INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Ferramenta Centralizadora de Monitorização

JORGE MARTINS

Relatório final (versão inicial) realizado no âmbito de Projecto e Seminário,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2020-2021

Orientadores : Engenheiro Artur Ferreira
Engenheiro Hugo José

Junho, 2021

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Ferramenta Centralizadora de Monitorização

JORGE MARTINS

Relatório final (versão inicial) realizado no âmbito de Projecto e Seminário,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2020-2021

Junho, 2021

Resumo

A necessidade de monitorização de diferentes aspectos do funcionamento de componentes informáticos torna as ferramentas de monitorização imprescindíveis a todos que requerem o uso das mesmas.

Essa mesma alta necessidade exige com que hajam múltiplas instâncias de várias ferramentas, fazendo com que a informação proveniente das ferramentas seja distribuída e dispersa através de vários ecrãs e visualizações, criando o problema de dispersão e desagregação de informação.

Este projeto visa a solução do problema apresentado através do desenvolvimento de uma aplicação web que centralize toda a informação distribuída num único ponto, apresentando esses mesmo dados numa *dashboard* eficaz e intuitiva.

Palavras-chave: Monitorização, Instâncias, Informação, Aplicação web

Abstract

Índice

Lista de Figuras **xi**

Lista de Tabelas **xiii**

Lista de Listagens **xv**

1 Introdução **1**

1.1 Problema 2

1.2 Organização do documento 2

2 Formulação do Problema **3**

2.1 Ferramentas de Monitorização 3

2.1.1 *Zabbix* 4

2.2 Desafio apresentado 5

2.3 Terminologia e conceitos 5

3 Abordagem **7**

3.1 Reuniões ClaraNet 7

3.2 Solução proposta 8

4 Implementação **9**

4.1 Interface geral para conectores 9

4.2 Conector *Zabbix* 10
4.2.1 Exemplo de pedido à API *Zabbix* 13
4.3 *Zabbix* Controller 14
4.4 Objetos *Back-end* 15
4.5 Autenticação de utilizadores 17

5 Trabalho em desenvolvimento **19**

Referências **21**

Lista de Figuras

2.1 Dashboard *Zabbix* 6

3.1 Estrutura do Projeto 8

Lista de Tabelas

4.1 Propriedades do objecto Event 15
4.2 Propriedades do objecto Trigger 16
4.3 Propriedades do objecto Host 16

Lista de Listagens

4.1 Código fonte para Interface Conector. 9

4.2 Código fonte conector Zabbix - Construtor. 10

4.3 Código fonte conector Zabbix - Método Init. 10

4.4 Código fonte conector Zabbix - Método Destroy. 11

4.5 Código fonte conector Zabbix - Método Call. 11

4.6 Código fonte conector Zabbix - Método Login. 12

4.7 Exemplo de pedido JSON - Event.get. 13

4.8 Código fonte controlador Zabbix 14

1

Introdução

As ferramentas de monitorização provam ser indispensáveis num planeta em rápida expansão tecnológica e, consequentemente, na monitorização de ditas tecnologias. Estas por sua vez permitem monitorizar diferentes componentes tais como servidores, máquinas virtuais, serviços de cloud e networks em inúmeros aspectos tais como utilização de rede, carga de CPU e consumo de espaço em disco.
Em dados serviços, estas informações, que são fornecidas em tempo real ou temporalmente mediante a necessidade dos utilizadores, são cruciais para a sua finalidade. Sem o fornecimento das mesmas, seria virtualmente impossível o controlo simples e eficaz de uma infraestrutura complexa de vários servidores, por exemplo. Contudo, existem ao dispor de um fornecedor de serviços de monitorização várias ferramentas. Todas apresentam as suas vantagens e desvantagens, porém, acabam por realizar o seu propósito final, ou seja, a monitorização de sistemas. Todavia, a disponibilidade de várias ferramentas, conforma-se devidamente noutro problema.
Este projeto foi realizado em parceria com a empresa ClaraNet. A ClaraNet é uma empresa que fornece serviços *ISP*, que se foca maioritariamente em soluções de *Cloud*, Cibersegurança e *Workplace*. Contudo, devido à sua natureza com operações relacionadas com a *Internet*, apresenta igualmente soluções de monitorização.

1.1 Problema

Devido à necessidade de monitorização de grandes quantidades de equipamentos provenientes de diferentes pontos e organizações, resultando na utilização de várias ferramentas de monitorização e de ainda múltiplas instâncias das mesmas, culminando numa enorme quantidade de informação distribuída por sua vez, em vários monitores para permitir a visualização das mesmas.
Como consequência, trabalhadores de empresas como a ClaraNet necessitam de um elevado número de ecrãs na sua área de trabalho de forma a monitorizar eficientemente um elevado número de alarmes despoletados pelas ferramentas de monitorização. Ademais da necessidade de uma parede de ecrãs que disponibiliza uma *overview* de várias instâncias de monitorização, com a finalidade de realçar eventos de maior severidade para trabalhadores que actuam sobre esses mesmos eventos.
O facto de ser apresentada uma grande quantidade de informação dispersa por vários monitores poderá dificultar a capacidade e eficiência de monitorização por parte de trabalhadores que estejam a deparar-se com as mesmas.

1.2 Organização do documento

A organização deste documento divide-se em cinco capítulos. No segundo capítulo, será aprofundado o desafio apresentado pela ClaraNet, juntamente com uma breve introdução às ferramentas de monitorização e o seu uso. No terceiro capítulo apresenta-se a solução desenvolvida para o problema em questão. No quarto apresenta-se o funcionamento da solução apresentada e por fim, no quinto capítulo são apresentadas as tarefas em desenvolvimento.

2

Formulação do Problema

Neste capítulo será realizada uma análise em maior profundidade ao problema apresentado. Na secção 2.1 é realizada uma apresentação às ferramentas de monitorização, juntamente com um destaque sobre a ferramenta *Zabbix* na secção 2.1.1. Seguidamente, na secção 2.2, é aprofundado o desafio apresentado.
Para primeiro entender o problema, é necessário uma contextualização sobre as ferramentas de monitorização e como estas funcionam.

2.1 Ferramentas de Monitorização

Tal como previamente mencionando, as ferramentas de monitorização têm como intuito possibilitar a monitorização de diferentes componentes tais como servidores, máquinas virtuais, serviços de cloud e *networks*. Estas conseguem fornecer várias informações tais como métricas de monitorização, utilização de redes, *workload* de um *CPU* e consumo de espaço em disco.

Neste projeto temos três ferramentas como referência: o *Zabbix*[1][8], o *Nagios*[2][4] e o *Nimsoft*[3][5]. Foram seleccionadas estas ferramentas como referência, não obstante da

[1] https://www.zabbix.com/
[2] https://www.nagios.org/
[3] https://support.nimsoft.com/

quantidade disponível no mercado, devido ao facto de serem as ferramentas mais utilizadas por parte da ClaraNet. Estas ferramentas disponibilizam API's de forma a que seja possível existir uma interação entre aplicações desenvolvidas por terceiros com as mesmas.

Todavia, devido a restrições de tempo e pelo desafio apresentado pela ClaraNet ser originalmente um projeto previsto para pelo menos dois alunos, foi realizado um ajuste de requisitos. Neste contexto, definiu-se como objetivo o desenvolvimento de uma interface que permita à implementação futura de outras ferramentas de monitorização, sendo realizada a implementação para a ferramenta de monitorização *Zabbix*.

2.1.1 *Zabbix*

Utilizou-se o *Zabbix* como a ferramenta implementada por instrução da ClaraNet, sendo a ferramenta que tem maior uso na empresa. A ferramenta *Zabbix* permite vários tipos de monitorização:

- Verificações simples que permitem verificar a disponibilidade e respotas de servicos simples como *Simple Mail Transfer Protocol* (SMTP) e *Hypertext Transfer Protocol* (HTTP) sem a instalação de software no *host* monitorizado.
- Um agente *Zabbix* poderá ser instalado em *hosts* com sistemas operativos do tipo Linux e Windows para monitorizar estatísticas tais como uso de CPU, uso de redes, espaço de memória em disco, entre outros.
- Como alternativa à instalação de agentes nos *hosts*, o *Zabbix* também suporta monitorização através de verificações SNMP, TCP e ICMP, suportanto assim uma grande variedade de mecanismos de notificação *"near-real-time"*.

No desenvolvimento desta web-app, utilizou-se a documentação referente à versão 4.2 da ferramenta *Zabbix*[4][9].

[4] https://www.zabbix.com/documentation/4.2/manual/api/reference

2.2 Desafio apresentado

Foi lançado desafio para o desenvolvimento de uma aplicação web que soluciona a distribuição elevada da informação proveniente das ferramentas de monitorização, centralizando essa informação numa única aplicação. Definiu-se como requisitos principais:

- Possibilitar a configuração de múltiplos conectores para ferramentas.
- Implementação de alarmes e listas agregadoras dos mesmos.
- Existência de perfis de acesso diferenciados para diversos tipos de utilizadores.
- Configurações a nível de utilizadores.
- Implementação de Dashboards analíticos sobre a aplicação.

2.3 Terminologia e conceitos

Existem algumas palavras chave que permitem obter uma melhor compreensão sobre o projecto em geral, nomeadamente:

Conector - Designa-se como conector a ligação entre a web-app com uma instância de uma ferramenta de monitorização. Por exemplo, é necessário a criação de um objecto não estático que estabeleça a conexão com o *end-point* http://195.22.17.158/zabbix/ de forma a permitir a interação entre a web-app com esse servidor *Zabbix*.

Alarmes - As ferramentas de monitorização funcionam na base de alarmes, também designados por eventos. Estes são despoletados através de *triggers* criados manualmente. Por exemplo, imaginemos que estamos a monitorizar um servidor e adicionamos um trigger para despoletar um alarme quando o mesmo tiver ocupado 80% da sua capacidade de armazenamento. Ao ser atingido esse valor, a ferramenta envia um alarme a avisar que o mesmo ocorreu. É possível realizar diferentes ações sobre estes alarmes, tais como adicionar texto complementar ou realizar um *acknowledge*.

Dashboards analíticos - Os dashboards analíticos são desenvolvidos na componente *Front-end* da web-app e têm como intuito permitir a visualização simples e eficaz sobre os componentes monitorizados. Esta deverá ser adaptável ao gosto do utilizador permitindo ser realçado diversos componentes mediante as necessidades do utilizador. Apresenta-se um exemplo de um dashboard analítico através da Figura 2.1, proveniente da ferramenta *Zabbix* com o intuito de representar o expectável.

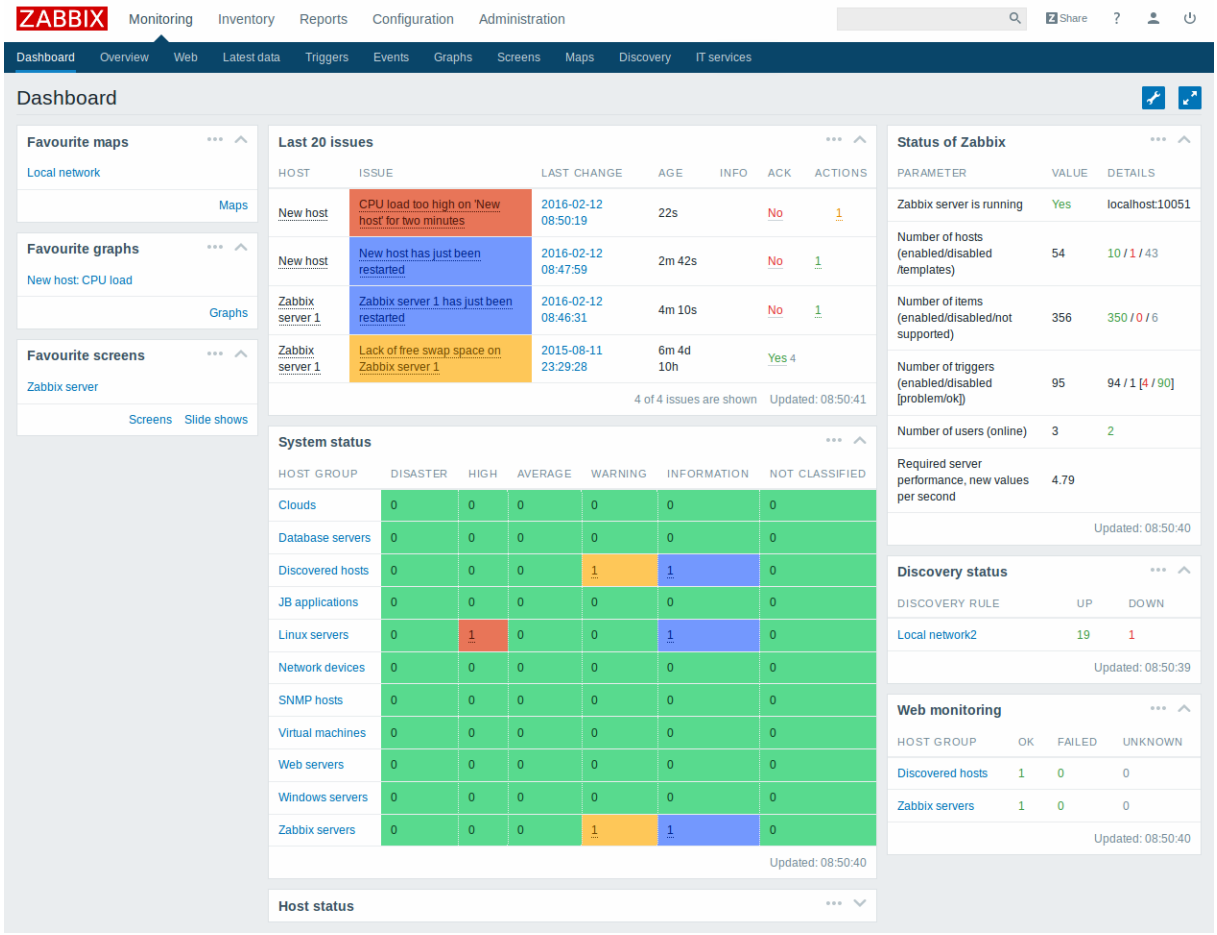

Figura 2.1: Dashboard *Zabbix*

Como podemos observar, são apresentados vários dados informativos distribuídos por vários componentes, cujos são alteráveis mediante as necessidades do utilizador. Mediante o foco principal de monitorização, poderá ser de maior interesse monitorizar aspectos referentes à quantidade de tráfego a entrar ou sair de por exemplo um servidor. Numa outra situação em que estejam a ser monitorizadas uma maior quantidade máquinas virtuais, poderá porventura ser de maior interesse ao utilizador ter um componente que apresente a utilização CPU de cada máquina virtual.

3

Abordagem

3.1 Reuniões ClaraNet

Numa primeira fase, foram realizadas reuniões com a ClaraNet com o intuito de ser feita a apresentação da empresa, às ferramentas de monitorização e dos membros da empresa que iriam acompanhar no desenvolvimento deste projeto. Foi definido que seriam realizadas reuniões semanais curtas de forma a que houvesse um maior acompanhamento, especialmente por parte do Engenheiro Hugo José, e esclarecimento de dúvidas quanto ao que seria necessário desenvolver.
Seguidamente, foi necessário chegar a um acordo quanto às tecnologias a serem utilizadas no desenvolvimento do projeto. Neste caso foi necessário chegar a um consenso, nomeadamente na definição de tecnologias do *Back-end*, devido ao facto de este ser um projeto que poderá ser usado e desenvolvido posteriormente pela ClaraNet, tornou-se necessário utilizar uma linguagem que esteja em conformidade com os trabalhadores da empresa.
Por último, requisitaram-se duas máquinas virtuais, uma para alocação da web-app e outra para a alocação de um servidor *Zabbix*. A aquisição destas mesmas máquinas acabou por ser algo demorada, principalmente por questões de segurança da empresa. Numa primeira fase foram entregues as máquinas, sendo a única camada de proteção existente, as credenciais de acesso às próprias máquinas. Contudo, isto apresentava

vulnerabilidades tais que poderiam apresentar perigo para a ClaraNet. Consequentemente, o acesso foi retirado por um curto espaço de tempo, sendo adicionado posteriormente como medida de segurança, o acesso restrito ao endereço *IP* onde o projeto estaria a ser desenvolvido.

3.2 Solução proposta

Formulou-se a ideia do desenvolvimento de uma single page web-app, representada pela Figura 3.1.:

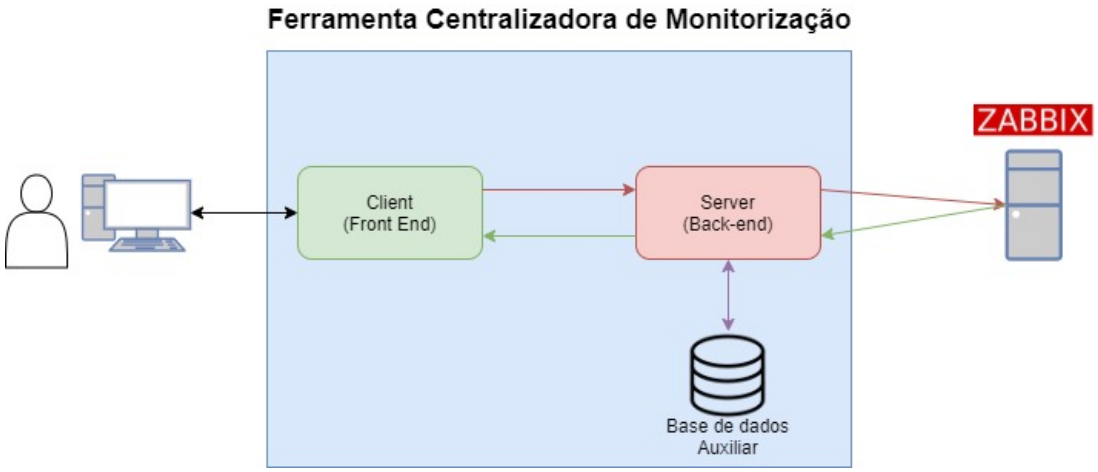

Figura 3.1: Estrutura do Projeto

Tal como representado pela Figura 3.1, será desenvolvida uma componente *Front-end* e uma componente *Back-end* com o auxílio de uma base de dados relacional. Para o *front-end* decidiu-se utilizar a biblioteca *JavaScript*, *React*[1][6]. Utilizou-se esta biblioteca pela sua simplicidade e forte capacidade de construir uma componente *client-side* e igualmente por utilizar a linguagem *JavaScript*, sendo uma linguagem à qual a ClaraNet é familiar. Em termos de *back-end*, utilizou-se a *framework Spring-boot*[2][7], desenvolvida em *Java*. Originalmente, foi proposto que a componente *server-side* fosse desenvolvida na linguagem *Kotlin*, contudo, devido ao facto de ser uma linguagem que é pouco familiar à ClaraNet, foi acordado que o desenvolvimento deveria ser realizado em *Java*. Por fim, foi desenvolvida uma base de dados auxiliar desenvolvida em *MariaDB*[3][3], que estará associada maioritariamente com o tratamento de utilizadores e autenticação.

[1] https://reactjs.org/
[2] https://spring.io/projects/spring-boot
[3] https://mariadb.org/

4

Implementação

4.1 Interface geral para conectores

Desenvolveu-se uma interface que define o que um conector terá que implementar obrigatoriamente. Podemos observar a definição da mesma através do seguinte código.

```
1  public interface Connector {
2
3      void init();
4
5      void destroy();
6
7      boolean login(String user, String password);
8
9      JSONObject call(Request request);
10 }
```

Listagem 4.1: Código fonte para Interface Conector.

Definiram-se quatro funções básicas as quais irão permitir realizar as operações críticas para estabelecer e manter um conector com as API's das ferramentas de monitorização. O método **init** estabelece uma conexão inicial e porventura manterá a mesma. Para terminar esta conexão, deverá ser utilizado o método **destroy**. O método **login**

será utilizado para realizar a autenticação às API's, recebendo um username e password, retornando um booleano de forma a sinalizar o resultado da autenticação. Por fim, o método **call** será utilizado para realizar os pedidos às API's das ferramentas, retornando um *JSONObject*, sendo este a resposta proveniente da API. A implementação destes métodos será exemplificada na secção 4.2.

4.2 Conector *Zabbix*

Apresenta-se a implementação do conector *Zabbix*, através da interface geral desenvolvida, juntamente com o construtor do objecto *ZabbixConnector*.

Construtor

```
1     public ZabbixConnector(String url){
2         try {
3             uri = new URI(url.trim());
4         } catch (URISyntaxException e) {
5             throw new RuntimeException("invalid url", e);
6         }
7     }
```

Listagem 4.2: Código fonte conector Zabbix - Construtor.

Com o propósito de serem realizados pedidos HTTP à API do *Zabbix*, requereu-se às bibliotecas *Apache HTTP Client*. Para realizar devidamente um pedido através será necessário indicar o end-point a realizar o pedido, o qual será recebido ao ser construido um objeto do tipo ZabbixConnector, tal como apresentado.

Init e Destroy

```
1     @Override
2     public void init() {
3         if (httpClient == null) {
4             httpClient = HttpClients.custom().build();
5         }
6     }
```

Listagem 4.3: Código fonte conector Zabbix - Método Init.

```
    public void destroy() {
        if (httpClient != null) {
            try {
                httpClient.close();
            } catch (Exception e) {
                logger.error("close httpclient error!", e);
            }
        }
    }
```

Listagem 4.4: Código fonte conector Zabbix - Método Destroy.

Nas duas listas apresentadas podemos observar como é inicializado e terminado a instância do objecto que permite a realização de pedidos HTTP com a API Zabbix. Neste caso, é utilizado uma variável denominada por httpClient do tipo CloseableHttpClient, que será inicializada no método Init e terminada no método Destroy.

Call

```
    public JSONObject call(Request request) {
        if (request.getAuth() == null) {
            request.setAuth(this.auth);
        }

        try {
            HttpUriRequest httpRequest = org.apache.http.client.methods.
    RequestBuilder.post().setUri(uri)
                    .addHeader("Content-Type", "application/json")
                    .setEntity(new StringEntity(JSON.toJSONString(request),
     ContentType.APPLICATION_JSON)).build();
            CloseableHttpResponse response = httpClient.execute(httpRequest
    );
            HttpEntity entity = response.getEntity();
            byte[] data = EntityUtils.toByteArray(entity);
            return (JSONObject) JSON.parse(data);
        } catch (IOException e) {
            throw new RuntimeException("ZabbixApi call exception!", e);
        }
    }
}
```

Listagem 4.5: Código fonte conector Zabbix - Método Call.

Tal como previamente mencionado, o método **call** será responsável por todos os pedidos realizados entre a web-App com a API, neste caso do *Zabbix*. Este recebe um *Request* previamente já formatado, envia o pedido em formato HTTP e devolve a resposta no formato JSONObject.

Login

```
1     public boolean login(String user, String password) {
2         this.auth = null;
3         Request request = RequestBuilder.newBuilder().paramEntry("user",
    user).paramEntry("password", password)
4                 .method("user.login").build();
5         JSONObject response = call(request);
6         String auth = response.getString("result");
7         if (auth != null && !auth.isEmpty()) {
8             this.auth = auth;
9             return true;
10        }
11        return false;
12    }
```

Listagem 4.6: Código fonte conector Zabbix - Método Login.

Demonstrou-se a utilização do método **call** através do método **login**. É formulado um Request através dos parâmetros user e password recebidos, para posteriormente ser chamado o método **call**, para realizar o pedido de login.

4.2.1 Exemplo de pedido à API *Zabbix*

Demonstra-se seguidamente um exemplo de um pedido à API Zabbix, neste caso um pedido de obtenção de eventos associados a um trigger.

```
{
    "jsonrpc": "2.0",
    "method": "event.get",
    "params": {
        "output": "extend",
        "select_acknowledges": "extend",
        "selectTags": "extend",
        "selectSuppressionData": "extend",
        "objectids": "13926",
        "sortfield": ["clock", "eventid"],
        "sortorder": "DESC"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Listagem 4.7: Exemplo de pedido JSON - Event.get.

Um pedido JSON à ferramenta *Zabbix* necessita obrigatoriamente dos seguintes parâmetros:

- **jsonrpc** - Identifica a versão do protocolo JSON-RPC[1][2] utilizada pela a API. É necessário enviar o valor **2.0** pois esta é a versão implementada pela API *Zabbix*.
- **method** - Indica o método a ser invocado na API. Neste caso está a ser realizada uma operação **event.get** que obtem designados eventos mediante os parâmetros especificados.
- **params** - Parâmetros que serão passados ao método a ser invocado na API
- **auth** - Referente ao *token* recebido por parte do utilizador após realizar o login na API.
- **id** - Identificador arbitrário do pedido

[1] https://www.jsonrpc.org/specification

4.3 *Zabbix* Controller

Com o objectivo de ser possível ter a conexão simultânea com várias instâncias *Zabbix*, foi criada uma estrutura de dados HashMap, onde a chave será gerada automaticamente ao ser realizada uma conexão com uma instância *Zabbix* e o valor associado, a própria instância. Desta forma, é simplificado o processo de obtenção da instância pretendida, sendo apenas necessário fornecer o ID atribuído à instância ao método getZab. Realça-se o facto do código apresentado na listagem seguinte ser meramente exemplificativo de uma abordagem à obtenção de um conector estável com a API de um servidor *Zabbix*.

```
1  @RestController
2  public class ConnectorController {
3
4      private ZabbixConnector zab;
5      private static final AtomicInteger nextId = new AtomicInteger(0);
6      private static HashMap<String,ZabbixConnector> connectors = new HashMap
       <>();
7
8      @GetMapping("/zabbixCon")
9      public void getZabbixCon(){
10
11         String url = "http://195.22.17.158/zabbix/api_jsonrpc.php";
12
13         zab = new ZabbixConnector(url);
14         zab.init();
15         nextId.incrementAndGet();
16
17         connectors.put(nextId.toString(),zab);
18
19
20         String user = "";
21         String password = "";
22         boolean login = zab.login(user, password);
23     }
24
25     public static ZabbixConnector getZab(String id){
26         return connectors.get(id);
27     }
28 }
```

Listagem 4.8: Código fonte controlador Zabbix

Note-se que neste exemplo, o url fornecido termina com "api_jsonrpc.php". Todos os pedidos realizados à API *Zabbix* necessitam de terminar dessa forma para que seja possível realizar pedidos. O pedido à web-app http://localhost:8080/zabbixCon cria uma instância ZabbixConnector e insere a mesma no HashMap connectors. Seguidamente efectua o login, de forma a possibilitar que este conector efetue futuramente pedidos à API *Zabbix*.

4.4 Objetos *Back-end*

Apresentam-se, numa primeira fase, os objetos principais que serão construidos no *Back-end* para posteriormente serem apresentados no *Front-end*. Estes são construídos através de diversos pedidos à API do Zabbix.

Event

Tabela 4.1: Propriedades do objecto Event

Propriedades	Tipo	Descrição
EventId	String	ID único de um evento
Object	Integer	Tipo de objecto associado ao evento
Object ID	String	ID único do objecto associado ao evento
Acknowledged	Integer	Sinaliza se o evento foi Acknowledged
Clock	Timestamp	Regista quando o evento foi criado
Value	Integer	Estado do objecto associado ao evento
Severity	Integer	Severidade do evento
Suppressed	Integer	Verifica se o evento encontra-se suprimido

Um evento poderá estar associado a vários tipos de objectos, neste projeto contudo, apenas será de interesse, numa primeira fase, os eventos criados por *triggers*, ou seja de uma forma geral, o campo **Object** terá o valor **0**, que corresponde devidamente a um evento criado por um *trigger*. Uma propriedade importante proveniente do Event é devidamente o campo **Acknowledged**, cujo permite saber se algum utilizador realmente já reconheceu a existência desse mesmo evento e porventura realizou algo sobre esse evento. O campo **Value**, no contexto do evento ter sido accionado por um *trigger*, poderá ter os valores **0** e **1**, representando um "Ok"ou "Problem", respetivamente. Em relação ao campo **Severity**, poderão ser recebidos no total 6 valores diferentes: **0** que

representa severidade não classificada; **1** que indica que se trata apenas de uma informação; **2** representando um aviso (severidade baixa); **3** indicando uma severidade média; **4** representando severidade alta; Por fim o valor **5**, alertando para uma severidade catastrófica.

Trigger

Tabela 4.2: Propriedades do objecto Trigger

Propriedades	Tipo	Descrição
Expression	String	Expressão reduzida do Trigger
Name	String	Nome do Trigger
Description	String	Descrição do Trigger

O campo **Expression** permite que sejam criados *triggers* sobre vários componentes presentes nos *hosts* a que estejam a ser monitorizados. Apresentam-se dois exemplos de expressões que poderão ser utilizadas como trigger:

1. "expression": "{server:system.cpu.load.avg(1h)}/{server:system.cpu.load.avg(1h,1d)>2}"
2. "expression": "{www.zabbix.com:vfs.file.cksum[/etc/passwd].diff()=1}"

Na primeira expressão, está a ser comparada a carga atual de CPU de um servidor com a carga de CPU do mesmo servidor, no mesmo horário do dia anterior. Esta expressão será verdadeira se a carga da última hora for duas vezes superior à carga do dia anterior, relativamente à hora.
Na segunda expressão, é realizada uma simples verificação se o ficheiro na directoria "/etc/passwd"foi alterado. Esta expressão retorna verdadeiro quando o último valor da verificação "checksum"do ficheiro for diferente da verificação anterior.

Host

Tabela 4.3: Propriedades do objecto Host

Propriedades	Tipo	Descrição
Name	String	Nome do Host
Proxy_hostid	String	ID do Proxy que é utilizado para monitorizar o host
Description	String	Descrição do Host

4.5 Autenticação de utilizadores

A web-app utiliza *Basic HTTP Authentication Scheme*[2][1], viabilizando uma autenticação simples, porém eficaz. Ao ser realizado o registo na web-app, será guardado o *username* juntamente com um campo encoded e ainda por default será atribuido o nível 0 de permissões. Para verificar se o utilizador se encontra devidamente autenticado será enviado no header do pedido (*Front-end* para *Back-end*) esse mesmo campo encoded.

Exemplo:
- Authorization: Basic Base64(username:password)
Resultado:
- Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=

Este valor encoded será mantido no *Front-end* através da instância *LocalStorage*, até ser realizado o *logout*, o qual irá remover esse mesmo valor presente na instância. É de notar que para realizar qualquer tipo de operação na web-app é necessário estar devidamente autenticado.

[2] https://datatracker.ietf.org/doc/html/rfc7617

5

Trabalho em desenvolvimento

O plano está a ser cumprido dentro do expectável, é de mencionar que principalmente numa fase inicial o trabalho sofreu uma quantidade considerável de atrasos, atrasando devidamente o desenvolvimento do projeto. Notavelmente, a aplicação encontra-se num estado inicial, não apresentando a componente *client-side* desenvolvida e com a componente *server-side* num estado de teste.

Referências

[1] *Http authentication scheme*. URL: https://datatracker.ietf.org/doc/html/rfc7617.

[2] *What is jsonrpc?* URL: https://www.jsonrpc.org/specification/.

[3] *Mariadb*. URL: https://mariadb.org/.

[4] *What is naggios?* URL: https://www.nagios.org/.

[5] *What is nimsoft?* URL: https://support.nimsoft.com/.

[6] *React*. URL: https://reactjs.org/.

[7] *Spring-boot*. URL: https://spring.io/projects/spring-boot/.

[8] *What is zabbix?* URL: https://www.zabbix.com/.

[9] *Zabbix 4.2 documentation*. URL: https://www.zabbix.com/documentation/4.2/manual/api/reference.