Gestão de Redes

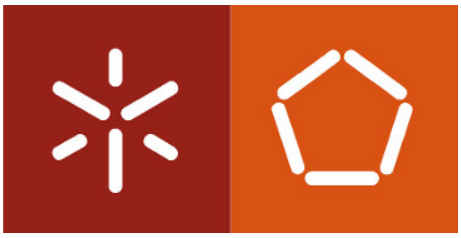
Trabalho Prático 2

15 de janeiro de 2021

---

a85367 Luís Lopes

Ferramenta de Monitorização



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

Lista de Figuras

1 Introdução

Nos dias correntes, a utilização de dispositivos informáticos, nomeadamente o computador, é cada vez mais adotada no dia a dia, seja em ambientes de trabalho ou mesmo para uso pessoal. Porém, com o crescente número de utilizadores surgem também várias ameaças para quem utiliza esta ferramenta de trabalho. Através de mallware, por exemplo, podemos destruir todo um trabalho efetuado até então pelo utilizador. Desta forma torna-se necessário a utilização de um sistema de monitorização de processos, para que se possa controlar toda a atividade, suspeita ou não, que ocorre nos nossos computadores. Assim se enquadra este trabalho prático da unidade curricular Gestão de Redes do curso do Mestrado Integrado em Engenharia Informática, que propõe a implementação de uma ferramenta de monitorização de processos.

2 Concepção

De acordo com a proposta do enunciado, a ferramenta de monitorização teria que ser modular de forma a que a captura de informação, análise e apresentação da mesma pudessem ser separadas e desenvolvidas por diferentes indivíduos sem implicar a alteração de algum dos módulos. Assim sendo a ferramenta é constituída por três programas distintos: **SNMP-Monitor**, **SNMP-Notifier** e por fim **SNMP-Monitor(Interface)**.

3 Implementação

3.1 SNMP-Monitor

O SNMP-Monitor foi desenvolvido na linguagem Java, visto a familiaridade com a mesma e a sua vasta informação e documentação online. Para a monitorização dos processos através de um servidor **SNMP** é necessário o uso de *libraries* externas, neste caso foi utilizada a *library* **SNMP4J** visto ser a adequada para a linguagem Java. Para conferir uma maior modularidade foram adotados dois formatos de texto para armazenar as informações obtidas, **JSON** e **TXT**. O formato **TXT** foi escolhido para criar um ficheiro de log, armazenado na máquina local, de todas as informações recolhidas. Por outro lado escolheu-se também guardar as informações em ficheiro do formato **JSON** para que se pudesse criar uma API de dados online, desta forma qualquer *Front-End Developer* consegue aceder às informações obtidas para criar uma interface independente da ferramenta desenvolvida. Utilizou-se o serviço Azure da Microsoft para ser possível colocar online o **Json-Server**.

3.1.1 Classes

Adotou-se uma programação orientada a objetos, baseada num modelo MVC, no entanto, a view não é necessária neste módulo, logo não foi implementada, não existindo qualquer interação homem-máquina.

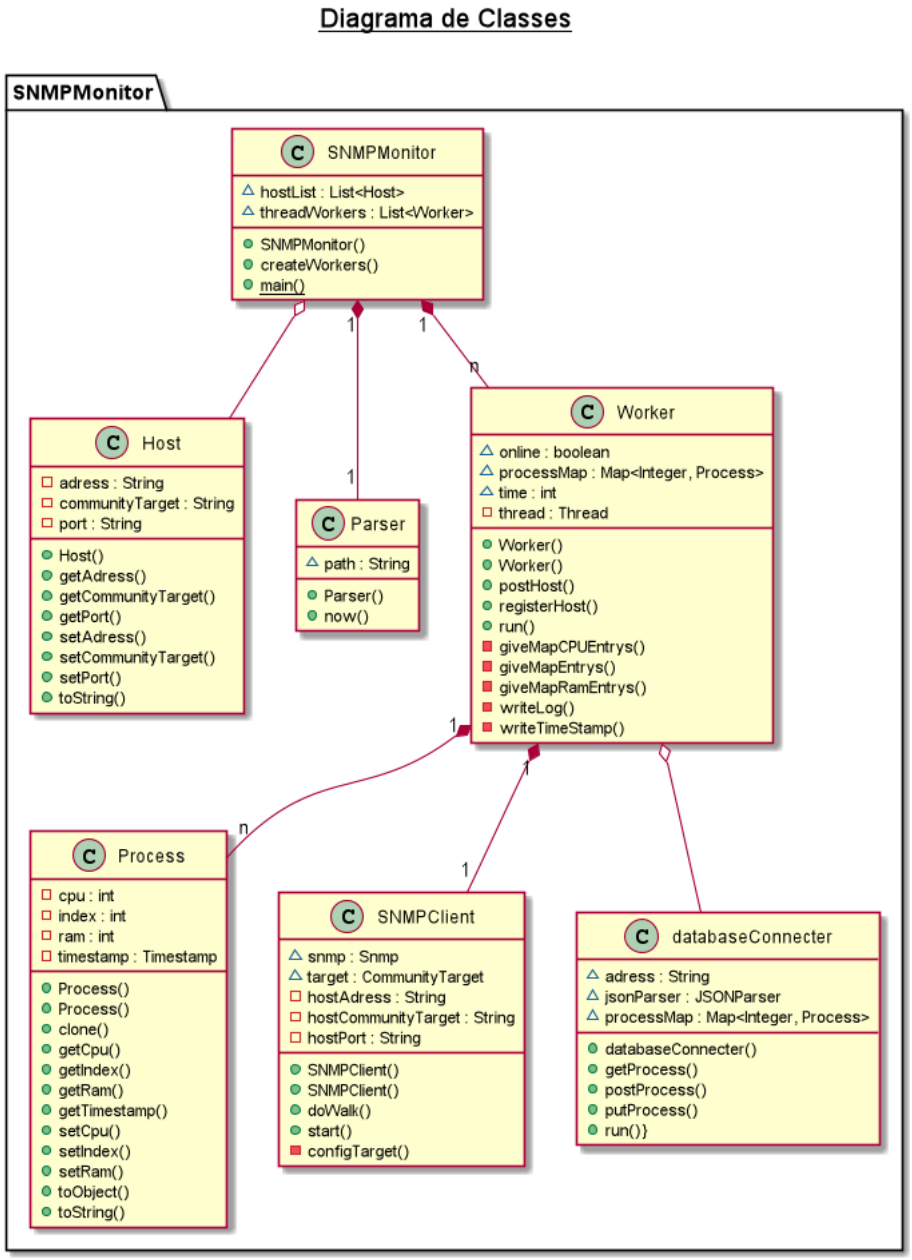


Figura 1: Diagrama de Classes SNMP-Monitor

Como podemos verificar pela figura 1 a implementação adota várias classes, seja para a efetuar *parsing* de ficheiros de texto, para conectar-se à base de dados, neste caso não relacional, implementada em Json-Server pelo Azure. O módulo inicia-se criando um primeiro objeto do tipo SNMP-Monitor. É nesta classe que, através da variável **giveMeHosts** do tipo **Parser**, se executa o *parsing* do ficheiro de configuração, alimentando a lista de **Host hostList**. Após a realização deste procedimento estamos preparados para criar uma classe **Worker** para cada *host* indicado. Este último procedimento acontece no método **createWorkers(String flag)**, que através de um ciclo *for* vai percorrendo a lista **hostList** para que possa criar uma thread para cada elemento presente. Assim temos um programa multi-thread capaz de monitorizar **N** número de *hosts*. Por predefinição todo este módulo é conectado à base de dados, no entanto se o utilizador não o desejar poderá introduzir a *flag* **'-off'** que implementa um *bypass* na ligação. A classe Worker contém uma variável privada do tipo SNMPClient, que por sua vez contém todas as variáveis e métodos necessários, com a ajuda do SNMP4J, para percorrer a mib de forma a encontrar a informação requerida. Após percorrer a mib a classe worker vai preencher o **processMap** com as informações retiradas. É necessário aceder a três tabelas distintas para recolher as informações necessárias, daí serem, no método **run()** executadas três 'caminhadas'. As tabelas encontram-se nos seguintes **OID**s:

ID	.1.3.6.1.2.1.25.4.2.1.1
CPU	.1.3.6.1.2.1.25.4.2.1.1.1
RAM	.1.3.6.1.2.1.25.4.2.1.1.2

Tabela 1: OIDs necessários

A classe worker faz o *refresh* num tempo não inferior a 30 segundos. Foi definido este valor, pois a atualização dos dados pelo servidor snmp também não ocorre em tempo inferior ao escolhido. Após a recolha de toda a informação, se a ferramenta estiver em modo *online*, numa nova thread é criada um objeto do tipo databaseConector que receber o **processMap** preenchido. É nesta classe que se executa todos os pedidos **HTTP** necessários para atualizar a base de dados da *cloud*. Simultaneamente escrevem-se as novas informações no ficheiro de log.

3.2 SNMP-Monitor(Interface)

Para a realização do segundo módulo proposto foi utilizada a ferramenta **Express.js**, novamente a familiaridade foi um fator decisivo na escolha da ferramenta. Em conjunção utilizaram-se também ferramentas como o **Axios**, para efetuar pedidos à base de dados e **pug** para o molde das páginas html a apresentar. Este módulo reage a pedidos http e contém alguns métodos necessários para o processamento da informação recebida.

3.2.1 Funcionamento

Quando se acede à página inicial são apresentadas duas listas com os mesmos elementos, no entanto, a primeira apresenta os links para a lista de processos dos diferentes hosts, a segunda lista permite aceder a uma página com *donuts charts* comparando os processos do host selecionado. Após aceder à lista de processos podemos aceder a cada página individual dos processos, o que nos leva a uma nova página html com dois gráficos, referentes ao uso da RAM e do CPU pelo processo. Estas informações estão também disponíveis em formato texto, é importante referir que toda a informação apresentada corresponde às ultimas sete medições.

3.2.2 Interface

Figura 2: Página Inicial

Lista de Processos

ID Processo	CPU	RAM	Last Timestamp
1	309	11928	2020-12-21 20:32:52.598
2	0	0	2020-12-21 20:32:52.598
3	0	0	2020-12-21 20:32:52.598
4	0	0	2020-12-21 20:32:52.598
6	0	0	2020-12-21 20:32:52.598
9	0	0	2020-12-21 20:32:52.598
10	6	0	2020-12-21 20:32:52.598
11	115	0	2020-12-21 20:32:52.598
12	1	0	2020-12-21 20:32:52.598

Figura 3: Lista de Processos

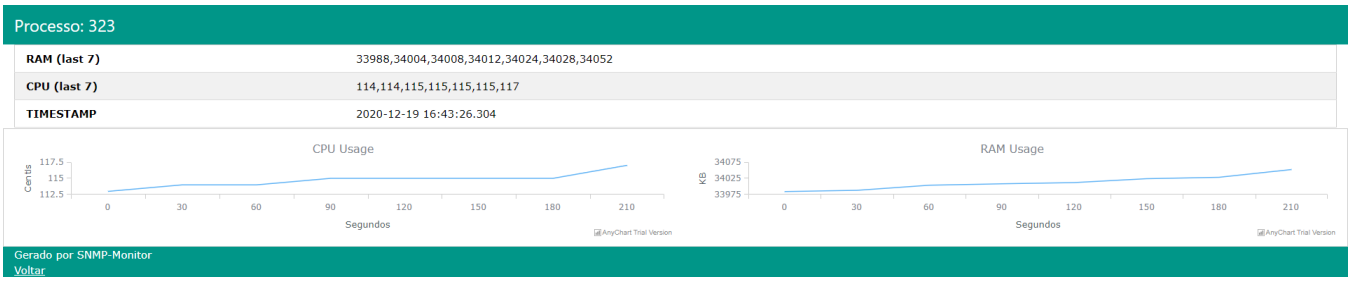


Figura 4: Página Individual de Processos

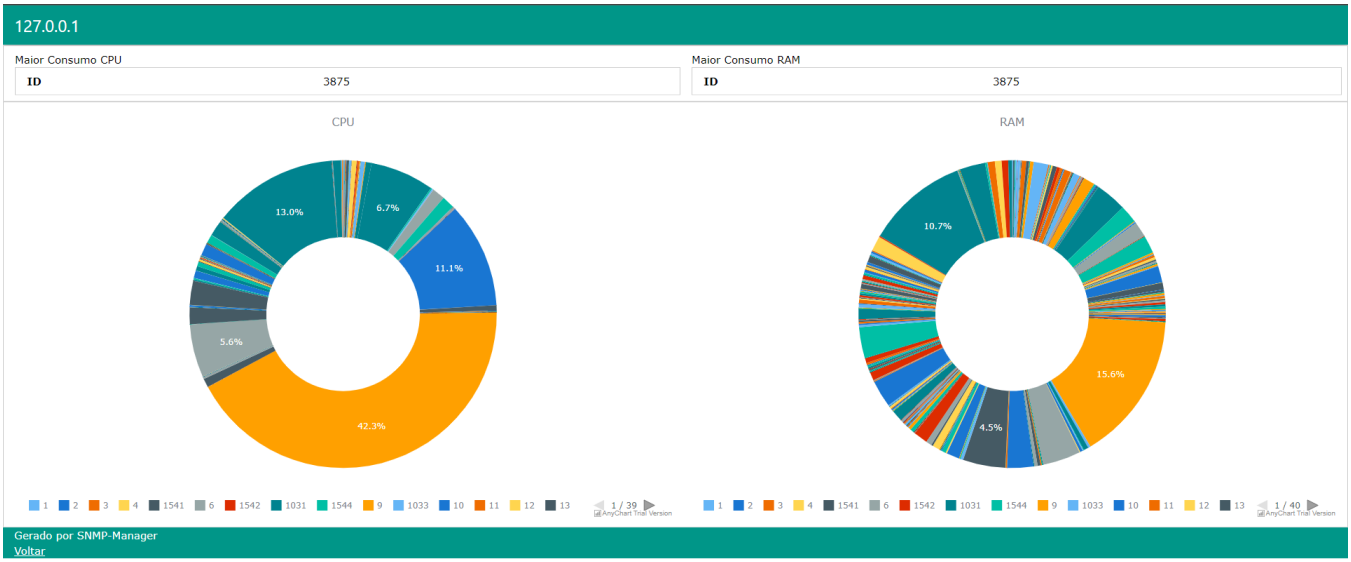


Figura 5: Pagina Geral de Processos

3.2.3 Implementação Azure

Nos dias correntes o uso do computador num ambiente de trabalho aumentou exponencialmente, desta forma, para um melhor controlo dos recursos de trabalho, uma empresa pode adotar esta ferramenta para monitorizar os seus dispositivos. Assim uma ferramenta só acessível nos dispositivo local não será a mais útil. Dito isto foi necessário implementar a ferramenta de um forma online, para que seja acessível de qualquer parte da empresa e mesmo do planeta, desde que com acesso à internet. Foi escolhida os serviços oferecidos pela Azure para a implementação online, isto por ser uma ferramenta gratuita para serviços de aplicações e alegadamente segura visto pertencer à Microsoft. Desta forma a ferramenta ficou completamente disponível no seguinte endereço *web* `http://snmp-monitor.azurewebsites.net/`.

4 SNMP-Notifier

A proposta para o terceiro módulo corresponde a uma ferramenta que através dos dados recolhidos notifica o utilizador de algum comportamento anormal. Para desenvolver este módulo recorreu-se à linguagem Java aliada com algumas *libraries* como **Simple-Json** e **Jakartmail**. Através de threads atingimos um programa que em paralelo analisa os diferentes *host* indicados no **config.json**.

4.1 Classes

O **SNMP-Notifier** é constituído por várias classes com propósito de ser mais simplificada apesar de estar mais dividido. De facto existem cinco classes. O *startup* do programa começa na classe **notifier**, é nesta que através da **Parser**, se inicia às varias threads **worker** para cada *host*.

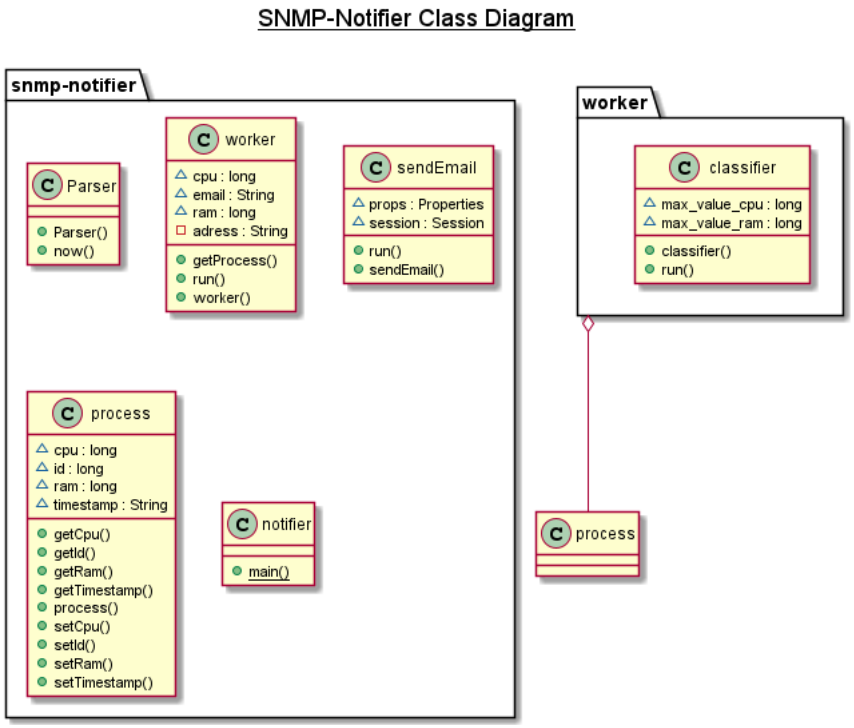


Figura 6: Diagrama de Classes do SNMP-Notifier

4.2 Funcionamento

Após ocorrer a criação de um objeto **worker** numa nova thread, este executa um pedido **http** do tipo **GET** à base de dados, recebendo assim um jsonarray com toda a informação dos processos do *host* pretendido. Através da classe **classifier** pertencente ao worker vai analisar de certo modo classificar se o utilizador precisa de ser notificado que os valores são suspeitos. Quando isto acontece, através da classe **sendEmail** envia um email ao utilizador. Para iniciar o programa é necessário indicar três argumentos `[max_value_cpu] [max_value_ram] [to_mail]`.

5 Conclusão

Este trabalho permitiu aprofundar os conhecimentos relativos à **MIB** e ao **SNMP**. Toda a pesquisa envolvida ajudou na consolidação e perceção dos conhecimentos. Dito isto podemos afirmar que os objetivos foram atingidos e todos os módulos requisitados implementados, claro que, sempre com espaço para melhorias e novas funcionalidades. Uma ferramenta deste género é bastante importante, por exemplo, na defesa contra ataques à integridade nos dispositivos conectados à internet, desta forma, ao monitorizar os processos conseguimos ter uma perceção se existe algum processo fora do comum ou com uma utilização de recursos imensa.

A config.json

Conteúdo do ficheiro "*config.json*".

```
{
  "hosts":
  [
    {
      "adress": "127.0.0.1",
      "port": "161",
      "communityTarget": "gr2020"
    }
  ]
}
```

B 'host'.txt

Exemplo de "*logger*".

```
2020-12-11 at 16:28:22 WET
1#518#13232#2020-12-11 16:28:22.711
2#1#0#2020-12-11 16:28:22.712
1026#9#20236#2020-12-11 16:28:22.712
3#0#0#2020-12-11 16:28:22.712
4#0#0#2020-12-11 16:28:22.712
5#10#0#2020-12-11 16:28:22.712
6#0#0#2020-12-11 16:28:22.712
9#0#0#2020-12-11 16:28:22.712
10#8#0#2020-12-11 16:28:22.712
1034#57#13232#2020-12-11 16:28:22.712
11#126#0#2020-12-11 16:28:22.713
12#1#0#2020-12-11 16:28:22.713
13#0#0#2020-12-11 16:28:22.713
17933#0#0#2020-12-11 16:28:22.713
14#0#0#2020-12-11 16:28:22.713
1038#18#5116#2020-12-11 16:28:22.713
15#0#0#2020-12-11 16:28:22.713
16#0#0#2020-12-11 16:28:22.713
4624#0#0#2020-12-11 16:28:22.713
17#13#0#2020-12-11 16:28:22.713
529#0#0#2020-12-11 16:28:22.713
1041#0#1720#2020-12-11 16:28:22.713
4625#0#0#2020-12-11 16:28:22.714
18#6#0#2020-12-11 16:28:22.714
530#0#0#2020-12-11 16:28:22.714
4626#2#0#2020-12-11 16:28:22.714
531#0#0#2020-12-11 16:28:22.714
20#0#0#2020-12-11 16:28:22.717
2068#29#25964#2020-12-11 16:28:22.717
21#0#0#2020-12-11 16:28:22.718
1045#17202#45528#2020-12-11 16:28:22.718
22#0#0#2020-12-11 16:28:22.718
23#13#0#2020-12-11 16:28:22.718
24#4#0#2020-12-11 16:28:22.718
1560#0#460#2020-12-11 16:28:22.718
26#0#0#2020-12-11 16:28:22.718
27#0#0#2020-12-11 16:28:22.718
```