da descrição do cenário avaliado por cada painel;
3. Obtenção, manipulação e envio de dados para o *Elasticsearch* utilizando *Logstash* e *Metricbeat*;
4. Conferência dos dados armazenados no servidor *Elasticsearch* através da ferramenta de descoberta de dados do *Kibana*;
5. Criação das visualizações no *Kibana*;
6. Agrupamento das visualizações formando os painéis de visualização com a ferramenta de *Dashboards* do *Kibana*.

Nesse projeto todas as ferramentas foram executadas na mesma máquina, mas em um ambiente com uma rede de computadores também é possível hospedar cada aplicação em uma máquina diferente desde que haja permissão para todas as máquinas envolvidas no processo compartilharem a sua aplicação. Também existe a possibilidade de hospedar esses serviços em nuvem com o serviço pago da *Elastic Cloud*.

As seções a seguir detalham as etapas de execução do projeto.

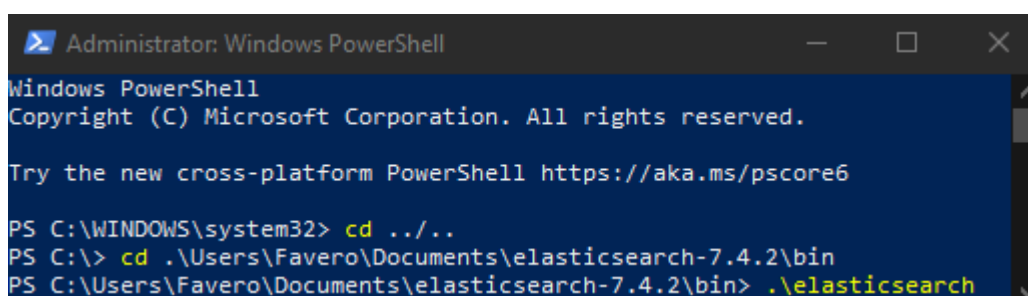
3.1 PREPARAÇÃO DO AMBIENTE

Antes de começar a manipulação dos dados para análise é preciso iniciar o servidor *Elasticsearch* e a interface do *Kibana*. E, no caso do painel de métricas, é preciso iniciar o servidor *Apache Tomcat* com o agente *Jolokia* ativo. É recomendada a realização dessas etapas na ordem que segue.

3.1.1 Iniciando o Servidor Elasticsearch

O Elasticsearch é obtido no formato compactado com a extensão “.zip”, sua instalação é feita pela descompactação de seu conteúdo que cria uma pasta de arquivos. Para esse projeto, a pasta criada é renomeada para "elasticsearch-7.4.2".

O servidor *Elasticsearch* é iniciado com a execução do arquivo "elasticsearch" contido na pasta "bin" através do console do *Power Shell* utilizando os comandos representados na Figura 5.



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

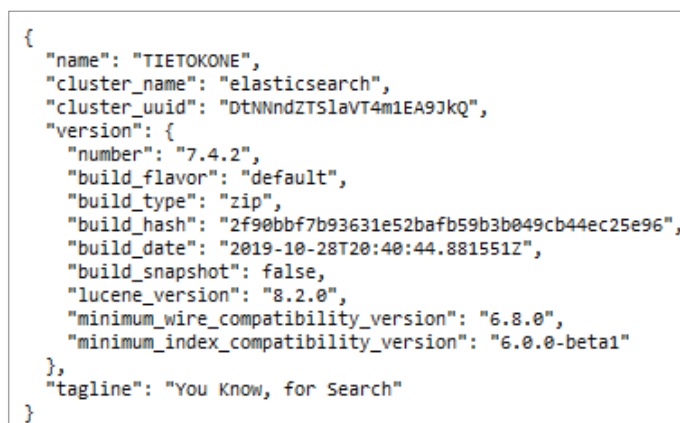
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> cd ../..
PS C:\> cd .\Users\Favero\Documents\elasticsearch-7.4.2\bin
PS C:\Users\Favero\Documents\elasticsearch-7.4.2\bin> .\elasticsearch
```

Figura 5 - Comandos para iniciar o servidor *Elasticsearch*.

Para conferir se o servidor está funcionando, no navegador *web* é acessada a *url* padrão "http://localhost:9200/", que é utilizada se o servidor estiver rodando na máquina local (*localhost*) e se estiver utilizando a porta padrão (9200).

Na Figura 6 é apresentada a mensagem de sucesso vista no navegador.

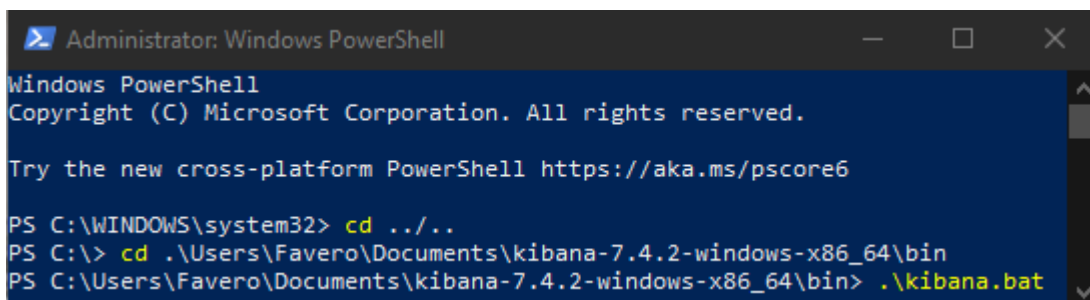


```
{
  "name": "TIETOKONE",
  "cluster_name": "elasticsearch",
  "cluster_uuid": "DtNNndZTslavT4m1EA9JkQ",
  "version": {
    "number": "7.4.2",
    "build_flavor": "default",
    "build_type": "zip",
    "build_hash": "2f90bbf7b93631e52bafb59b3b049cb44ec25e96",
    "build_date": "2019-10-28T20:40:44.881551Z",
    "build_snapshot": false,
    "lucene_version": "8.2.0",
    "minimum_wire_compatibility_version": "6.8.0",
    "minimum_index_compatibility_version": "6.0.0-beta1"
  },
  "tagline": "You Know, for Search"
}
```

Figura 6 - Mensagem de sucesso do *Elasticsearch*.

3.1.2 Iniciando o Kibana

O *Kibana* também é obtido no formato compactado e sua instalação e execução seguem os mesmos padrões da seção anterior. Os comandos utilizados no *Power Shell* estão representados na Figura 7.



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> cd ../../
PS C:\> cd .\Users\Favero\Documents\kibana-7.4.2-windows-x86_64\bin
PS C:\Users\Favero\Documents\kibana-7.4.2-windows-x86_64\bin> .\kibana.bat
```

Figura 7 - Comandos para iniciar o *Kibana*.

Quando ele estiver pronto para uso, sua interface é acessada por padrão pela url "<http://localhost:5600/>", representada na Figura 8.

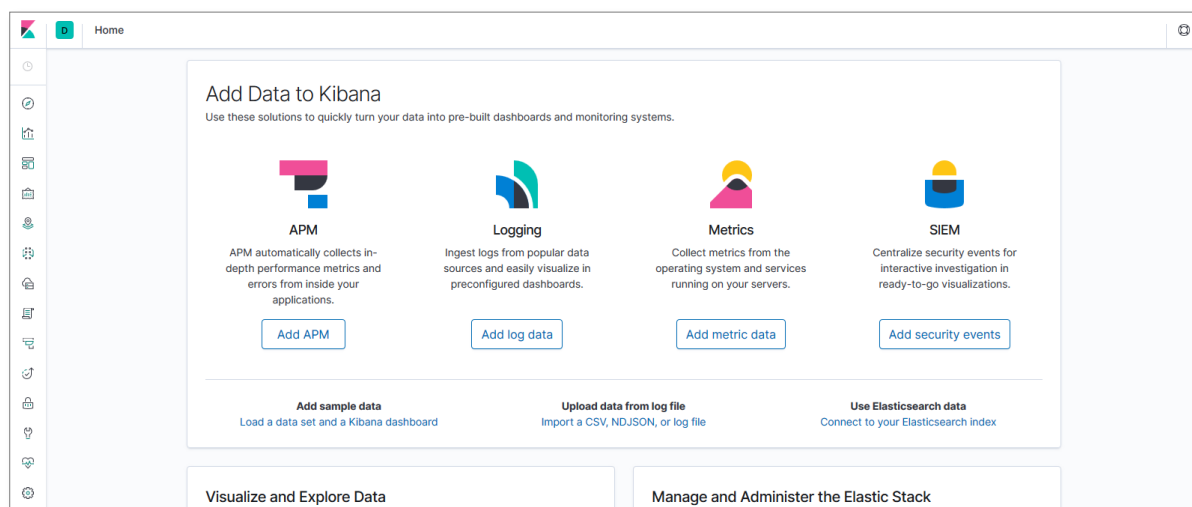


Figura 8 - Página inicial do *Kibana*.

3.1.3 Iniciando o Servidor Apache Tomcat

O servidor *Apache Tomcat* é obtido no formato executável de nome "[apache-tomcat-8.5.54.exe](#)", sendo assim, sua instalação é feita de maneira direta através da execução do arquivo utilizando as opções sugeridas por padrão.

A maneira mais simples de iniciar o servidor é através do executável "Tomcat8w.exe" encontrado por padrão em "C:\Program Files\Apache Software Foundation\Tomcat 8.5\bin". Esse executável mostra uma janela de propriedades (Figura 9) onde é possível iniciar o servidor com o botão "Start".

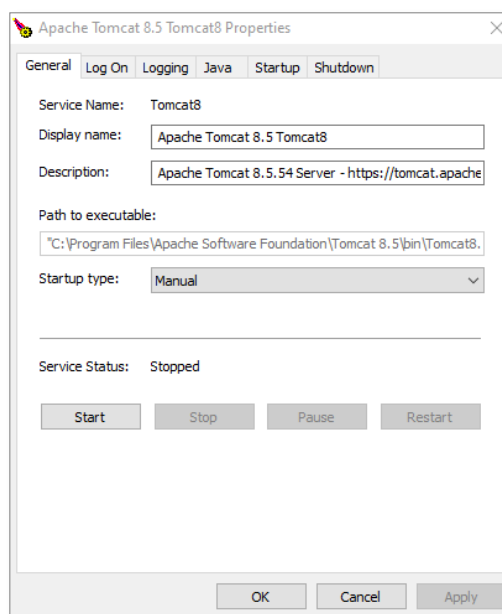


Figura 9 - Propriedades do *Tomcat* no Windows.

Para acessar a página inicial do servidor (Figura 10) e verificar se ele está em execução é acessada a *url*, por padrão, "http://localhost:8080/".

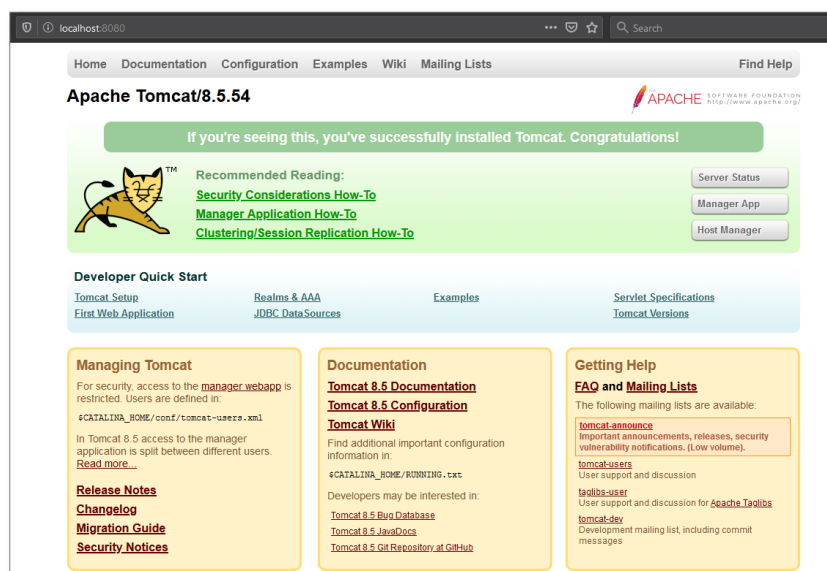


Figura 10 - Página inicial do servidor *Apache Tomcat*.

3.1.4 Iniciando o Agente Jolokia

O *Jolokia* é baixado no formato "*WAR-Agent*" (*jolokia-war-1.6.2.war*), essa versão possui uma camada de segurança onde o acesso dos dados só é permitido ao usuário do servidor *Tomcat* que tenha um papel (*role*) de "*jolokia*".

O *Jolokia* funciona como uma aplicação web hospedada pelo servidor, sendo assim, o arquivo "*jolokia-war-1.6.2.war*" deve ser armazenado no diretório "*webapps*" dentro da pasta principal do *Tomcat* (padrão *C:\Program Files\Apache Software Foundation\Tomcat 8.5\webapps*).

É adicionado, no *Tomcat*, um usuário de testes chamado "*jolokia*" com senha "*jolokia*" e o *role* "*jolokia*" (usuário e senha podem ser definidos com qualquer nome) que irá acessar os dados do agente.

A adição do usuário é feita com a adição de uma linha de texto no arquivo "*tomcat-users.xml*" encontrado por padrão em "*C:\Program Files\Apache Software Foundation\Tomcat 8.5\conf*". A nova linha é adicionada com um editor de texto como mostra a Figura 11.

```
<user username="jolokia" password="jolokia" roles="jolokia"/>
</tomcat-users>
```

Figura 11 - Documento criando usuário do *Tomcat*.

Com isso, o arquivo de usuários deve ser salvo e o servidor reiniciado, executando "*Tomcat8w.exe*" e utilizando a opção "*Restart*".

Para conferir se o *Jolokia* está em execução, é acessada a *url* "*http://localhost:8080/jolokia/*" e são inseridas as credenciais do usuário quando solicitado, nesse caso, nome: "*jolokia*" e senha: "*jolokia*".

A mensagem de sucesso está representada abaixo, na Figura 12.

```
{
  "request": {
    "type": "version",
    "value": {
      "agent": "1.6.2",
      "protocol": "7.2",
      "config": {
        "listenForHttpService": "true",
        "maxCollectionSize": "0",
        "authIgnoreCerts": "false",
        "agentId": "192.168.0.108-5624-2658ecc-servlet",
        "agentType": "servlet",
        "policyLocation": "classpath:/jolokia-access.xml",
        "agentContext": "\\jolokia",
        "mimeType": "text/plain",
        "discoveryEnabled": "false",
        "streaming": "true",
        "historyMaxEntries": "10",
        "allowDnsReverseLookup": "true",
        "maxObjects": "0",
        "debug": "false",
        "serializeException": "false",
        "detectorOptions": {}
      }
    },
    "dispatcherClasses": "org.jolokia.http.Jsr160ProxyNotEnabledByDefaultAnymoreDispatcher",
    "maxDepth": "15",
    "authMode": "basic",
    "authMatch": "any",
    "canonicalNaming": "true",
    "allowErrorDetails": "true",
    "realm": "jolokia",
    "includeStackTrace": "true",
    "useRestrictorService": "false",
    "debugMaxEntries": "100"
  },
  "info": {
    "product": "tomcat",
    "vendor": "Apache",
    "version": "8.5.54",
    "timestamp": 1595270021,
    "status": 200
  }
}
```

Figura 12 - Mensagem de sucesso do *Jolokia*.

3.2 DESCRIÇÃO DOS PAINÉIS E CONFIGURAÇÃO DO LOGSTASH

Antes de executar o *Logstash* para enviar os dados para o servidor *Elasticsearch*, é preciso criar um arquivo de configuração, com extensão “.conf”, que é responsável por determinar as ações que o *Logstash* realiza. Para a criação desse tipo de arquivo foi utilizado o editor de texto *Visual Studio Code*.

Para cada painel há um arquivo de configuração correspondente, esses arquivos estão representados na seção a seguir. Todos os arquivos de configuração possuem três funções principais: função *input* que recebe os dados, função *filter* que faz a preparação dos dados (organização, filtragem, etc) e a função *output* que envia os dados transformados para outra aplicação.

3.2.1 Painel de Acessos

O objetivo desse painel é mostrar a quantidade de acessos de cada usuário bem como mostrar os acessos por comarca e por município.

Para a sua criação foi analisado um arquivo de texto, em formato *csv*, com informações pessoais do usuário que acessou o sistema em determinada data. O arquivo é separado em quatro colunas (data, *IP*, matrícula e nome), como pode ser visto no exemplo abaixo:

```
"2019-01-15 16:24:14.473;172.26.8.15;1112;Hana Morenos".
```

A definição da localização de cada acesso foi feita a partir de uma tabela criada posteriormente que associa o número do *IP* do acesso com a comarca de origem.

A configuração do *Logstash* começa com a seção de entrada de dados (função *input*), vista na Figura 13, que aponta a localização do arquivo de *log* que é lido.

```
input{
  file{ ##Lendo arquivo Acessos.csv
    path=>"C:/Users/Favero/Documents/acessos/Acessos.csv"
    start_position => "beginning"
    sinedb_path => "NULL"
  }}

```

Figura 13 - Código da função *Input* recebendo *Acessos.csv*.

A manipulação dos dados é feita dentro da função *filter* (Figura 14) e começa com a nomeação dos campos, também chamados de colunas (*columns*), delimitados por ";".

```
filter {
  csv { ##Nomeando campos iniciais
    separator => ";"
    skip_header => true
    columns => ["Data_Hora", "Ip", "Matrícula", "Nome"]
  }
}
```

Figura 14 - Código nomeando campos do acesso.

É utilizada a função *date* (Figura 15) para converter data e hora para o formato de campo temporal (*timestamp*) utilizado pelo Kibana.

```
date { ##Usando Data_Hora como @timestamp
  match => ["Data_Hora", "yyyy-MM-dd' 'HH:mm:ss'.'SSS"]
  target => "@timestamp"
}
```

Figura 15 - Código convertendo data e hora.

Agora, na Figura 16, o campo *IP* tem sua primeira e terceira partes separadas para se fazer a associação com município e comarca posteriormente.

```
mutate { ##Copiando valores de Ip para Ip_tmp
  copy => { "Ip" => "Ip_tmp" }
}
mutate { ##Separando a primeira e a terceira parte
  split => ["Ip_tmp", "."]
  add_field => {"Ip_1" => "%{[Ip_tmp][0]}"}
  add_field => {"Ip_3" => "%{[Ip_tmp][2]}"}
}
if [Ip_1] == "172" or [Ip_1] == "0:0:0:0:0:0:0:1" { ##Ips do Anexo 5
  mutate { ##Ip_3 alterado para auxiliar a classificação
    replace => {"Ip_3" => "999"}
  }
}
```

Figura 16 - Código manipulando o *IP*.

A classificação do local do acesso é feita comparando o valor de "Ip_3" com arquivos de dicionário que contém a comarca e o município correspondentes através da função *translate*. Valores de *IP* sem correspondência no dicionário recebem a palavra "Desconhecido" no respectivo campo que o dicionário servir para classificar.

Os dicionários estão representados nas Figuras 17 (comarca) e 19 (município), enquanto que as funções de conversão estão nas Figuras 18 e 20.

```
166,Parnamirim
167,Tororós
169,Poço Branco
999,PGJ - Anexo 5
```

Figura 17 - Trecho do arquivo de dicionário de comarcas.

```
translate{ ##Associando Ip_3 e Comarca
  field => "[Ip_3]"
  destination => "[Comarca]"
  dictionary_path => "C:/Users/Favero/Documents/data/ip3_comarca.csv"
  fallback => "Desconhecido"
}
```

Figura 18 - Código da função *translate* para criar a comarca.

```
166,Parnamirim
167,Natal
169,Poço Branco
999,Natal
```

Figura 19 - Trecho do arquivo de dicionário de municípios.

```
translate{ ## Associando Ip_3 e Município
  field => "[Ip_3]"
  destination => "[Município]"
  dictionary_path => "C:/Users/Favero/Documents/data/ip3_cidade.csv"
  fallback => "Desconhecido"
}
```

Figura 20 - Código da função *translate* para criar o município.

O mesmo processo é feito para criar o campo de coordenadas a partir do campo de município. O dicionário utilizado é representado na Figura 21 enquanto que a função de criação do novo campo é exibida na Figura 22.

```
Natal,-05.79/-35.25
Extremoz,-05.70/-35.31
Taipú,-05.62/-35.59
Touros,-05.19/-35.46
```

Figura 21 - Trecho do arquivo de dicionário de coordenadas.

```
translate{ ## Associando Município e Coordenada
  field => "[Município]"
  destination => "[Coord]"
  dictionary_path => "C:/Users/Favero/Documents/data_sigilo/cidade_coord.csv"
  fallback => "Desconhecido"}
```

Figura 22 - Código da função *translate* para criar a coordenada.

Agora o campo de coordenada é adaptado para o padrão usado no *Kibana* (ver Figura 23).

```
if [Coord] != "Desconhecido"{
mutate{ ## Separando latitude e longitude
  split => ["Coord","/"]
  add_field => {"lat" => "%{[Coord][0]}"}
  add_field => {"lon" => "%{[Coord][1]}"}
}
mutate { ## Criando as partes do campo location
  rename => {
    "lon" => "[location][lon]"
    "lat" => "[location][lat]"
  }}
}
```

Figura 23 - Código criando o campo *location* no padrão do Kibana.

O laço *filter* e as operações de transformação dos dados são fechados com a remoção de campos que não serão mais utilizados (ver Figura 24).

```
mutate{ ## Removendo campos
  remove_field => ["Data_Hora","Ip_tmp","Ip_3","Ip_1","Ip"]
}
} ## Fechando o laço "filter"
```

Figura 24 - Código removendo campos que não serão mais utilizados.

Para finalizar a configuração, os dados são enviados para o *Elasticsearch* e para um arquivo de texto na função de saída (*output*), detalhes na Figura 25.

```
output {
if "_dateparsefailure" not in [tags] { ##Removendo documentos com erro
  elasticsearch { ##Enviando para o Elasticsearch
    index => "acessos-v1"
    manage_template => "false"
    document_type => "_doc"
    template_name=>"acessos_template"
    hosts => "http://localhost:9200"
  }
}
file{ ##Enviando para arquivo
  path => "C:/Users/Favero/Documents/acessos/arquivo.txt"
}
stdout { } }}
```

Figura 25 - Código para saída dos dados de acessos.

3.2.2 Painéis de Alertas do Sistema e Erros de Acesso de Usuários

No painel de alertas do sistema são exibidos eventos informados pelo *servlet Catalina* associado ao servidor *Tomcat*, esses eventos são informações relacionadas ao funcionamento da aplicação e se dividem em três categorias:

1. *Info*: quando há apenas uma informação geral relacionada à aplicação, como versão do Sistema Operacional, nome do servidor, etc;
2. *Warning*: aviso de quando algum detalhe da execução deve ser visto, geralmente informa algo que pode causar um erro no futuro e requer atenção;
3. *Severe*: informa um erro grave na aplicação que deve ser visto logo.

Cada linha de texto do arquivo de *log* que é considerada para análise contém, respectivamente, informações de data, hora, categoria, *thread* pertencente e mensagem do evento. Todas as linhas fora desse padrão são desconsideradas.

Exemplo de linha de *log*:

```
18-Feb-2019      08:56:38.828      INFO      [main]
org.apache.coyote.AbstractProtocol.init      Initializing
ProtocolHandler ["https-jsse-nio-8443"]
```

Além de armazenar as informações contidas no texto, também são criadas subcategorias para agrupar mensagens de conteúdo similar. Uma dessas subcategorias, a de "Erro de acesso de usuário", que engloba vários problemas que impedem o *login* de um usuário, é considerada muito importante para o administrador do sistema e pode ser dividida em algumas subcategorias mais específicas, por isso, é também criado o painel de erros de acesso de usuários para estudar mais atentamente esse tipo de erro.

As mensagens relacionadas a esse tipo de erro possuem a matrícula do usuário e o tipo de erro de acesso. Essas informações são mapeadas e o número de matrícula é usado como chave para um dicionário que liga nome de usuários com matrículas.

Exemplo de mensagem:

```
"12-Jul-2019 15:19:29.329 SEVERE [https-jsse-nio-8443-exec-8]mprn.home.view.controle.util.log.LogControlUtil.escreveLog Tentativa de autenticacao de [1095]. Usuario ou senha nãõ confere."
```

Nesse caso pode-se identificar a matrícula como sendo 1095 e a classificação como "Usuário ou senha não confere". Erros de escrita gerados no arquivo de *log* são corrigidos no arquivo de configuração do *Logstash*.

Dessa maneira, tanto o painel de alertas quanto o de erros de usuários se baseiam no mesmo arquivo de *log* "catalina_prod.out", por isso é criada apenas uma configuração do *Logstash* que serve de base para os dois.

A configuração começa com a entrada de dados padrão vista na Figura 26.

```
input{
  file{ ##Recebendo o arquivo catalina_prod.out
    path => "C:/Users/Favero/Documents/alertas/catalina_prod.out"
    start_position => "beginning"
    sinedb_path => "NULL"
  }
}
```

Figura 26 - Código da função *input* recebendo *catalina_prod.out*.

Cada parte da mensagem é associada a um campo com base em sua posição na linha de texto (ver Figura 27).

```
filter {
  dissect{ ##Associando campos com partes da mensagem
    mapping => {
      "message" => "%{Data_Hora} %{+Data_Hora} %{Aviso} [%{Thread}] %{Msg}"
    }
  }
}
```

Figura 27 - Código da função *dissect*.

A data e a hora são convertidas para o padrão do Kibana na função *date*, (ver Figura 28).

```
date { ## Modelo 18-Feb-2019 08:56:38.780
  match => ["Data_Hora", "dd-MMM-yyyy' 'HH:mm:ss'.'SSS"]
  target => "@timestamp"
}
```

Figura 28 - Código da função *date*.

Agora é feita a criação do campo "Categoria" que classifica mensagens de conteúdo semelhante, essas categorias foram criadas manualmente analisando as mensagens encontradas no arquivo de *log*.

Exemplo de classificação:

Frase:"org.apache.catalina.startup.VersionLoggerListener.1
og Server version: Apache Tomcat/9.0.12", categoria: "Versão do Servidor".

É utilizado um dicionário diferente para cada tipo de aviso (*info*, *severe*, *warning*) que compara trechos das mensagens (campo *Msg*). Um trecho de dicionário pode ser visto na Figura 29 enquanto que as funções de classificação das categorias *info* e *severe* são mostradas nas Figuras 30 e 31, respectivamente.

```
Starting ProtocolHandler,Iniciando protocolo
Using a shared selector for servlet,Seletor compartilhado
Initialization processed,Tempo de inicialização
```

Figura 29 - Trecho do arquivo de dicionário de categorias de alertas.

```
if [Aviso] == "INFO"{
translate{ ## Classificando alertas de tipo "INFO"
  exact => true
  regex => true
  field => "[Msg]"
  destination => "[Categoria]"
  dictionary_path => "C:/Users/Favero/Documents/alertas/categoria_info.csv"
"
  fallback => "Desconhecida"
}
```

Figura 30 - Código classificando alertas de tipo "INFO".

```
}else if [Aviso] == "SEVERE"{
  translate{ ## Classificando alertas de tipo "SEVERE"
    exact => true
    regex => true
    field => "[Msg]"
    destination => "[Categoria]"
    dictionary_path => "C:/Users/Favero/Documents/alertas/categoria_severe.csv"
"
    fallback => "Desconhecida"
}
```

Figura 31 - Código classificando alertas de tipo "SEVERE".

Entre as categorias relacionadas ao aviso *severe* se encontra a de "Erro de Autenticação de Usuário" que é a base para o painel de erros de usuários. Nesse

caso, é feito um mapeamento, visto na Figura 32, que determina os campos de matrícula do usuário e de categoria do erro do usuário.

```

if [Categoria] == "Erro de Autenticação de Usuário"{
    dissect{
        mapping => { ## Mensagem de exemplo:
##15-Jul-2019 09:34:36.895 SEVERE [https-jsse-nio-8443-exec-
223] mprn.home.view.controle.util.log.LogControlUtil.escreveLog Tentativa de a
utenticacao de [10003]. Usuiçário ou senha nãõ confere.
        "message" => "%{} {} {} [%{}] {}.%{}.%{}.%{}.%{}.%{}.%{}.%{}.%{}
} {} {} {} {} [%{Matrícula}]. %{Categoria_Erro_Usuário}"
    }}
    mutate{
        gsub => [## Substituindo erros comuns encontrados:
        "Categoria_Erro_Usuário","Usuãçrio","Usuário",
        "Categoria_Erro_Usuário","nãõ","nãõ",
        "Categoria_Erro_Usuário","nãõ","nãõ",
        "Categoria_Erro_Usuário","indisponível","indisponível",
        "Categoria_Erro_Usuário","autenticaiçõ","autenticação",
        "Categoria_Erro_Usuário","Usuiçário","Usuário"]}
        translate{ ## Definindo o nome dos usuários:
        exact => true
        field => "[Matrícula]"
        destination => "[Usuário]"
        dictionary_path => "C:/Users/Favero/Documents/alertas/users.csv"
        fallback => "Desconhecido"}
    }
}

```

Figura 32 - Código classificando erros de usuários.

Depois é feita a classificação dos alertas do tipo *warning* e campos desnecessários são removidos (ver Figura 33).

```

}}else {
    translate{ ## Classificando alertas de tipo "WARNING"
    exact => true
    regex => true
    field => "[Msg]"
    destination => "[Categoria]"
    dictionary_path =>"C:/Users/Favero/Documents/alertas/categoria_warning.c
sv"
    fallback => "Desconhecida"
    }}
    mutate{
        remove_field => ["Data_Hora","column1"]
    }}
}

```

Figura 33 - Código classificando alertas de tipo "WARNING" e retirando campos.

A saída de dados é feita de forma padrão para o *Elasticsearch* e para um arquivo de texto através da função *output* (ver Figura 34).

```
output {
  if "_dateparsefailure" not in [tags] and "_dissectfailure" not in [tags]{ ##Re
  movendo documentos com data inválida
    elasticsearch {
      hosts => "http://localhost:9200"
      index => "alertas"
      manage_template => "false"
      document_type => "_doc"
    }
  }
  file{
    path => "C:/Users/Favero/Documents/alertas/arquivo.txt"
  }
  stdout {}}
```

Figura 34 - Código de saída dos dados de alertas.

3.2.3 Painel de Requisições

Esse painel é responsável por analisar as requisições feitas ao servidor, identificar a localização dos *IPs* que fizeram cada requisição, contabilizar o total de *bytes* enviados e de requisições além da proporção de métodos utilizados.

O arquivo de texto analisado é um *log* de requisições padrão de um servidor *Apache Tomcat*, seu formato é o seguinte:

```
"172.26.3.58 - - [18/Mar/2020:07:03:28 -0300] "GET
/HOME/menu.jsp HTTP/2.0" 200 24763".
```

Nesse padrão são identificados, respectivamente, *IP* do usuário, data e hora, método utilizado, objeto requerido, versão do protocolo, código de resposta recebido e *bytes* enviados. A entrada de dados é similar à vista na seção 3.2.2 (ver Figura 35).

```
input{
  file{ ##Recebendo o arquivo req_log.txt
    path=>"C:/Users/Favero/Documents/data_2/Acess_log/req_log.txt"
    start_position => "beginning"
    sinedb_path => "NULL"
  }}
}
```

Figura 35 - Código de função input recebendo req_log.txt.

Os campos são criados com um filtro *Grok* (Figura 36) que reconhece esse tipo de padrão de mensagem e cria os campos de forma automática.


```
filter {
  grok { ##Filtrando com o filtro grok
    match => { "message" => "%{COMMONAPACHELOG}" } }
}
```

Figura 36 - Código de filtro *Grok*.

É feita a conversão de data e hora para o formato do *Kibana* (ver Figura 37).

```
date { ##Exemplo: 18/Mar/2020:07:03:28 -300
  match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"] }
```

Figura 37 - Código de função *date*.

Depois, alguns campos desnecessários são removidos e alguns campos são renomeados (ver Figura 38).

```
mutate{ ##Removendo campos
  remove_field => ["auth","ident","timestamp","httpversion","time"]
}
mutate{ ##Renomeando campos
  rename => {"clientip" => "Ip"}
  rename => {"verb" => "Método"}
  rename => {"bytes" => "Bytes enviados"}
  rename => {"response" => "Resposta"}
  rename => {"request" => "URL"} }
```

Figura 38 - Código removendo e renomeando campos.

É feito o mesmo processo visto no painel de acessos (seção 3.2.1), que utiliza a função *translate* para criar os campos de município e comarca baseados no *IP* (ver Figura 39).

```
mutate { ##Copiando Ip em Ip_tmp
  copy => { "Ip" => "Ip_tmp" }
}
mutate { ##Separando Ip_tmp e criando dois campos com suas partes
  split => ["Ip_tmp", "."]
  add_field => {"Ip_1" => "%{[Ip_tmp][0]}"}
  add_field => {"Ip_3" => "%{[Ip_tmp][2]}"}
}
if [Ip_1] == "172" or [Ip_1] == "0:0:0:0:0:0:0:1"{
  mutate{
    replace => {"Ip_3" => "999"}
  }}
translate{ ##Criando "Comarca" a partir de dicionário
  field => "[Ip_3]"
  destination => "[Comarca]"
  dictionary_path => "C:/Users/Favero/Documents/data_2/Acess_log/ip3.csv"
  fallback => "Desconhecido"
}
translate{ ##Criando "Município" a partir de dicionário
  field => "[Ip_3]"
  destination => "[Município]"
  dictionary_path => "C:/Users/Favero/Documents/data_2/Acess_log/ip3_cidade.csv"
  fallback => "Desconhecido" }
```

Figura 39 - Código criando campos de comarca e município.

Para finalizar a etapa de manipulação de dados, como mostra a Figura 40, alguns campos são removidos e o campo de "Bytes enviados" é convertido para o tipo inteiro (*integer*), devido a operações que serão feitas no *Kibana*.

```
mutate{ ##Removendo campos
  remove_field => ["Ip_tmp","Ip_1","Ip_3"]
}
mutate{ ##Convertendo "Bytes enviados" de texto para inteiro
  convert => {
    "Bytes enviados" => "integer"
  }}}
```

Figura 40 - Código removendo e convertendo campos.

A saída de dados é feita da mesma maneira vista nos painéis anteriores (ver a Figura 41).

```
output {
  elasticsearch { ##Enviando para o Elasticsearch
    index => "request"
    manage_template => "false"
    document_type => "_doc"
    hosts => "http://localhost:9200"
  }
  file{ ##Enviando para arquivo de texto
    path => "C:/Users/Favero/Documents/data_2/Acess_log/teste10.txt"
  }stdout {}
}
```

Figura 41 - Código de saída dos dados de requisições.

3.2.4 Painel de Métricas

Esse painel realiza o monitoramento de métricas relacionadas ao funcionamento do servidor *Apache Tomcat* que hospeda a aplicação.

As seguintes métricas são monitoradas:

- Carga de Cpu (*CPU Load*) : A carga de processos que a *cpu* está processando. Por exemplo, um *core* de *cpu* tem uma carga máxima igual a um. Se for medido um valor de *CPU Load* de 0,5 isso quer dizer que o processador só está utilizando metade do seu poder de processamento, se for criado um novo processo, ele será executado automaticamente sem necessidade de espera. Quando o valor de *CPU Load* for acima de um, isso quer dizer, novos processos deverão entrar em uma fila para serem executados, pois o processador já está usando o máximo de sua capacidade. O valor

máximo equivale ao número de *cores*, um processador *single core* tem a carga um, *dual core* tem dois e assim por diante (LEWIS, 2019);

- *Memory Heap*: Espaço de memória alocada para todas as instâncias de classe e *arrays*, criado pela máquina virtual Java (*JVM*) em sua inicialização e seu tamanho pode ser regulado pelo administrador do sistema. O seu tamanho máximo padrão é de 64 MB. Essa memória armazena objetos *Java* (*JVM*, 2020);
- *Memory non Heap*: Também criada pela *JVM* em sua inicialização, armazena estruturas de cada classe como código para métodos e construtores além de outros metadados (*JVM*, 2020);
- *Uptime*: Tempo em que o servidor está em execução.

Ao contrário dos códigos de configuração vistos anteriormente, a entrada de dados nesse caso é feita a partir do *Metricbeat*, que recebe os dados obtidos pelo agente *Jolokia* que é executado junto do servidor *Apache Tomcat*, e os envia, por padrão, para a porta 5044 como mostra a Figura 42 a seguir.

```
input { ##Recebendo dados do Metricbeat na porta 5044
  beats {
    port => 5044
  }
}
```

Figura 42 - Código de função *input* recebendo dados da porta 5044.

É feita apenas uma manipulação, no campo "*CPU Load*" que é multiplicado por cem para transformá-lo em porcentagem (ver Figura 43).

```
filter {
  if [metricset][namespace] == "composite-services-n1" {
    ruby {
      code => 'event.set("cpu.usage", event.get("[jolokia][composite-services-n1][jvm][process][cpu][load]") * 100)'
    }
  }
}
```

Figura 43 - Código manipulando os dados de métricas.

A saída de dados é a padrão com adição do interpretador de código *ruby* para interpretar a operação feita anteriormente (ver Figura 44).

```
output { stdout { codec => rubydebug }
  file{ ##Enviando para arquivo de texto
    path => "C:/Users/Favero/Documents/jolokia_log/teste.txt"}
  elasticsearch { ##Enviando para o Elasticsearch
    index => "jmx_v1"
    manage_template => "false"
    document_type => "_doc"
    hosts => ["localhost:9200"] }}
```

Figura 44 - Código enviando dados de métricas.

3.3 CONFIGURAÇÃO E EXECUÇÃO DO METRICBEAT

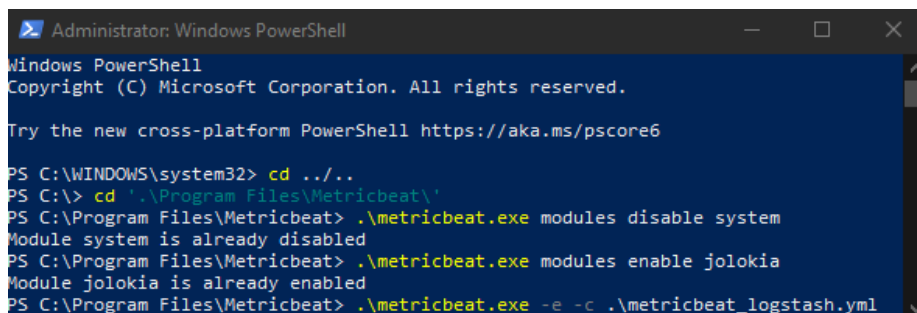
O *Metricbeat* é utilizado apenas pelo painel de métricas, ele é encontrado no formato compactado em seu site de download, sua instalação é feita apenas com a descompactação de seu conteúdo, é recomendado mover a pasta criada para "Arquivos de Programa" e renomeá-la para "Metricbeat".

O *Metricbeat* possui um módulo específico para coletar métricas do *Jolokia*, mas nesse caso, é utilizada uma versão com algumas alterações nomeada "jolokia.yml" que deve ser armazenada na pasta "modules.d". A nova versão do arquivo "jolokia.yml" está representada na Figura 45.

```
- module: jolokia
  metricsets: ["jmx"]
  period: 10s
  hosts: ["localhost:8080"]
  namespace: "metrics"
  path: "/jolokia/?ignoreErrors=true&canonicalNaming=false"
  username: "jolokia"
  password: "jolokia"
  jmx.mappings:
  - mbean: 'java.lang:type=Runtime'
    attributes:
    - attr: Uptime
      field: uptime
  - mbean: 'java.lang:type=Memory'
    attributes:
    - attr: HeapMemoryUsage
      field: memory.heap_usage
    - attr: NonHeapMemoryUsage
      field: memory.non_heap_usage
  - mbean: 'java.lang:type=OperatingSystem'
    attributes:
    - attr: ProcessCpuLoad
      field: jvm.process.cpu.load
  jmx.application:
  jmx.instance:
```

Figura 45 - Módulo *Jolokia* do *Metricbeat*.

Com esse módulo atualizado já é possível executar o *Metricbeat*, para isso, é utilizado o console do *Power Shell* com os comandos exibidos na Figura 46.



```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> cd ../../
PS C:\> cd '.\Program Files\Metricbeat\'
PS C:\Program Files\Metricbeat> .\metricbeat.exe modules disable system
Module system is already disabled
PS C:\Program Files\Metricbeat> .\metricbeat.exe modules enable jolokia
Module jolokia is already enabled
PS C:\Program Files\Metricbeat> .\metricbeat.exe -e -c .\metricbeat_logstash.yml

```

Figura 46 - Executando o *Metricbeat* com *Jolokia* habilitado.

3.4 PREPARAÇÃO DO KIBANA

Antes de executar o *Logstash*, apenas no caso do painel de acessos, é preciso inserir uma configuração que tornar o campo "*location*" como tipo "*geo_point*" que é o tipo de dado utilizado para gerar coordenadas na visualização do tipo mapa. Para isso, é inserido e executado o seguinte código no console do *Kibana* que se encontra na seção "*Dev_Tools*".

O código visto na Figura 47, a seguir, cria um *template* padrão que aplica essas alterações.

```

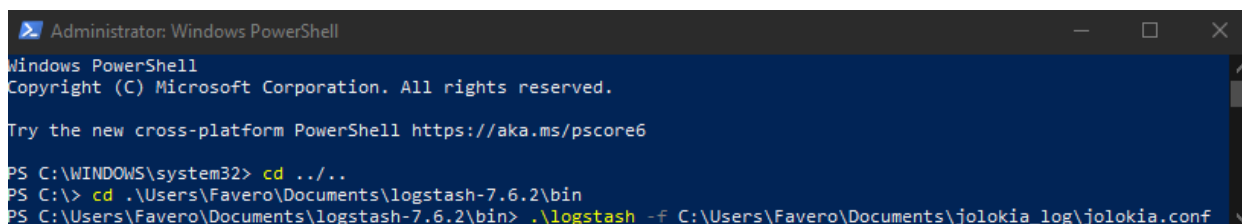
PUT _template/acessos_template
{
  "index_patterns": ["acessos-*"],
  "settings": { "index": { "refresh_interval": "5s" } },
  "mappings": { "dynamic_templates": [
    { "message_field": {
      "path_match": "message",
      "match_mapping_type": "string",
      "mapping": { "type": "text", "norms": false } } },
    { "string_fields": {
      "match": "*",
      "match_mapping_type": "string",
      "mapping": {
        "type": "text",
        "norms": false,
        "fields": {
          "keyword": { "type": "keyword", "ignore_above": 256 } } } } } ],
  "properties": {
    "@timestamp": { "type": "date" },
    "@version": { "type": "keyword" },
    "location": { "type": "geo_point" },
    "tags": { "type": "keyword" }
  }, "aliases": { } }

```

Figura 47 - *Template* padrão para o *index* de acessos.

3.5 EXECUÇÃO DO LOGSTASH

O *Logstash* é executado através do console do *Power Shell*, além de executar a aplicação, é apontado o caminho para o respectivo arquivo de configuração. Para todos os painéis, a execução é feita da mesma maneira, mas havendo apenas a alteração do caminho do arquivo de configuração utilizado como mostra a Figura 48.



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> cd ../../
PS C:\> cd .\Users\Favero\Documents\logstash-7.6.2\bin
PS C:\Users\Favero\Documents\logstash-7.6.2\bin> .\logstash -f C:\Users\Favero\Documents\jolokia_log\jolokia.conf
```

Figura 48 - Exemplo de execução do *Logstash*.

3.6 CRIAÇÃO DO INDEX PATTERN NO KIBANA

Depois da execução do *Logstash*, quando os dados estiverem sendo recebidos pelo *Elasticsearch*, já é possível manipulá-los no *Kibana*. A primeira coisa a se fazer é criar um "*Index Pattern*" que é um padrão que recebe os dados de um *index* do *Elasticsearch* e organiza esses dados para visualização e uso na criação de *dashboards*.

O *Index Pattern* é criado no menu de gerenciamento (*Management*), seguindo os sub menus de "*Index Patterns*" e "*Create Index Patterns*".

O nome do *index pattern* deve ser igual ou englobar o nome do *index* correspondente no *Elasticsearch*. Além disso, é perguntado depois se há um campo de filtro temporal (*Time filter field*), em todos os casos desse projeto, esse campo será "*@timestamp*" que é o campo de data e hora lido pelo *Kibana* para gerar análise, gráficos e filtros baseados no tempo.

Para exemplificar, o *index pattern* associado aos dados de acessos, nomeado "acessos-v1" está representado na Figura 49, exibida a seguir, onde é possível ver parte da tabela com os campos que pertencem a esse *index pattern*.

acessos-v1*

Time Filter field name: @timestamp

This page lists every field in the **acessos-v1*** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch [Mapping API](#)

Fields (24) | Scripted fields (0) | Source filters (0)

Q Filter All field types

Name	Type	Format	Searchable	Aggregatable	Excluded
@timestamp	date		•	•	✎
@version	string		•	•	✎
Comarca	string		•		✎
Comarca.keyword	string		•	•	✎
Coord	string		•		✎
Coord.keyword	string		•	•	✎
Matricula	string		•		✎
Matricula.keyword	string		•	•	✎
Municipio	string		•		✎
Municipio.keyword	string		•	•	✎

Rows per page: 10 < 1 2 3 >

Figura 49 - Index Pattern para o painel de acessos.

3.7 DESCOBERTA DE DADOS NO KIBANA

Com a criação do *index pattern*, já é possível manipular os dados no *Kibana*, mas antes de fazer qualquer operação, é recomendado visualizar esses dados para conferência através da ferramenta de descoberta na seção "*Discover*".

Caso nenhum dado esteja visível, é importante verificar o filtro de tempo no canto superior da tela e ajustá-lo de acordo com os dados, nesse projeto foi preciso utilizar "*last 2 years*" (últimos dois anos) pois todos os dados estão em um intervalo de até dois anos antes desse projeto ser realizado (a maioria dos dados são de 2019).

Dados descobertos de "acessos-v1" podem ser vistos abaixo, na Figura 50.

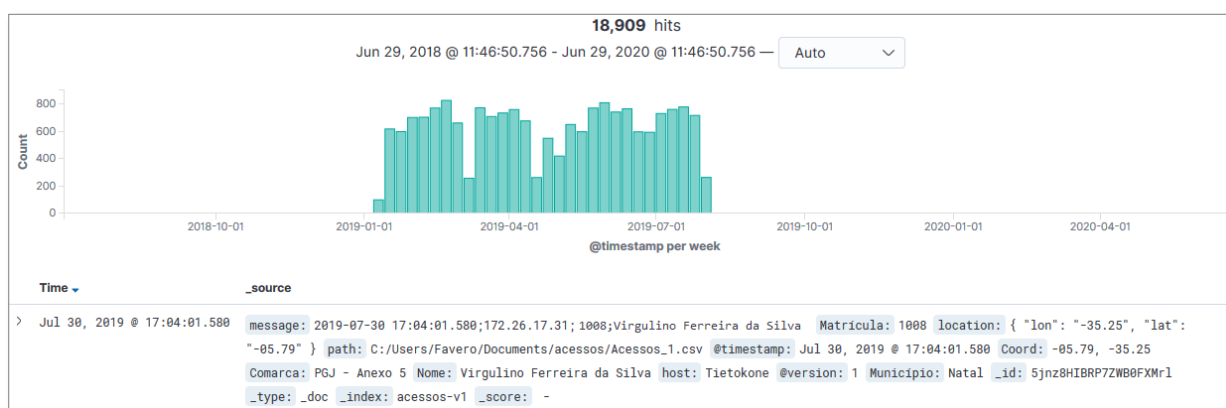


Figura 50 - Documentos de "acessos-v1" descobertos.

3.8 CRIAÇÃO DE VISUALIZAÇÕES NO KIBANA

No *Kibana* existe o menu de visualizações (*Visualize*) que é o local para criação de forma individual de cada elemento que vai ser utilizado nos *dashboards*. O *Kibana* oferece diversas categorias diferentes de visualização que devem ser escolhidas previamente de acordo com a necessidade do usuário, como gráfico de setores, gráfico de barras, mapa, tabelas, etc.

As visualizações que representam o título da aplicação e logo do MPRN são utilizadas em vários painéis diferentes. Elas são criadas a partir da linguagem de marcação, *Markdown*, que é um tipo de linguagem que permite a criação de mensagens de texto e a inclusão de imagens da internet através de uma *url*. No *Kibana*, esse tipo de visualização é chamado de *Markdown*. A linguagem geradora e as visualizações resultandas estão representadas nas Figuras 51 e 52.

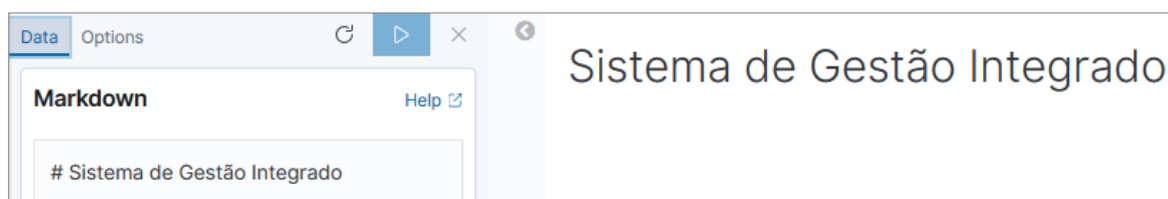


Figura 51 - Visualização do título da aplicação.

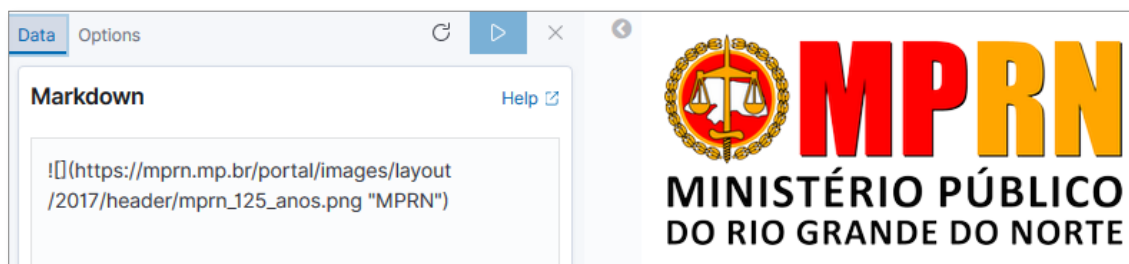


Figura 52 - Visualização do logo da instituição.

Nas seções a seguir, são apresentadas as visualizações criadas para cada painel.

3.8.1 Painel de Acessos

Nesse painel, que recebe dados do *index* "acessos-v1", as visualizações utilizadas são:

- Título da aplicação e logo do MPRN (ver Figuras 51 e 52);
- Visualização de tipo métrica (*Metric*) com soma do total de acessos (ver Figura 53);



Figura 53 - Visualização da métrica de total de acessos.

- Duas visualizações de tabela de dados (*Data Table*) com acessos por nome e matrícula (Figura 54) e por município (Figura 55);

Nome	Matrícula	Acessos
Gabriel Garcia Marques	1004	459
Hana Morenos	1112	424
Lúcio de Mendonça	1010	370
Joel Natalino Santana	1113	354
Satoshi Batista	1044	339
Juan Gabriel Vasquez	1032	280
Marta Vieira da Silva	1007	274
Sônia Maria Campos Braga	1047	274
Francisco Cândido Xavier	1005	272
Jonas Kahnwald	1103	262

Export: [Raw](#) [Formatted](#)

1 2 3 4 5 ... 34 »

Figura 54 - Visualização da tabela de acessos por nome e matrícula.

Município	Acessos
Natal	12.741
Mossoró	1.736
Parnamirim	554
São Gonçalo do Amarante	318
Caicó	307
Ceará Mirim	294
Pau dos Ferros	247
Macau	246
Nova Cruz	204
Monte Alegre	202

Export: [Raw](#) [Formatted](#)

1 2 3 4 5 ... 6 »

Figura 55 - Visualização da tabela de acessos por município.

- Gráfico de setores pertencente à categoria “Pie” com um top 10 de comarcas com mais acessos (Figura 56);

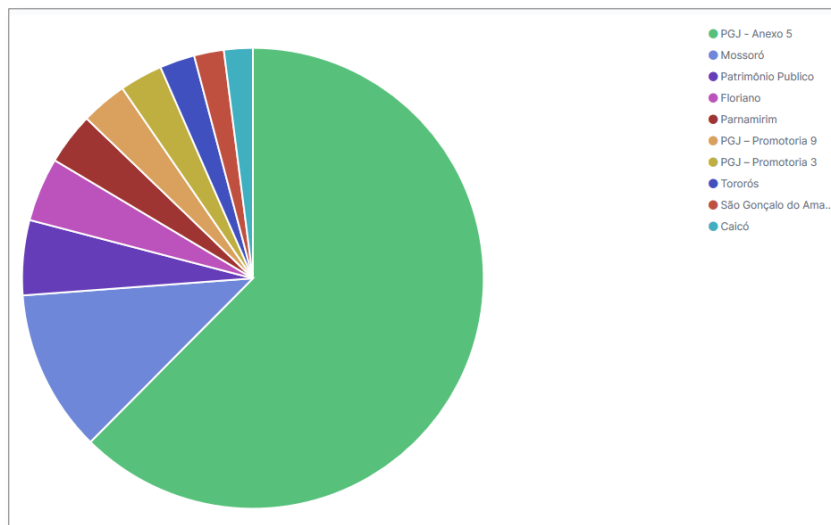


Figura 56 - Visualização do gráfico de top 10 de comarcas.

- Mapa do estado do RN criado na categoria "Coordinate Map" (Figura 57).

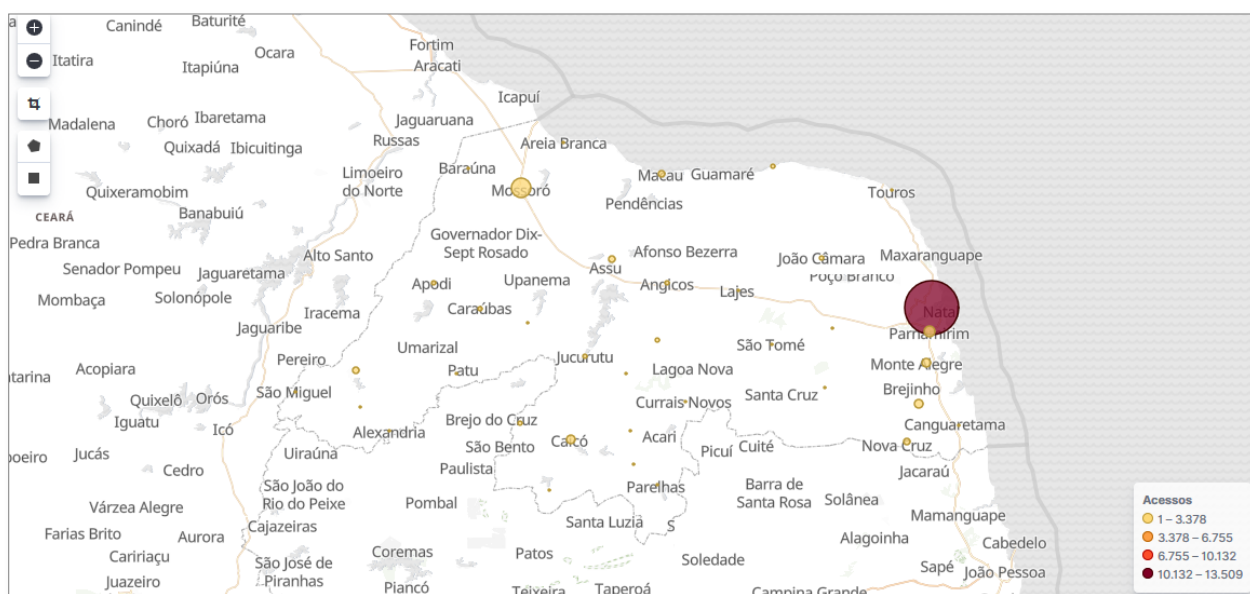


Figura 57 - Visualização do mapa de acessos no RN.

3.8.2 Painel de Alertas do Sistema

Esse painel utiliza o *index* "alertas" e suas visualizações são as seguintes:

- Logo do MPRN (Figura 51);
- Título do painel criado com a visualização de tipo *Markdown* (Figura 58);

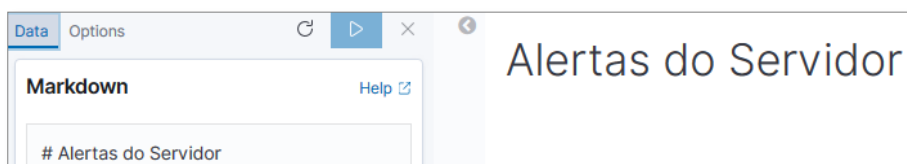


Figura 58 - Visualização do título "Alertas do Servidor".

- Visualização de tipo métrica (*Metric*) com soma do total de alertas (Figura 59);



Figura 59 - Visualização da métrica de total de alertas

- Duas visualizações de tipo tabela de dados (*Data Table*) com tabelas de categorias de alertas (Figura 60) e mensagens (Figura 61);

Categoria	Total
Exceção de Persistência (Erro para Obter Conexão JDBC)	16.892
Memory leak (Falha em encerrar thread)	1.118
Argumento de linha de comando	701
Não pode extrair ResultSet (SQL)	644
Falha Para Remover Threads	453

Export: Raw Formatted

1 2 3 4 5 ... 16 »

Figura 60 - Visualização da tabela de total de alertas por categoria.

Alerta	Mensagem	Total
SEVERE	mprm.perdigueiro.view.controle.util.log.LogControlUtil.escreveLog org.hibernate.exception.JDBCConnectionException: Unable to acquire JDBC Connection	15.474
SEVERE	mprm.perdigueiro.view.controle.util.log.LogControlUtil.escreveLog javax.persistence.PersistenceException: org.hibernate.exception.JDBCConnectionException: Unable to acquire JDBC Connection	1.373
SEVERE	mprm.perdigueiro.view.controle.util.log.LogControlUtil.escreveLog org.hibernate.exception.SQLGrammarException: could not extract ResultSet	461
SEVERE	mprm.perdigueiro.view.controle.util.log.LogControlUtil.escreveLog null	195
SEVERE	mprm.perdigueiro.view.controle.util.log.LogControlUtil.escreveLog javax.persistence.PersistenceException: org.hibernate.exception.SQLGrammarException: could not extract ResultSet	182

Export: [Raw](#) [Formatted](#)

1 2 3 4 5 ... 70 »

Figura 61 - Visualização da tabela de mensagens.

- Gráfico de setores pertencente a categoria *Pie* com tipos de alertas (Figura 62);

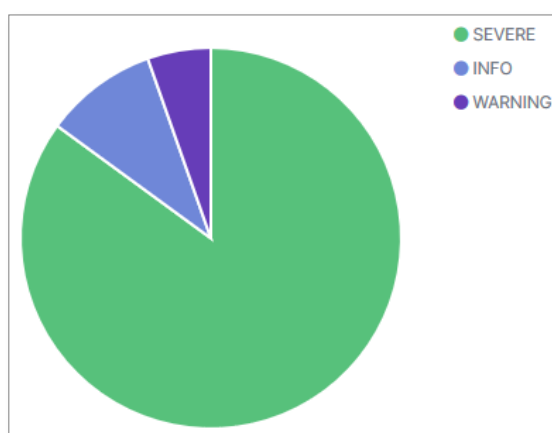


Figura 62 - Visualização do gráfico de tipos de alertas.

3.8.3 Painel de Erros de Acesso de Usuários

Esse painel também é baseado no *index* "alertas" e possui as seguintes visualizações:

- Título da aplicação e logo do MPRN (Figuras 51 e 52);
- Título do painel criado com a visualização de tipo *Markdown* (ver Figura 63);

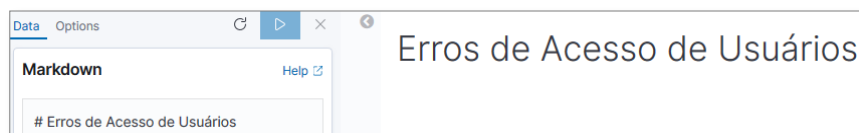


Figura 63 - Visualização do título "Erros de Acesso de Usuários".

- Visualização de tipo métrica (*Metric*) com soma do total de erros (Figura 64);

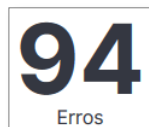


Figura 64 - Visualização da métrica de total de erros de acesso de usuários.

- Uma visualização do tipo tabela de dados (*Data Table*) mostrando nome e matrícula de usuários (Figura 65);

Usuário	Matrícula	Total
Mads Dittman Mikkelsen	1095	38
Desconhecido	Inválida	15
Emerson Antônio Dias Silva	1260	11
Jonas Kahnwald	1103	3
Michael Paul Chan	1051	3

Export: [Raw](#) [Formatted](#)

1 2 3 4 5 »

Figura 65 - Visualização da tabela de usuários com erros.

- Gráfico de setores (categoria *Pie*) com *top 10* de usuários (Figura 66);

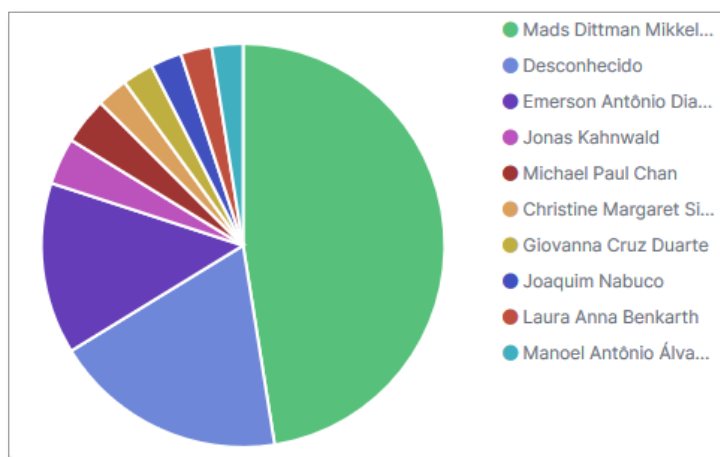


Figura 66 - Visualização do gráfico *top 10* de usuários com erros.

- Gráfico de barras do tipo *horizontal bar* com tipos de erros (Figura 67).

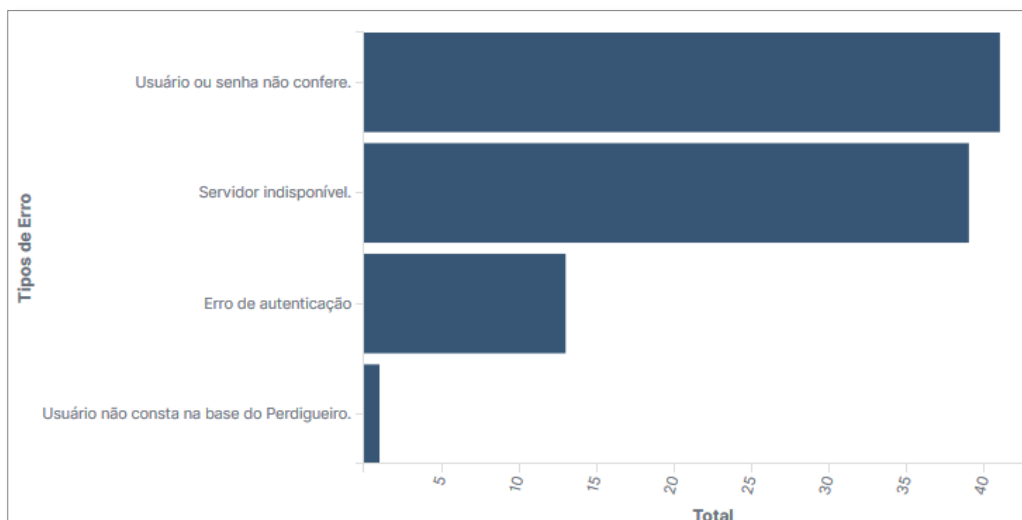


Figura 67 - Visualização do gráfico de categorias de erros de acesso de usuários.

3.8.4 Painel de Requisições

Esse painel utiliza o *index "request"* e tem as seguintes visualizações:

- Título da aplicação e logo do MPRN (Figuras 51 e 52);
- Título do painel criado com a visualização de tipo *Markdown* (Figura 68);

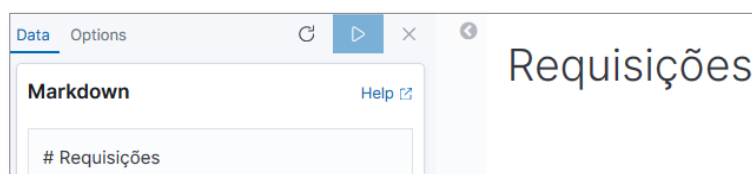


Figura 68 - Visualização do título "Requisições"

- Duas Visualizações de tipo métrica (*Metric*) com soma de *gigabytes* enviados (Figura 70) e total de requisições (Figura 71). É utilizada uma fórmula, na opção "*json input*" para converter de *bytes* para *gigabytes* em uma das métricas (ver Figura 69);

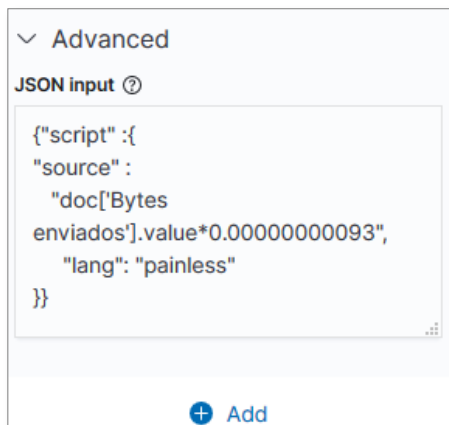


Figura 69 - Configuração convertendo de *bytes* para *gigabytes*.

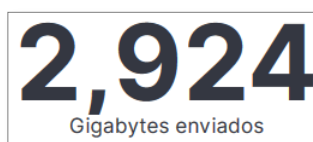


Figura 70 - Visualização de métrica da soma de *gigabytes* enviados.

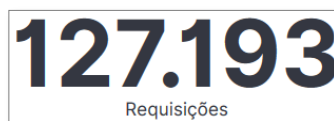


Figura 71 - Visualização da métrica de total de requisições.

- Duas visualizações do tipo tabela de dados (*Data Table*) com tabelas de requisições por *IP* (Figura 72) e por objeto (Figura 73);

Ip	Requisições
192.168.116.162	23.419
192.168.104.151	11.048
172.26.9.28	6.113
172.26.3.58	5.314
172.26.8.18	5.151

Export: [Raw](#) [Formatted](#)

1 2 3 4 5 ... 18 »

Figura 72 - Visualização da tabela de requisições por *IP*.

Objeto	Requisições
/Perdigueiro/ajax/login/controle/controleChecaBloqueio.jsp	26.573
/Perdigueiro/logout.jsp	26.559
/Perdigueiro/ajax/login/controle/controleTempoSessao.jsp	25.303
/Perdigueiro/index.jsp	23.574
/Perdigueiro/js/plugin/tinymce/skins/lightgray/content.min.css	1.149
/Perdigueiro/ajax/grafico/controle/ControleGestaoGraficoVinculo.jsp	959
/Perdigueiro/menu.jsp	890
/Perdigueiro/ajax/guia/controle/controleNumProcedimentoAberto.jsp	748
/Perdigueiro/img/oloader/loader.gif	561
/Perdigueiro/ajax/grafico/controle/ControleGraficoVinculoV.jsp	560

Export: [Raw](#) [Formatted](#)

1 2 3 4 5 ... 119 »

Figura 73 - Visualização da tabela de requisições por objeto.

- Dois gráficos de setores pertencentes à categoria *Pie* com tipos de requisições (Figura 74) e comarcas e municípios (Figura 75);

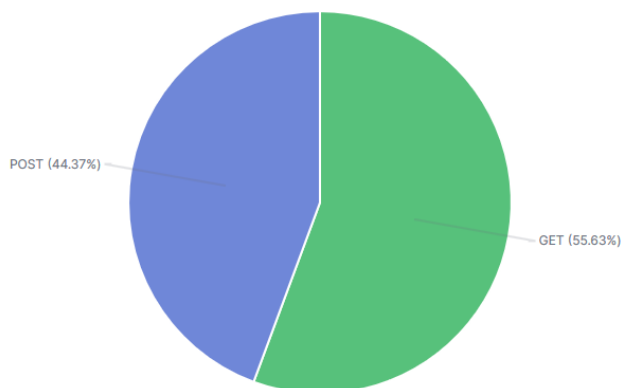


Figura 74 - Visualização do gráfico de tipos de requisição.

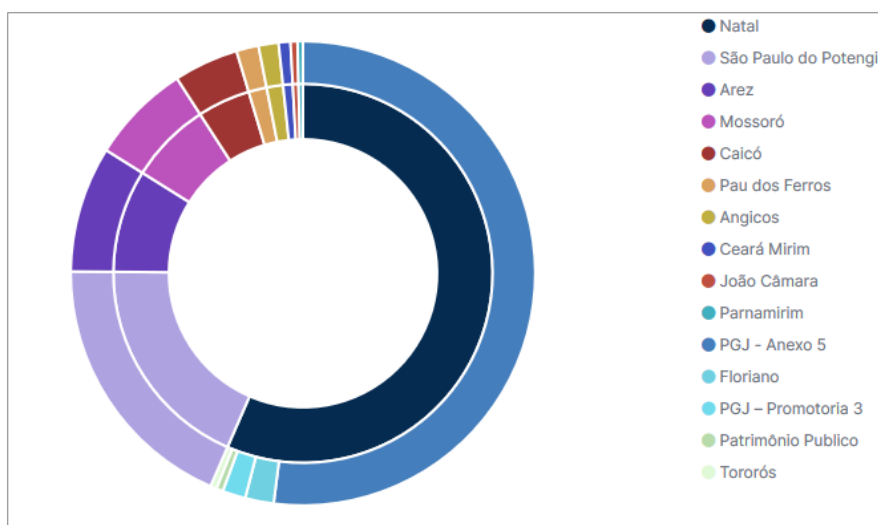


Figura 75 - Visualização do gráfico de comarcas e municípios.

3.8.5 Painel de Métricas

Esse painel utiliza o *index* "jmx-v1" e utiliza as seguintes visualizações:

- Duas visualizações do tipo *Markdown* para título do painel (Figura 76) e logo do *Apache Tomcat* (Figura 77);

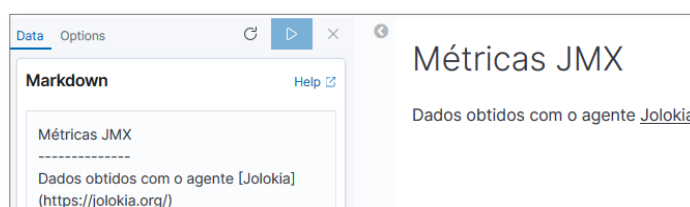


Figura 76 - Visualização do título "Métricas JMX".

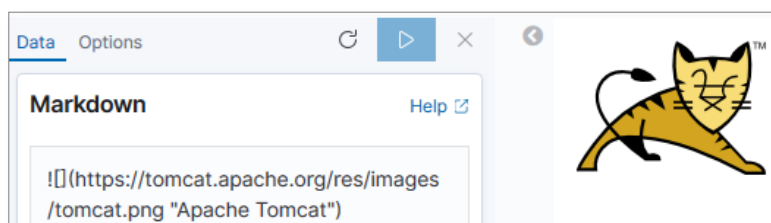


Figura 77 - Visualização do logo do *Apache Tomcat*.

- Três visualizações de tipo métrica (*Metric*) para mostrar o *uptime* (Figura 79), total de eventos (Figura 80) e nome e versão do servidor (Figura 81). É utilizada uma fórmula, na opção "*json input*" para converter de milissegundos em horas na métrica de *uptime* (Figura 78);

```
JSON input ?  
{  
  "script" : {  
    "source" :  
    "doc['jolokia.metrics.uptime'].value*2.  
    7778e-7",  
    "lang": "painless"}}  
}
```

Figura 78 - Configuração convertendo de ms para h.

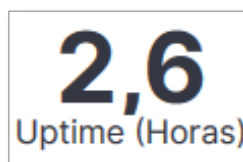


Figura 79 - Visualização da métrica de *uptime* em horas.



Figura 80 - Visualização da métrica do total de eventos.



Figura 81 - Visualização da métrica de nome e versão do servidor.

- Dois gráficos de linha com visualizações do tipo "*line*" para representar as variações de memória *heap* (Figura 82) e *non heap* (Figura 83).

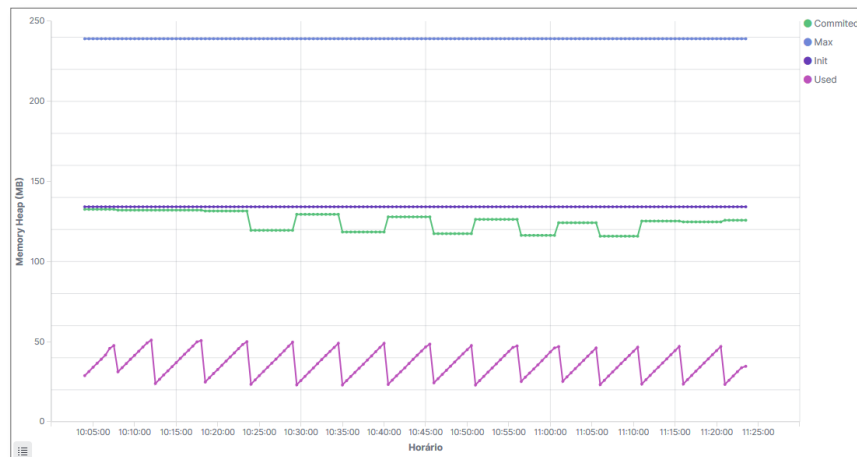


Figura 82 - Visualização do gráfico da memória *heap*.

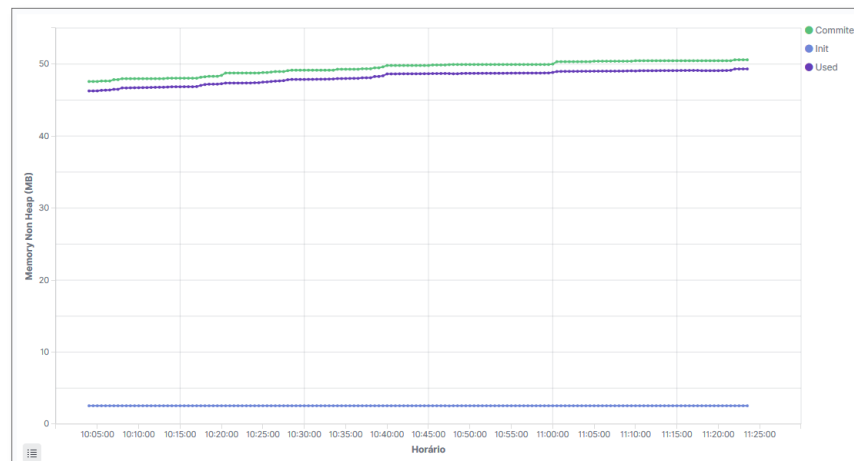


Figura 83 - Visualização do gráfico da memória *non heap*.

- Um gráfico do tipo "Area" que representa a variação na carga de *cpu* (Figura 84).

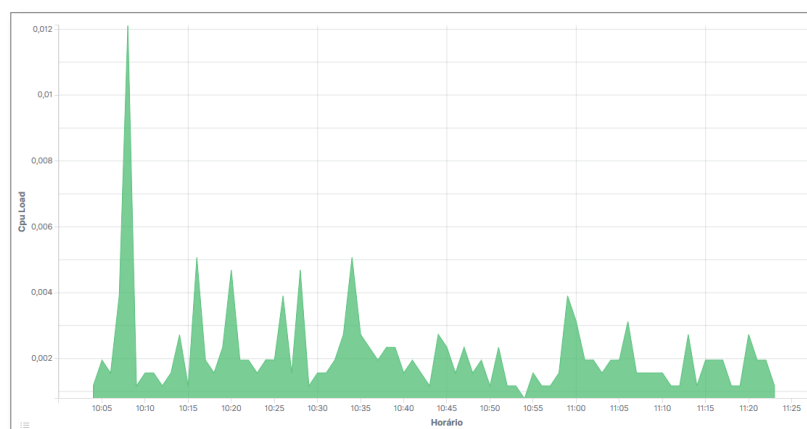


Figura 84 - Visualização do gráfico de carga da *cpu* (CPU Load).

3.9 CRIAÇÃO DOS PAINÉIS DE VISUALIZAÇÃO NO KIBANA

Os painéis de visualização, também chamados *dashboards*, são criados no *Kibana* na seção "*Dashboards*", eles são telas que reúnem várias visualizações.

Para esse trabalho são criados cinco painéis diferentes, onde cada um deles reúne as visualizações associadas a uma categoria de dado. Os painéis de acessos (Figura 85), alertas (Figura 86), erros de acesso de usuários (Figura 87), requisições (Figura 88) e métricas *JMX* (Figura 89) estão ilustrados abaixo.

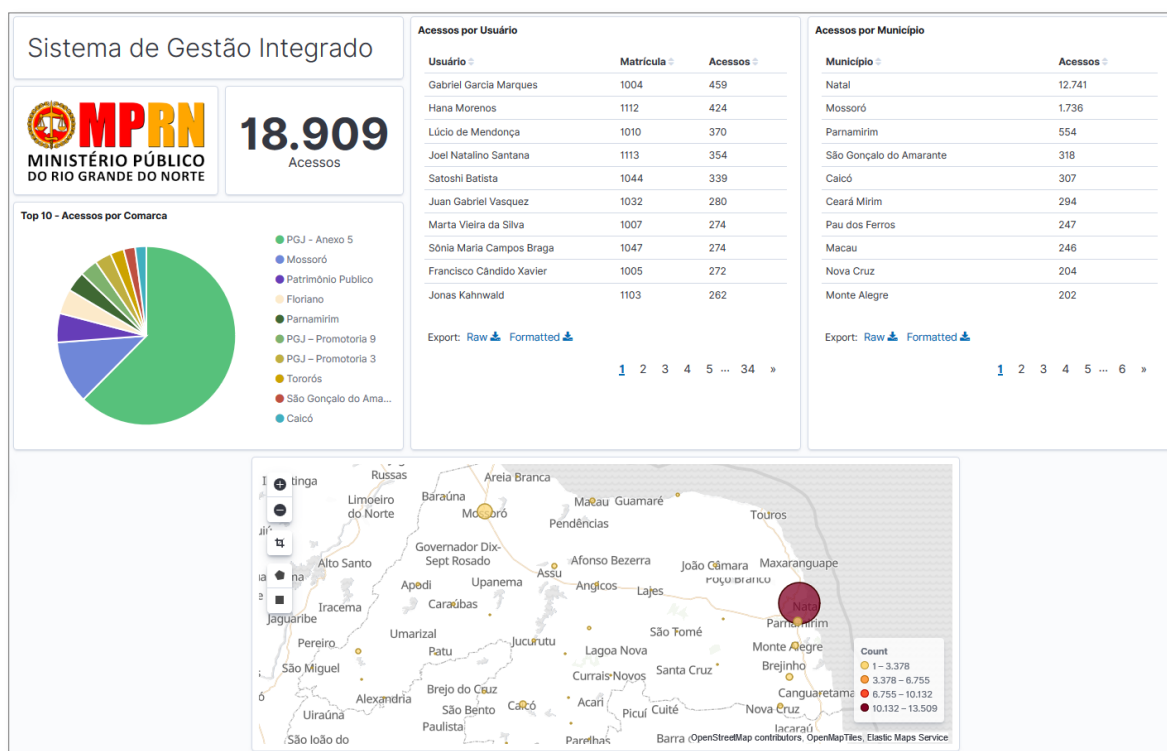


Figura 85 - Painel de acessos.



Figura 86 - Painel de alertas.

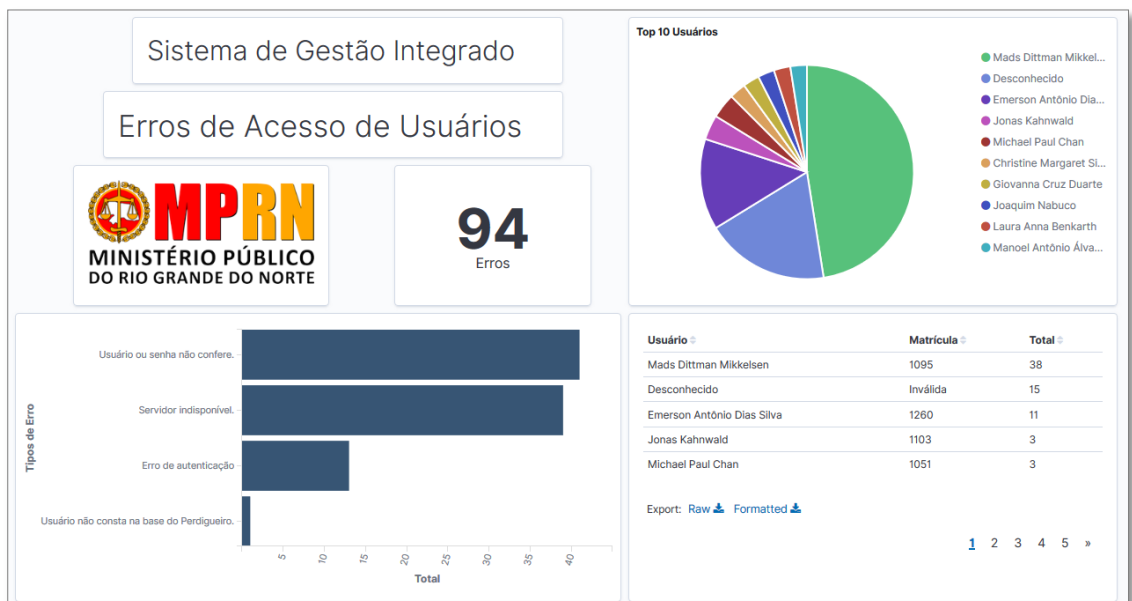


Figura 87 - Painel de erros de acesso de usuários.



Figura 88 - Painel de requisições.

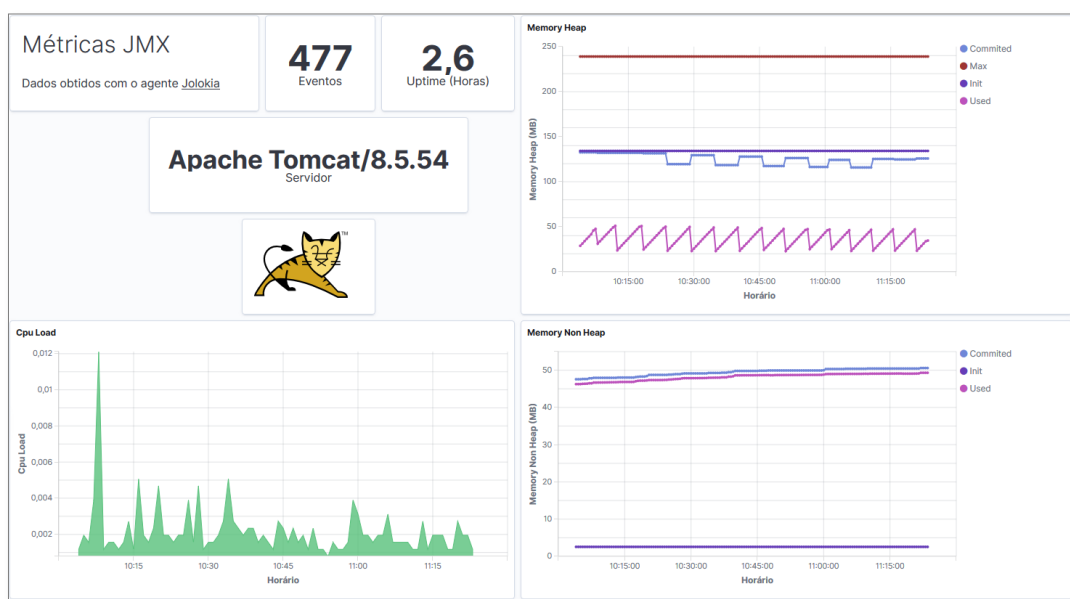


Figura 89 - Painel de métricas JMX.

Com os painéis criados, é encerrada a construção do projeto, na próxima seção é detalhada a utilização dos painéis e de ferramentas auxiliares para realizar o monitoramento dos dados de log.

4 UTILIZAÇÃO DOS PAINÉIS E DE FERRAMENTAS AUXILIARES

Com os *painéis* criados, o projeto já está pronto para realizar o seu objetivo central de prover uma análise de dados de forma simples. Essa análise além de ser feita com a visualização dos painéis criados, pode contar com o apoio de algumas ferramentas do *Kibana*, que estão representadas nas seções seguintes, como filtros de texto ou tempo, descoberta de dados, console, interação com os gráficos e tabelas, consultas de texto além de exportação de dados.

4.1 DESCOBERTA DE DADOS

A descoberta de dados é a forma mais simples de se consultar qualquer informação, ela é feita na seção "*Discover*" que foi abordada na seção 3.7. Essa ferramenta permite a visualização de todos os documentos presentes no *index* selecionado com a opção de filtrar os dados por data, por campo, texto e ainda ver uma linha do tempo baseada no campo de data dos documentos.

4.2 CONSOLE

Através do console, no menu "*Dev Tools*", é possível realizar requisições com padrão *REST* tanto para consulta quanto manipulação de dados. Essas operações não são muito intuitivas e o uso dessa ferramenta é recomendado apenas para usuários familiarizados com requisições desse tipo. O console foi utilizado na seção 3.6 para a criação de um *template*. Abaixo será mostrado um exemplo de pesquisa por documentos que possuem certo valor em um campo (Figura 90).

```
GET alertas/_search
{
  "query": {
    "match": {
      "Aviso" : "WARNING"
    }
  }
}
```

Figura 90 - Pesquisa pelo console.

A busca encontrou diversos resultados, na Figura 91 são exibidas informações relacionadas a busca enquanto que na Figura 92 é visto um dos documentos encontrados pela busca.

```
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2086,
      "relation" : "eq"
    },
    "max_score" : 2.3453205,
  }
}
```

Figura 91 - Estatísticas da busca feita no console.

```
"hits" : [
  {
    "_index" : "alertas",
    "_type" : "_doc",
    "_id" : "NNDuOHMBGuVZdYpMH3kf",
    "_score" : 2.3453205,
    "_source" : {
      "host" : "Tietokone",
      "Aviso" : "INFO",
      "@version" : "1",
      "path" : "C:/Users/Favero/Documents/alertas/catalina_prod_1.out",
      "Thread" : "Thread-3",
      "Msg" : ""org.apache.coyote.AbstractProtocol.pause Pausing ProtocolHandler [\"https-jsse-nio-8443\"]"",
      "message" : ""18-Feb-2019 10:58:35.211 INFO [Thread-3] org.apache.coyote.AbstractProtocol.pause Pausing ProtocolHandler [\"https-jsse-nio-8443\"]"",
      "@timestamp" : "2019-02-18T13:58:35.211Z",
      "Categoria" : "Pausando protocolo"
    }
  }
],
```

Figura 92 - Exemplo de documento encontrado na busca do console.

4.3 FILTROS

Tanto nos *dashboards* quanto na seção de descoberta de dados é possível filtrar os resultados a partir de seus campos de data (filtro temporal) ou pelo conteúdo de texto de campos (filtro de texto).

No caso do filtro de tempo é possível ver os resultados dentro de um intervalo relativo (categoria *Relative*) como dois anos atrás ou com o uso de datas e horários exatos como entre dias 7 e 27 de janeiro de 2019 (categoria *Absolute*) além de poder-se usar o momento atual (categoria *Now*) que sempre se atualiza de acordo com o horário e a data registrados no navegador.

Esse filtro encontra-se sempre na parte superior direita da tela. A seguir, na Figura 93, é exibido um filtro que representa os dados de março de 2019.

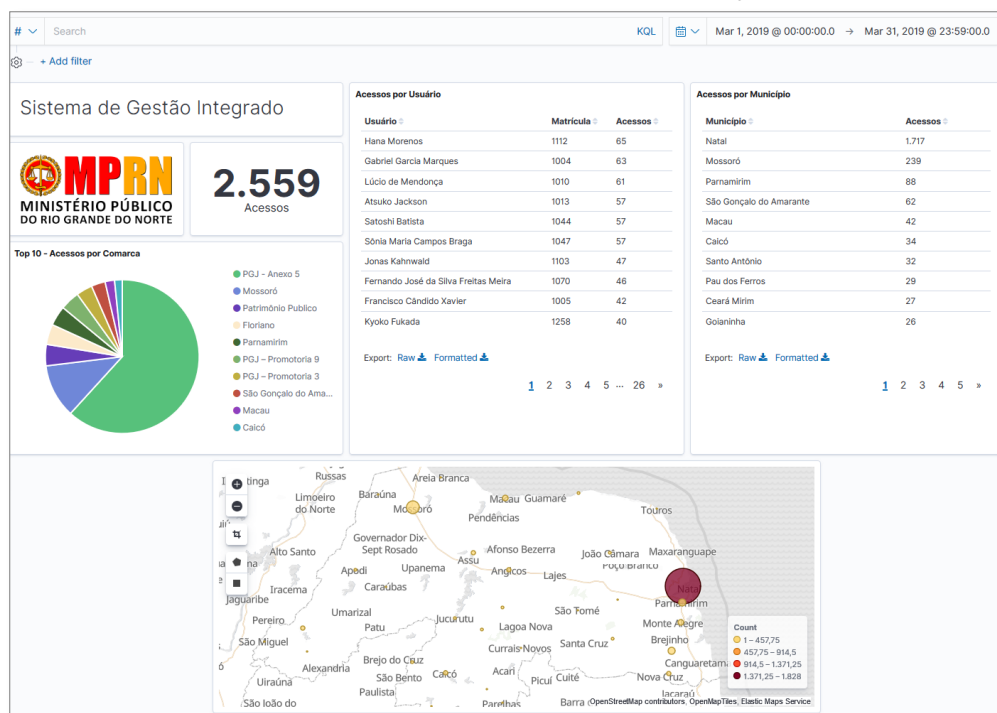


Figura 93 - Painel de acessos de março de 2019.

Com o filtro de texto é possível exibir apenas os resultados que possuem certa característica relacionada a seus campos, como mostrar resultados onde o município seja Natal, além disso é possível fazer combinações lógicas com os operadores "é" (*is*), "não é" (*is not*), "é um de" (*is one of*), "não é um de" (*is not one of*), "existe" (*exists*) e "não existe" (*does not exist*) para combinar diversas características ao mesmo tempo, como por exemplo, dados de acessos que não são dos municípios de Natal ou Parnamirim (utilizando a expressão "*is not one of*").

O filtro de texto pode ser fixado na parte superior da tela para ser reutilizado em outras ferramentas do Kibana, como a de descoberta (*Discover*). O filtro de texto se encontra na parte superior esquerda da tela abaixo da caixa de texto para consulta.

Na Figura 94 pode-se ver um filtro de texto que inclui apenas os acessos que não pertencem aos municípios de Natal ou Parnamirim.

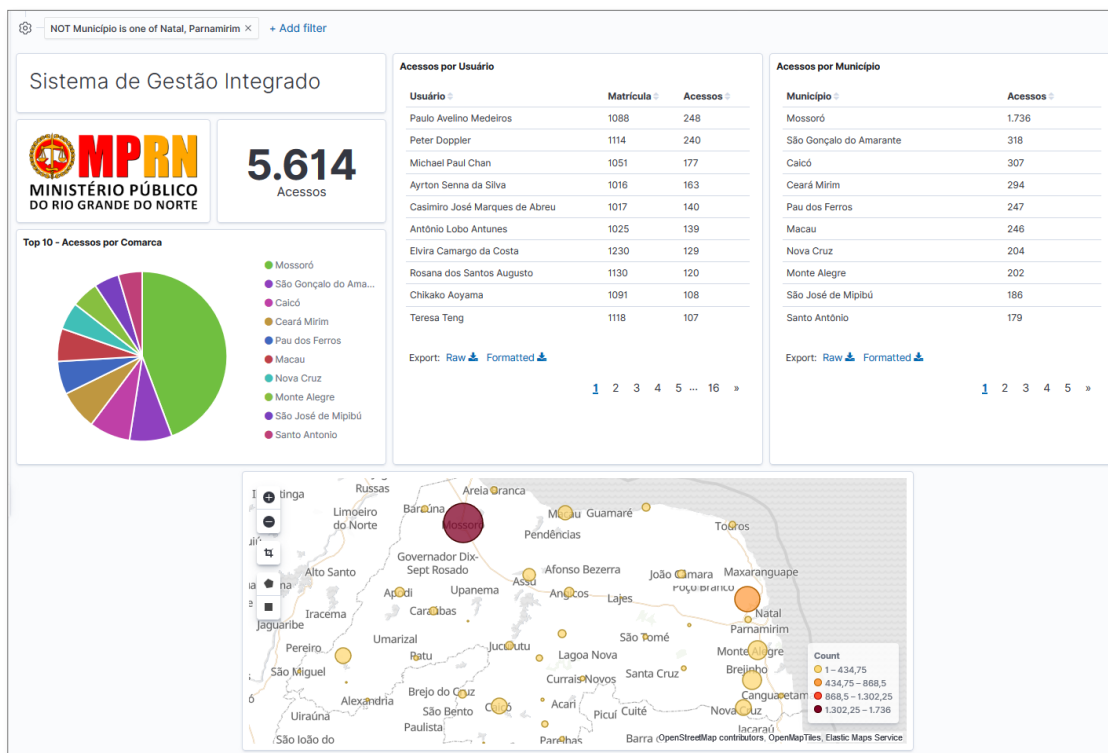


Figura 94 - Painel com acessos que não são de Natal ou Parnamirim.

4.4 INTERAÇÃO COM AS VISUALIZAÇÕES

Outra forma de filtrar os dados exibidos se dá pela interação direta com as visualizações ao passar o cursor sobre alguns elementos pertencentes a elas. Por exemplo, ao clicar em qualquer elemento de uma tabela é possível filtrar os dados associados a aquele elemento, como quando se clica no nome de um usuário, serão exibidos todos os dados de documentos que tenham aquele nome de usuário associado.

Para ilustrar esse caso é mostrado, na Figura 95, um usuário selecionado e depois, na figura 96, o painel filtrado para exibir apenas os acessos desse usuário.

Acessos por Usuário

Usuário	Matrícula	Acessos
Gabriel Garcia Marques	1004	459

Figura 95 - Seleção de nome de usuário.

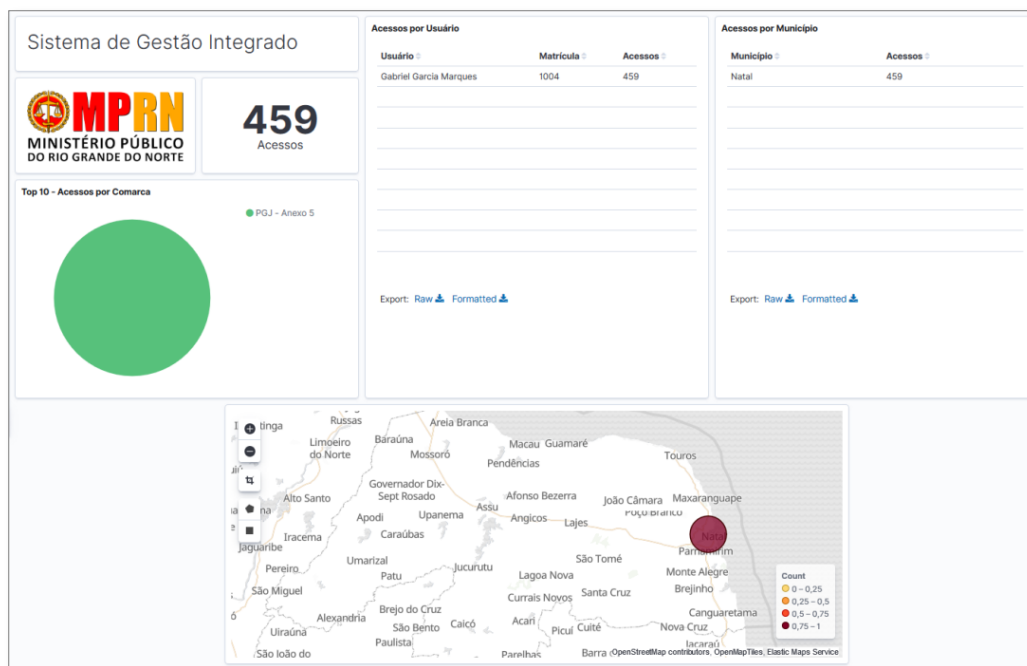


Figura 96 - Painel de acessos feitos apenas pelo usuário selecionado.

É utilizado um filtro inclusivo nesse caso, mas ao clicar no elemento, também é apresentada a opção de filtro excludente.

Esse método cria um filtro que funciona da mesma maneira que o filtro textual visto na seção anterior. No caso de gráficos, pode-se clicar em uma categoria da legenda para criar o filtro ou pode-se clicar em seções do próprio gráfico, com o detalhe que clicando em seções do gráfico só há a opção de filtro inclusivo.

É importante lembrar que mapas de coordenadas não apresentam esse tipo de interação.

4.5 CONSULTAS

Na parte superior esquerda da tela, ao lado da barra do filtro temporal existe a barra de consulta, nela é possível visualizar os dados com base no conteúdo de texto encontrado em algum dos campos dos documentos. Sua função é similar à do filtro de texto.

Para realizar consultas é preciso digitar na barra de consultas comandos na linguagem chamada de *KQL* (*Kibana Query Language*) que é própria do *Kibana* e tenta ser uma versão mais simples de outras linguagens de consulta como *SQL*,

caso seja necessário, também existe a opção de utilizar a linguagem *Lucene*, para isso basta clicar em "*KQL*" e escolher a opção de desativar.

A linguagem *KQL* é bem simples mas não é necessário conhecê-la para realizar uma consulta, pois ao digitar na barra de consultas surge uma lista de operações disponíveis que podem ser selecionadas.

Os operadores disponíveis são o "igual" (*equals* ":") para pesquisar se existe algum termo específico no campo, "existe" (*exists* ".*") para mostrar documentos onde certo campo existe, "e" (*and*) que combina duas condições e "ou" (*or*) que alterna duas condições.

Para exemplificar, no painel de acessos, é feita um consulta por todos os acessos do município de Mossoró e que sejam de usuários com o sobrenome Silva. A consulta é mostrada na Figura 97 e o resultado no painel da Figura 98.



Figura 97 - Consulta por município e nome.

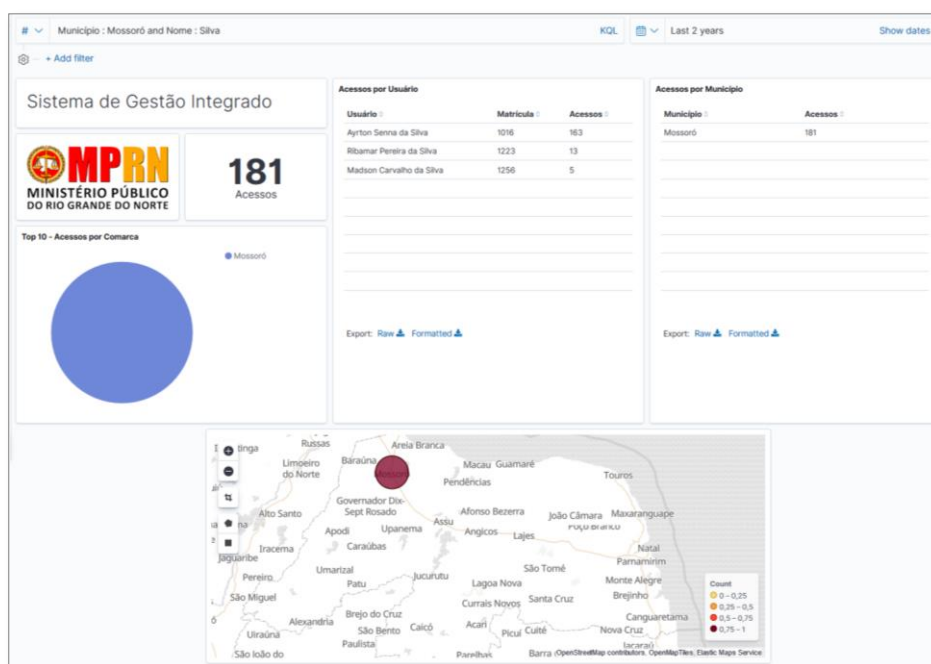


Figura 98 - Painel de acessos de Mossoró com usuários de sobrenome Silva.

As consultas podem ser salvas para uso posterior, além de que é possível salvar filtros junto da consulta.

4.6 EXPORTAÇÃO E COMPARTILHAMENTO DE DADOS

Existem algumas ferramentas no *Kibana* responsáveis por compartilhamento dos dados e elementos visualizados. Os principais elementos que podem ser exportados são dados presentes em visualizações, elementos completos e dados descobertos com a ferramenta "*Discover*".

É possível exportar os dados de uma visualização através da opção de inspeção (*Inspect*), vista logo abaixo na Figura 99, que exibe uma tabela com os dados presentes e permite a exportação desses dados em formato "csv" através da opção "*Download CSV*" que pode ser tanto formatado (*formatted*) como bruto (*raw*).

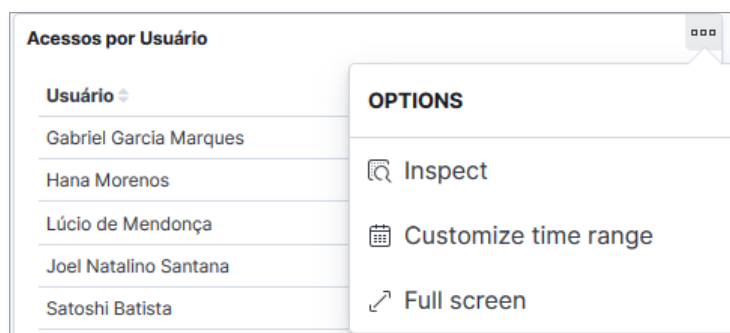


Figura 99 - Opções de visualização.

A janela expandida da função *Inspect* é exibida a seguir, na Figura 100, enquanto que na Figura 101 é exibido um trecho do arquivo csv exportado.

Usuário	Matricula	Acessos
Gabriel Garcia Marques	1004	459
Hana Morenos	1112	424
Lúcio de Mendonça	1010	370
Joel Natalino Santana	1113	354
Satoshi Batista	1044	339
Juan Gabriel Vasquez	1032	280
Marta Vieira da Silva	1007	274
Sônia Maria Campos Braga	1047	274
Francisco Cândido Xavier	1005	272
Jonas Kahnwald	1103	262

Figura 100 - Inspeccionando a visualização.

```
"Usuário","Matrícula",Acessos
"Gabriel Garcia Marques",1004,459
"Hana Morenos",1112,424
```

Figura 101 - Trecho da tabela exportada em formato csv.

Toda visualização com exceção daquelas que não exibem dados do *index*, como o tipo *Markdown* (que exibem apenas um título ou imagem ilustrativa), podem ter seus dados visualizados dessa maneira, em forma de tabela, com a opção de exportação. Caso algum filtro ou consulta esteja ativo, os dados exportados também serão filtrados. Os dados exportados são aqueles exibidos no momento.

Dados exibidos na seção *Discover* também podem ser exportados em formato ".csv", para isso é preciso primeiro salvar a consulta atual com a opção "Save" e depois utilizar a ferramenta de compartilhar "Share", vista na Figura 102, que permite gerar o arquivo csv (*CSV Reports*) ou gerar um link de acesso a essa busca (*Permalink*) que pode ser acessado por usuários que tiverem acesso a mesma instância do Kibana. Um trecho do arquivo csv gerado é exibido na Figura 103.

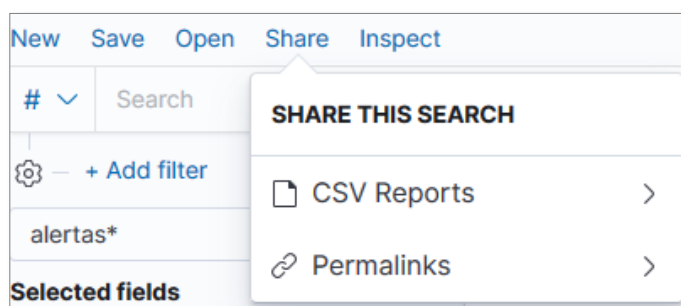


Figura 102 - Compartilhando busca.

```
"@timestamp","@version",Aviso,Categoria,Msg,Thread,"_id","_index","_score","_type",host,message,path
"Feb 18, 2019 @ 13:58:01.589",1,SEVERE,"Compilar classe para JSP","org.apache.catalina.core.StandardW
"Feb 18, 2019 @ 13:58:35.211",1,INFO,"Pausando protocolo","org.apache.coyote.AbstractProtocol.pause E
```

Figura 103 - Trecho do arquivo csv de busca salvo.

Além disso, o Kibana disponibiliza em seu menu de gerenciamento (*Management*), na seção de objetos salvos (*Saved Objects*), vista na Figura 104 abaixo, a opção de tanto editar qualquer elemento criado como de importar e exportar elementos. Assim, é possível compartilhar objetos no formato *ndjson* que é um arquivo onde cada linha representa um documento *json* separado, por isso

vários objetos podem ser exportados juntos em um único documento. Além do compartilhamento, com essa ferramenta é possível realizar o backup de todos os elementos criados.

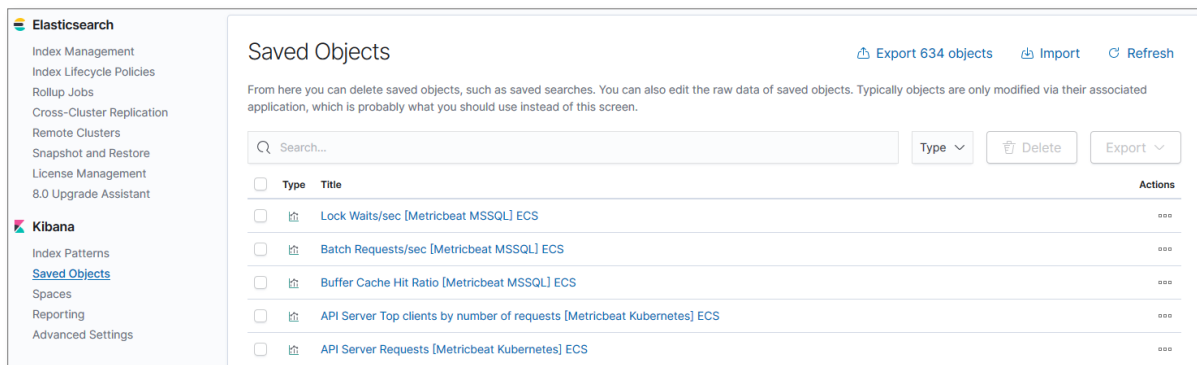


Figura 104 - Lista de objetos salvos.

A próxima seção trata das considerações finais sobre tudo que foi construído nesse trabalho.

5 CONSIDERAÇÕES FINAIS

A partir da demanda do MPRN de monitorar uma aplicação de gerenciamento de usuários foi criado este projeto de painéis de visualização interativos, baseados nas ferramentas pertencentes à *Elastic Stack*, que monitoram dados provenientes de arquivos de *log* gerados pela aplicação de Sistema de Gestão Integrada do MPRN.

Neste trabalho foram criados seis painéis de visualização interativos que monitoram respectivamente dados de acessos de usuários, erros e alertas da aplicação, erros de acesso de usuários, requisições feitas à aplicação e métricas do servidor que hospeda a aplicação.

Esse projeto conseguiu cumprir o objetivo de analisar todos os dados de *log* relevantes a partir dos arquivos de texto fornecidos. Como limitação, tem-se o fato dele ter sido implantado em um ambiente de testes com todas as ferramentas executadas em uma única máquina. Como melhoria, recomenda-se que o sistema seja implementado em um ambiente de computadores em rede onde cada ferramenta seja executada em uma máquina diferente, assim mais recursos computacionais podem ser voltados para cada componente além de que, no caso de alguma máquina falhar, parte do sistema pode continuar funcionando. Além disso, todo o sistema criado pode ser hospedado em nuvem através de serviços pagos como o *Elastic Cloud*.

REFERÊNCIAS

AGÊNCIA BRASIL. **Publicado decreto que lança a Estratégia de Governo Digital 2020-2022**, 29 de abr. 2020. Disponível em <<https://agenciabrasil.ebc.com.br/economia/noticia/2020-09/governo-atinge-900-servicos-digitalizados-em-20-meses>>. Acesso em: 09 set. 2020.

AGÊNCIA BRASIL. **Governo atinge 900 serviços digitalizados em 20 meses**, 06 de set. 2020. Disponível em <<https://agenciabrasil.ebc.com.br/economia/noticia/2020-09/governo-atinge-900-servicos-digitalizados-em-20-meses>>. Acesso em: 09 set. 2020.

APACHE Tomcat. Versão 8. **The Apache Software Foundation**, 2020. Disponível em: <www.tomcat.apache.org/download-80.cgi>. Acesso em : 04 mai. 2020.

BAELDUNG. **Introduction to Java Servlets**, 02 de nov. 2018. Disponível em <<https://www.baeldung.com/intro-to-servlets> >. Acesso em: 14 dez. 2020.

BEATS. **Elasticsearch B.V.**, 2020. Beats: Agentes de dados lightweight. Disponível em <<https://www.elastic.co/pt/beats/> >. Acesso em: 26 mar. 2020.

DIGIX. **O que é o gerenciamento eletrônico de objetos (GED)?**, 02 de jan. 2020. Disponível em <<https://www.digix.com.br/o-que-e-o-gerenciamento-eletronico-de-documentos-ged/> >. Acesso em: 07 dez. 2020.

ELASTIC. **Elasticsearch B.V.**, 2020. O que é o ELK Stack ? Por que se chama Elastic Stack? Disponível em <<https://www.elastic.co/pt/elk-stack>>. Acesso em: 26 mar. 2020.

ELASTICSEARCH 7. Versão 7.4.2. **Elasticsearch B.V.**, 2019. Disponível em: <<https://www.elastic.co/downloads/past-releases/elasticsearch-7-4-2>>. Acesso em: 26 mar. 2020.

GORMLEY, Clinton; TONG, Zachary. **Elasticsearch: The Definitive Guide: A distributed Real-Time Search Engine**. 1 ed. EUA: O'Reilly Media, 2015.

HUSS, Roland. Jolokia - JMX on Capsaicin. Overview. **Jolokia**, 2019. Disponível em < <https://jolokia.org/features/overview.html> > . Acesso em : 04 mai. 2020.

JOLOKIA. Versão 1.6.2. **Huss**, Roland, 2019. Disponível em: <<https://jolokia.org/download.html>> . Acesso em : 04 mai. 2020.

JVM Memory Structure. **YourKit**, 2020. Java Profiler Knowledge Base. Disponível em <<https://www.yourkit.com/docs/kb/sizes.jsp>> Acesso em: 18 jul. 2020.

KALYANI, Darshita; MERTHA, Dr. Devarshi. **Paper on Searching and Indexing Using Elasticsearch**. International Journal Of Engineering And Computer Science ISSN:2319-7242 Volume 6 Issue 6, 6 jun. 2017.

KIBANA. **Elasticsearch B.V.**,2020. Kibana: A sua janela para o Elastic Stack. Disponível em < <https://www.elastic.co/pt/kibana/> >. Acesso em: 26 mar. 2020.

VUKOVIC, Aleksa; GOODWILL, James. **Apache Tomcat 7**. 1 ed. Nova York: Apress, 2011.

KIBANA 7. Versão 7.4.2. **Elasticsearch B.V.**, 2019. Disponível em: <<https://www.elastic.co/downloads/past-releases/kibana-7-4-2>>. Acesso em : 26 mar. 2020.

LEWIS, Andre. Understanding Linux CPU Load - when should you be worried ? **Scout APM**, 28 jul. 2019. Disponível em <<https://scoutapm.com/blog/understanding-load-averages>> Acesso em: 18 jul. 2020.

LIMA, Maria Teresa. **Governo Digital: muito além da desburocratização da esfera pública**, 05 ago. 2020. Disponível em <<https://sis-publique.convergenciadigital.com.br/cgi/cgilua.exe/sys/start.htm?inoid=54425&sid=15>> Acesso em: 08 set. 2020.

LOGSTASH. **Elasticsearch B.V.**, 2020. Logstash. Centralize, transforme e oculte os seus dados. Disponível em < <https://www.elastic.co/pt/logstash> >. Acesso em: 26 mar. 2020.

LOGSTASH 7. Versão 7.4.2. **Elasticsearch B.V.**, 2020. Disponível em: <<https://www.elastic.co/downloads/past-releases/logstash-7-4-2>>. Acesso em: 01 abr. 2020.

METRICBEAT 7. Versão 7.6.2. **Elasticsearch B.V.**, 2020. Disponível em: <<https://www.elastic.co/downloads/past-releases/metricbeat-7-6-2>>. Acesso em : 04 mai. 2020.

MOZILLA Firefox. Versão 80. **Mozilla Corporation**, 2020. Disponível em: < <https://www.mozilla.org/pt-BR/firefox/new/>>. Acesso em : 26 mar. 2020.

ORACLE. **Oracle Java Documentation**, 2019. Lesson: Introducing MBeans . Disponível em < <https://docs.oracle.com/javase/tutorial/jmx/mbeans/index.html> >. Acesso em: 15 set. 2020.

ROCK CONTENT. **Entenda o que é um Log File e aprenda a criar um**, 11 mar. 2020. Disponível em < <https://rockcontent.com/br/blog/log-file/#:~:text=Com%20base%20nos%20registros%20gerados,de%20eventuais%20erros%20ou%20falhas.> > Acesso em: 07 dez. 2020.

ROUSE, Margaret. Java virtual machine (JVM). **The Server Side**, abr. 2019. Disponível em < <https://www.theserverside.com/definition/Java-virtual-machine-JVM> > Acesso em: 14 dez. 2020.

SORIANO, Jaime. Monitoramento de aplicações em Java com Metricbeat e Jolokia. **Elastic Blog**, 26 jun. 2018. Disponível em <<https://www.elastic.co/pt/blog/monitoring-java-applications-with-metricbeat-and-jolokia>> Acesso em: 04 mai. 2020.

TYSON, Matthew. What is JSP? Introduction to JavaServer Pages. **InfoWorld**, 29 jan. 2019. Disponível em < <https://www.infoworld.com/article/3336161/what-is-jsp-introduction-to-javascript-pages.html> > Acesso em: 14 dez. 2020.

VISUAL Studio Code. Versão 1.49.0. **Microsoft**, 2020. Disponível em: <<https://code.visualstudio.com/>> . Acesso em: 26 mar. 2020.