# Universidade do Minho

Gestão de Redes

Trabalho Prático Nº 3

# Servidor de criação e monitorização de containers com interface SNMP


Nuno Leite
A70132

Bruno Carvalho
A76987

Gonçalo Duarte
A77508


9 de Fevereiro de 2019

# Conteúdo

**1 Introdução** **iii**

**2 Conceitos Teóricos** **iv**
2.1 Agente SNMP . . . . . iv
2.2 Docker . . . . . iv

**3 Solução** **v**

**4 Implementação** **vi**
4.1 Fase A - Definição Containership MIB . . . . . vi
4.2 Fase B - Implementação containershipParam . . . . . vii
4.3 Fase C - Implementação containershipImageTable . . . . . viii
4.4 Fase D - Implementação containershipContainerTable . . . . . ix
4.5 Fase E - Implementação da mudança de estado . . . . . x
4.6 Fase F - implementação containershipStatus . . . . . xi
4.7 Fase G - Implementação de medidas de segurança . . . . . xi

**5 Manual de Utilização** **xii**

**6 Resultados e Discussão** **xii**

**7 Conclusão** **xiii**

# Lista de Figuras


1 Menu interativo da aplicação gestora . . . . . v
2 Containership MIB . . . . . vii
3 Valores Iniciais da MIB . . . . . viii
4 Escolher container para carregar na MIB . . . . . viii
5 Valores finais na MIB . . . . . viii
6 Resultado de um *snmpwalk* após a fase C . . . . . ix
7 Menu após criação de container . . . . . ix
8 Resultado de um *snmpwalk* após a fase D . . . . . x
9 Resultado da execução do comando *docker ps -a* . . . . . x
10 Mensagem apresentada na recusa de pedido pelo facto do container ter o seu estado em mudança . . . . . x
11 Resultado da execução de um comando *snmpwalk* após a fase F . . . . . xi

# 1 Introdução

A resolução deste trabalho foi proposta no âmbito da Unidade Curricular de Gestão de Redes, inserida no perfil de especialização de Gestão e Virtualização de Redes lecionado no primeiro semestre do quarto ano do Mestrado Integrado de Engenharia Informática.

A realização desta tarefa tem como intuito:

- A consolidação da utilização prática do modelo de gestão preconizado pelo *Internet-standard Network Management Framework*(INMF), dando especial relevo ao *Simple Network Management Protocol*(SNMP) e às *Management Information Bases*(MIBs).
- A utilização de APIs SNMP para construção de ferramentas de gestão (agentes e gestores).
- A investigação da aplicação do SNMP em sistemas de gestão nos mais variados ramos da engenharia aplicacional.

Neste trabalho pretende-se o desenvolvimento de um agente SNMP que seja um servidor de criação e gestão de containers. Este tipo de serviço remoto através da internet estaria assim disponível para que outros sistemas pudessem criar e gerir containers, pudendo estes, mais tarde, vir a ser usados por outros utilizadores. O serviço criado deverá também permitir a um administrador criar containers a partir de um conjunto de imagens pré-determinadas e consultar o estado dos containers existentes usando o SNMP.

De modo a satisfazer todos os requisitos especificados e os objetivos mencionados, o relatório apresenta a seguinte estrutura de exposição do trabalho realizado:

- Conceitos Teóricos, onde serão, resumidamente, abordados certos conceitos necessários à melhor compreensão do trabalho desenvolvido.
- Solução, onde será explicada a solução para a elaboração da ferramenta proposta.
- Implementação, onde serão explicitadas todas as fases de desenvolvimento da ferramenta.
- Manual de utilização, que contém as diretrizes para facilitar a utilização da ferramenta.
- Resultados e Discussão, onde constam os resultados obtidos pelo grupo bem como a sua apreciação
- Conclusão

# 2 Conceitos Teóricos

Como já foi mencionado anteriormente, esta secção tem o intuito de explicar certos conceitos que são relevantes para a melhor perceção do trabalho realizado. No contexto deste trabalho, o grupo decidiu oportuno abordar os conceitos que se seguem.

## 2.1 Agente SNMP

Um agente SNMP é um programa integrado nos elementos de uma rede. A ativação deste agente permite que este recolha a informação de gerência dos dispositivos localmente, tornando esta informação disponível para o gerente SNMP sempre que for requisitado.

As principais funções de um agente SNMP são:

- Recolher informações de gerência acerca do seu ambiente local.
- Armazenar e recuperar informações de gerência tal como estão definidas nas MIBs
- Sinalizar eventos ao gestor.

## 2.2 Docker

O Docker é um programa que desempenha uma virtualização a nível do sistema operativo. O Docker é usado para executar pacotes de software designados*containers*. Os *containers* são isolados uns dos outros em que cada um contém sua própria aplicação, ferramentas, librarias e ficheiros de configuração.

Um *container* é uma unidade standard de software que integra código e todas as suas dependências, para que uma aplicação funcione rapidamente e confiavelmente de um ambiente de computação para outro. Um Docker *container* é um pacote de software leve, independente e executável que contém tudo o que é necessário para correr uma aplicação.

Todos os *containers* são executados pelo mesmo kernel do sistema operativo e por isso são mais leves do que as máquinas virtuais. Os *containers* são criados a partir de "imagens"que especificam, precisamente, o seu conteúdo.

Graças a este sistema de *containers*, os programadores estão seguros de que a aplicação irá ser executada em qualquer outra máquina, independentemente das configurações personalizadas que essa máquina pode ter e que a difiram da máquina que foi usada, originalmente, para escrever e testar o código.

# 3 Solução

A solução criada pelo grupo passou pela criação de um agente *SNMP* utilizando a biblioteca *SNMP4J*, que também integra uma pequena aplicação gestora para efeitos de demonstração, que visa substituir a necessidade de executar as operações todas sempre com comandos *SNMP*, apesar de que o agente continua a ser acessível com as ferramentas de linhas de comando do *NET-SNMP* por exemplo.Uma imagem da aplicação gestora que permite controlar a *MIB* presente no agente pode ser visualizada na figura 1.

O ponto de entrada do agente é a classe *SNMPAgentStarter* que lê o ficheiro de configuração de forma a perceber em que porta escuta o agente, a community string a utilizar bem como as imagens que ele deve suportar. De seguida, este programa cria uma *Thread* que será a aplicação gestora (num cenário real, deveria ser outro programa a tentar comunicar com o servidor que contém o agente). Além disso, inicia também a classe *SNMPAgentFunctions*, que implementa a MIB definida e as operações necessárias neste contexto (carregamento de parâmetros, criação de containers, iniciar e parar os mesmos bem como removê-los e listá-los).

Além disso a solução desenvolvida também implementa uma metodologia de mudança de estado, sendo que qualquer mudança de estado terá que necessariamente passar pelo estado **changing**, de forma a prevenir incorrecções durante múltiplos acessos a um parâmetro da MIB. Se um dado container se encontra nesse estado, não devem ser executadas mais nenhumas operações sobre o mesmo, enquanto que não chegar a um estado final (up ou down). Nesta solução, é também definida uma medida de segurança funcional com o objetivo de limitar o número de pedidos por minuto que o servidor aceita e, neste sentido, o grupo decidiu que o servidor não deverá aceitar mais de 120 pedidos por minuto.

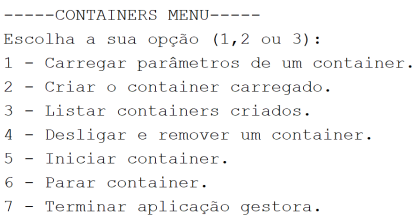


Figura 1: Menu interativo da aplicação gestora

# 4 Implementação

## 4.1 Fase A - Definição Containership MIB

Nesta fase o grupo começou por definir a MIB que viria a ser implementada no agente SNMP. A MIB é composta pelos seguintes parâmetro:

- *containershipParam* - parâmetro com objetos que representam um *container* novo a ser criado. Este parâmetro contém 4 objetos: o nome da imagem do docker tal como existe no repositório (indexParam - *DisplayString*), um nome opcional mais descritivo da imagem (nomeParam - *DisplayString*), o indice da imagem na tabela de imagens (indexIParam - *Integer32*) e uma flag a indicar se este deve ou não ser criado (flagParam - *Integer32, 0 ou 1*).

- *containershipImageTable* - Tabela das imagens disponíveis. Cada entrada desta tabela possui dois objetos: o indice da imagem (indexImage - *Integer32*) e o seu nome (image - *DisplayString*).

- *containershipContainerTable* - parâmetro com a tabela de containers que já foram criados. Cada entrada desta tabela contém 6 colunas: o índice do *container* (indexcontainer - *Integer32*), o seu nome (container_name - *DisplayString*), o índice da imagem que o originou (image_index - *Integer32*), o seu estado (status - *Integer32. 1 (creating) ,2 (changing) ,3 (up) ,4 (down) ou 5 (removing))*, a percentagem de utilização do processador desse *container* (processador - *DisplayString*) e o id do *container* (containerID - *DisplayString*).

- *containershipStatus* - parâmetro com 2 objetos escalares, sendo os mesmos a data de inicialização do agente (data - *DisplayString*) e um contador com o número de containers criados (contador - *Integer32*).

Tendo estas especificações todas em conta, o grupo chegou à seguinte estrutura da MIB a implementar.

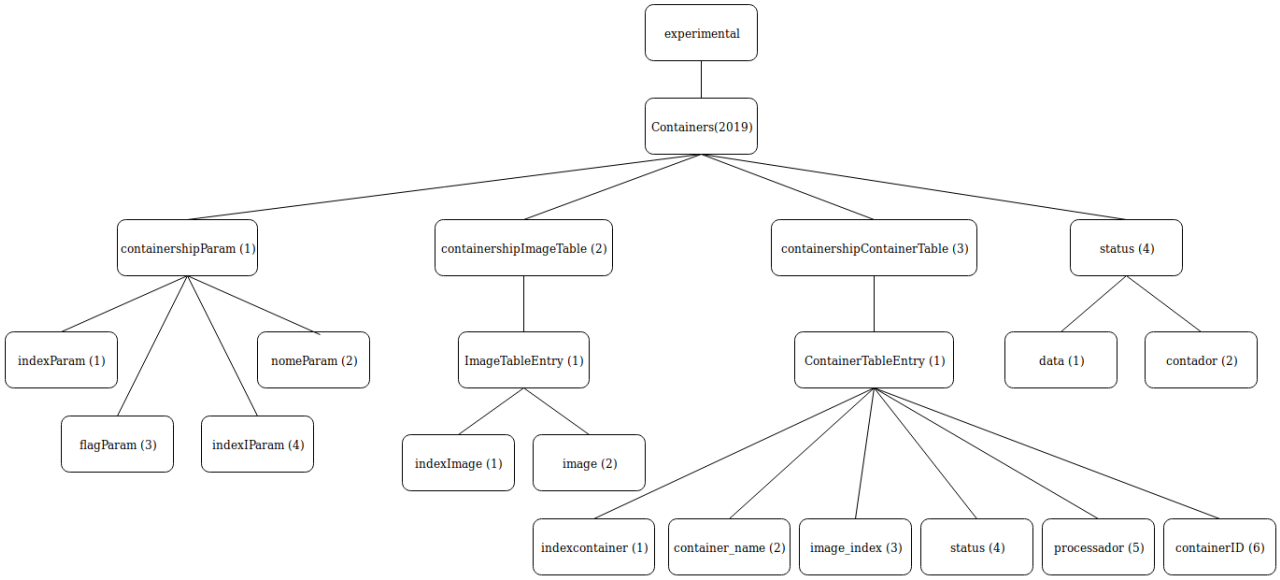


Figura 2: Containership MIB

## 4.2 Fase B - Implementação containershipParam

Nesta fase era necessário construir um agente SNMP capaz de ler o ficheiro de configuração e implementar o grupo *containershipParam(1)* da *ContainerShip MIB*.

Inicialmente começamos por fazer a construção do nosso agente SNMP, consultando a porta que é carregada no ínicio do programa e recorrendo à biblioteca *snmp4j*.

```
agent = new SNMPAgent("127.0.0.1/" + agente.getPorta());
agent.start();
```

Depois de construir o nosso agente SNMP é necessário carregar a informação da imagem que o utilizador escolher. Uma vez que, ao iniciar o programa, as imagens lidas do ficheiro de configuração são carregadas para a *containershipImageTable(2)* as opções mostradas ao utilizador são feitas fazendo um get aos valores contidos em *containershipImageTable(2)*.

Inicialmente o valor de *nameParam(1.2)* é **"NONE"** de maneira ao nosso programa poder saber se já existe algum container carregado em *containerShipParam(1)*, o valor da flag é 0 e o valor de *indexIparam(1.4)* também é 0.

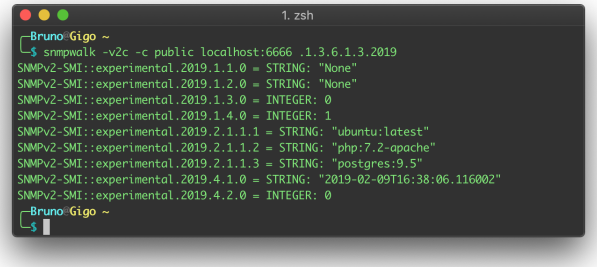


Figura 3: Valores Iniciais da MIB

Quando o utilizador escolhe a imagem que pretende carregar em *containerShipParam(1)* os valores são alterados para os respetivos valores referentes à imagem escolhida.

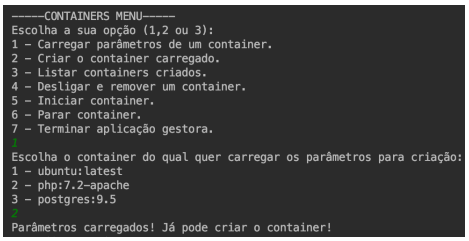


Figura 4: Escolher container para carregar na MIB

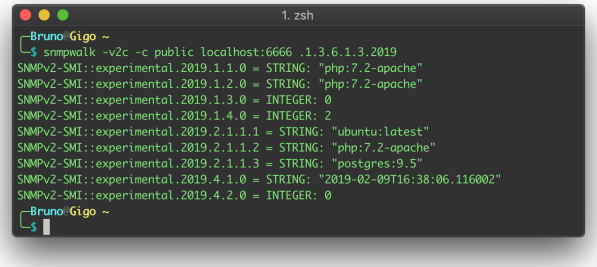


Figura 5: Valores finais na MIB

Posteriormente quando o container for criado estes valores levam um *reset* e voltam ao estado inicial. De referir também que, quando já existe um container carregado em *containerShipContainer(1)*, o programa não deixa o utilizador carregar um novo container.

### 4.3 Fase C - Implementação containershipImageTable

O objetivo desta fase passava por implementar a tabela de imagens disponíveis no agente, permitindo que estas fossem lidas por forma a executar comandos. Assim, foi implementada no agente uma tabela de imagens, em que os seus elementos são apenas *read-only*, ou seja, não é permitido alterar os elementos criados. Após a definição da tabela propriamente dita,

as imagens previamente lidas do ficheiro de configuração foram carregadas para a tabela. A correta implementação desta tabela pode ser vista na figura 6, onde após executar um comando *snmpwalk* podemos verificar que os objetos implementados na fase B e nesta fase já se encontram carregados. É necessário ter em conta que os objetos escalares implementados na fase B ainda se encontram na forma inicial, ou seja, sem nenhum container carregado.

```
λ snmpwalk -v2c -c public localhost:6666 1.3.6.1.3.2019
SNMPv2-SMI::experimental.2019.1.1.0 = STRING: "None"
SNMPv2-SMI::experimental.2019.1.2.0 = STRING: "None"
SNMPv2-SMI::experimental.2019.1.3.0 = INTEGER: 0
SNMPv2-SMI::experimental.2019.1.4.0 = INTEGER: 1
SNMPv2-SMI::experimental.2019.2.1.1.1 = STRING: "ubuntu:latest"
SNMPv2-SMI::experimental.2019.2.1.1.2 = STRING: "php:7.2-apache"
SNMPv2-SMI::experimental.2019.2.1.1.3 = STRING: "postgres:9.5"
```

Figura 6: Resultado de um *snmpwalk* após a fase C

## 4.4 Fase D - Implementação containershipContainerTable

O objetivo desta fase passava por implementar a tabela de containers criados no agentes, permitindo que aos mesmos pudessem ser aplicadas operações para o inicializar ou parar por exemplo. Esta tabela deverá conter todos os containers que são criados pelo utilizador da aplicação gestora, exceto aqueles que já foram removidos. Cada entrada desta tabela tem os 6 objetos já referidos na definição da MIB sendo que apenas o *status* é *read-write* e os restantes objetos são *read-only*. A correta implementação desta tabela pode ser vista na figura 8, onde após executar um comando *snmpwalk* podemos verificar que os objetos implementados na fase B, fase C e nesta fase já se encontram carregados, sendo que, nesta altura um container já foi criado para efeitos de demonstração. Como é possível ver na figura 9, o container também é criado no docker. Na figura 7 é possível visualizar a correta criação do container a partir do menu.

```
-----CONTAINERS MENU-----
Escolha a sua opção (1,2 ou 3):
1 - Carregar parâmetros de um container.
2 - Criar o container carregado.
3 - Listar containers criados.
4 - Desligar e remover um container.
5 - Iniciar container.
6 - Parar container.
7 - Terminar aplicação gestora.
2
4e830ba132f84eb604fce9006ea6af10f79143b5bb0763b218485816706e10c4
Identificador do container criado: 4e830ba132f84eb604fce9006ea6af10f79143b5bb0763b218485816706e10c4
```

Figura 7: Menu após criação de container

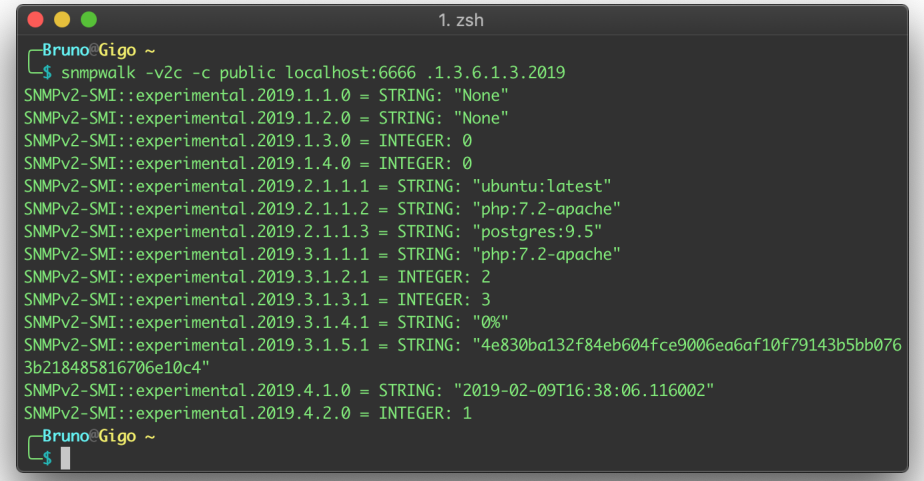


Figura 8: Resultado de um *snmpwalk* após a fase D

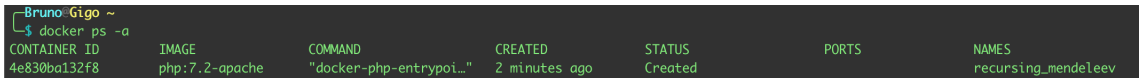


Figura 9: Resultado da execução do comando *docker ps -a*

## 4.5 Fase E - Implementação da mudança de estado

Nesta fase, o objetivo era implementar um mecanismo que permitisse sinalizar a mudança de estado, ou seja sempre que um container tivesse que ser iniciado, parado ou removido, o estado do mesmo na tabela deveria ser primeiramente alterado para o estado 2 (changing), para que não existissem alterações concorrentes ao mesmo container. Assim, sempre que é pedida uma operação de remoção, iniciação ou paragem de um dado container, o agente verifica se o estado do mesmo está a *changing* e, se estiver, não executa o pedido. No caso de não estar, o estado desse container é alterado para *changing*, são executadas as operações necessárias para realizar o pedido feito e, finalmente, o container é alterado para o estado correspondente ao pedido feito.

Na figura 10, é possível visualizar a mensagem recebida após tentar a remoção de um container que se encontra no estado changing, não executando a operação referida.

```
1
Este container está a ser alterado atualmente! Tente novamente mais tarde!
-----CONTAINERS MENU-----
```

Figura 10: Mensagem apresentada na recusa de pedido pelo facto do container ter o seu estado em mudança

## 4.6 Fase F - implementação containershipStatus

O objetivo desta fase passava por implementar o objeto *containershipStatus*, que irá conter dois escalares representando a data de início do agente e um número inteiro correspondente ao número de containers criados via pedidos feitos ao agente. A correta implementação deste objeto, bem como um exemplo de valores do mesmo podem ser vistos na figura 11, que aparecem após a execução de um comando *snmpwalk*. A execução deste comando foi feita após a criação de um container. Deve ser denotado que o containerID ainda está a changing visto que ainda não foi recebido o ID do docker e, nesse sentido, o ID ainda não foi atualizado.

```
λ snmpwalk -v2c -c public localhost:6666 1.3.6.1.3.2019
SNMPv2-SMI::experimental.2019.1.1.0 = STRING: "None"
SNMPv2-SMI::experimental.2019.1.2.0 = STRING: "None"
SNMPv2-SMI::experimental.2019.1.3.0 = INTEGER: 0
SNMPv2-SMI::experimental.2019.1.4.0 = INTEGER: 0
SNMPv2-SMI::experimental.2019.2.1.1.1 = STRING: "ubuntu:latest"
SNMPv2-SMI::experimental.2019.2.1.1.2 = STRING: "php:7.2-apache"
SNMPv2-SMI::experimental.2019.2.1.1.3 = STRING: "postgres:9.5"
SNMPv2-SMI::experimental.2019.3.1.1.1 = STRING: "ubuntu:latest"
SNMPv2-SMI::experimental.2019.3.1.2.1 = INTEGER: 1
SNMPv2-SMI::experimental.2019.3.1.3.1 = INTEGER: 2
SNMPv2-SMI::experimental.2019.3.1.4.1 = STRING: "0%"
SNMPv2-SMI::experimental.2019.3.1.5.1 = STRING: "changing"
SNMPv2-SMI::experimental.2019.4.1.0 = STRING: "2019-02-09T17:13:40.121774100"
SNMPv2-SMI::experimental.2019.4.2.0 = INTEGER: 1
```

Figura 11: Resultado da execução de um comando *snmpwalk* após a fase F

## 4.7 Fase G - Implementação de medidas de segurança

Nesta fase, o objetivo passava por discutir o tipo de quebras de segurança que um servidor deste género pode sofrer e implementar uma ou várias medidas que permitissem mitigar este facto. Após uma discussão entre o grupo, resolvemos que a melhor medida a aplicar seria limitar o número de pedidos que o servidor deveria aceitar/executar por minuto e, assim sendo, definimos que um servidor deste tipo não deveria aceitar mais do que 120 pedidos por minuto (2 pedidos por segundo), visto que esta medida também permite menos colisões nos pedidos dos utilizadores em termos de mudanças de estado por exemplo.

A implementação desta medida é difícil de demonstrar, visto que não temos acesso a vários sistemas ao mesmo tempo para conseguir executar mais do que 120 pedidos durante um minuto.

Esta verificação é feita no início de execução de qualquer uma das operações definidas no agente, pelo que se a verificação falha, a operação termina imediatamente, tendo que o utilizador esperar pelo próximo minuto para satisfazer o seu pedido.

# 5 Manual de Utilização

Para executar este programa deverá realizar as seguintes ações:

- Executar o *SNMPAgentStarter*, utilizando o comando:

```
java -jar SNMPAgentStarter.jar
```

- Dentro do programa poderá:
  - Carregar os parâmetros de um *container*
  - Criar o *container* que foi carregado anteriormente
  - Listar os *containers* criados
  - Desligar e remover um *container*
  - Iniciar um *container*
  - Parar um *container*

# 6 Resultados e Discussão

Nesta secção pretendemos discutir os resultados que obtemos, abordando a sua correcção e possíveis melhoramentos que deviam existir.

Apesar da aplicação gestora construída e inserida junto com o agente criado fornecer todas as operações necessárias num servidor de monitorização e criação de containers, o grupo tem a opinião que, possivelmente, este servidor poderia ter sido implementado de forma diferente, tal que aceitasse apenas pedidos via comandos *SNMP* e escutasse por alterações de estado na MIB. A aplicação construída provocou mais dificuldades na sua implementação e, apesar da correcção em termos do que deve ser fornecido como possibilidade ao utilizador, não está construída de forma a suportar apenas comandos SNMP, apesar de ser compatível com os mesmos.

Um ponto que o grupo acha que foi muito bem conseguido é o facto de existir integração com o *Docker*, ou seja, a criação, remoção, iniciação e paragem de containers reflete-se no próprio *Docker*, permitindo executar os containers necessários para um dado utilizador.

# 7 Conclusão

Analisando os resultados previamente apresentados, o grupo encontra-se bastante satisfeito com a ferramenta elaborada, visto que foi conseguida a implementação de todas as fases propostas. O sucesso na implementação das mesmas permitiu que a ferramenta produzida fosse capaz de criar e gerir *containers* através do *Docker*, que era o objetivo principal deste trabalho.

Apesar disto, o grupo sente que poderia melhorar a ferramenta, futuramente, no que diz respeito à maior utilização de comandos *SNMP* para a execução das operações necessárias, visto que uma das finalidades do projeto incidia numa maior familiarização com este protocolo.

No que diz respeito à passagem da solução teórica para a sua implementação prática, o grupo considera que esta foi bem conseguida pois apesar de certos pequenos contratempos que surgiram, o resultado final foi bastante positivo.