

UNIVERSIDADE DE AVEIRO

DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICAÇÕES E
INFORMÁTICA

GESTÃO INTEGRADA DE REDES E SISTEMAS

Docker Data Center

**Relatório Virtualização, Armazenamento e
Monitorização**

8240 - MESTRADO INTEGRADO EM ENGENHARIA DE
COMPUTADORES E TELEMÁTICA

António Rafael da
Costa Ferreira
NMec: 67405

Rodrigo Lopes
da Cunha
NMec: 67800

Docentes: Diogo Gomes,
João Paulo Barraca

Abril de 2016
2015-2016

Conteúdos

1	Introdução	2
2	Virtualização	3
2.1	Docker	3
2.2	Rancher	5
3	Armazenamento	8
3.1	RAID	8
3.2	NAS/SAN	9
3.3	Rancher Storage Pool	12
4	Monitorização	15
4.1	Centreon	15
5	Datadog	20
6	Load balancer	22
6.1	Nginx como Load Balancer	23
6.2	Load balancing tendo em conta a carga do CPU dos containers	26
7	Conclusão	36

1 Introdução

O relatório reflete todos os passos no desenvolvimento das diferentes componentes, virtualização, armazenamento e monitorização, presentes no Docker Data Center, que está a ser desenvolvido no âmbito da disciplina de Gestão Integrada de Redes e Sistemas.

2 Virtualização

Após a apresentação sobre as várias soluções de virtualização foi dada a possibilidade de se escolher qual solução que se iria explorar durante o semestre.

Foi escolhido o Docker por ser uma tecnologia bastante recente e com uma comunidade bastante ativa.

O objetivo será usando o Docker, simular uma infraestrutura virtual para um data center.

2.1 Docker

No Docker, os containers correm numa única máquina e partilham o mesmo kernel do sistema operativo de forma a que iniciem imediatamente e torne o uso da RAM mais eficiente.

As imagens são construídas com base em camadas de sistema de ficheiros para que seja possível partilhar ficheiros comuns, tornando o uso do disco e o download de imagens mais eficiente.

Os containers são baseados em "open standards" permitindo que os containers corram em praticamente todas as distribuições Linux e sistemas operativos Microsoft com suporte para todas as infraestruturas. Estes isolam as aplicações uns dos outros e a infraestrutura subjacente, proporcionando uma camada adicional de proteção para a aplicação.

Comparação com Máquinas Virtuais

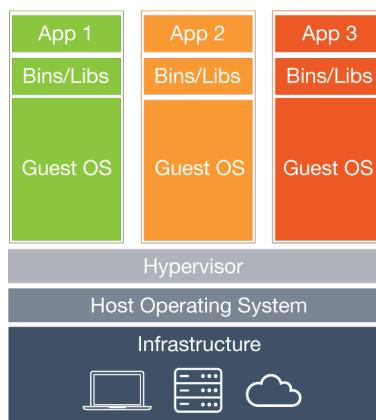


Figura 1: Diagrama docker

Cada máquina virtual inclui a aplicação, os binários necessários, as bibliotecas e um sistema operativo "guest" inteiro - todos podem ser dezenas de GBs.

Já os containers incluem a aplicação e todas as suas dependências, mas partilham o kernel com outros containers. Eles correm em processos isolados no espaço do utilizador (userspace) no sistema operativo do host. Eles não estão limitados a uma infraestrutura específica, correm em - containers Docker podem correr em qualquer computador ou numa infraestrutura na cloud.

DockerFile > Image > Container

Uma imagem docker contém todos os pacotes necessários pré-instalados o que é diferente de uma imagem de uma máquina Ubuntu.

Quando a imagem está pronta é possível correr a mesma, tendo assim um container.

O bom da partilha de imagens é que, por exemplo, uma equipa pode criar o seu ambiente de desenvolvimento, com a mesma versão do Python, o mesmo Django e o mesmo PyCharm com todas as configurações pré-configuradas.

Limitações

Limite de CPU

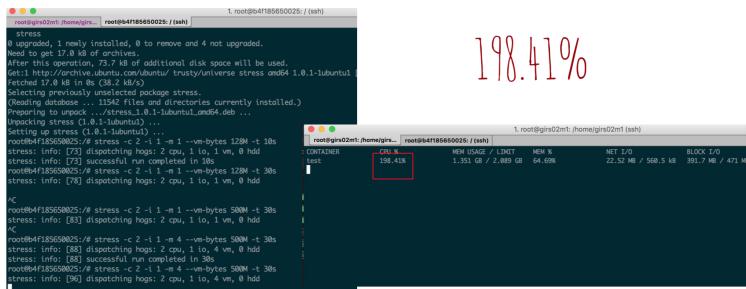


Figura 2: Limite de CPU 198%

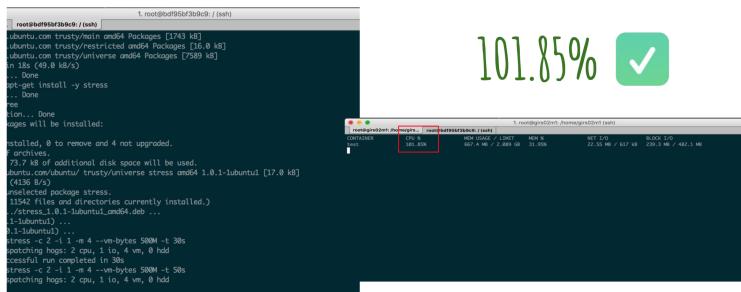


Figura 3: Limite de CPU 101%

Na primeira figura, não tem limite de CPU, por isso enchemos 198.41% do

Rafael Ferreira nmec: 67405

Rodrigo Cunha nmecc: 67800

CPU. Na segunda figura, com o limite de CPU, enchemos apenas 101.85%, sendo assim, o limite de CPU funciona.

Límite de RAM

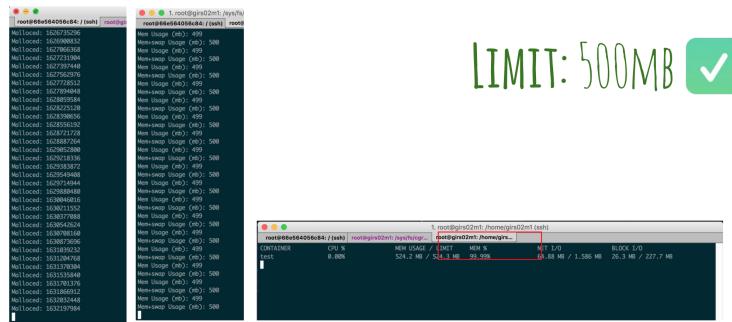


Figura 4: 500 RAM

Com o limite de memória RAM, também foi possível 500 MB.
GeekBench

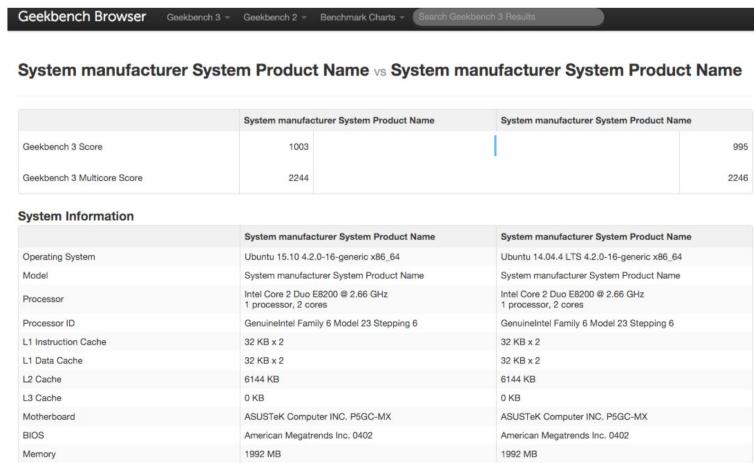


Figura 5: GeekBench, Host vs Container

A comparação entre os resultados do container e do host são praticamente idênticos.

2.2 Rancher

Rancher é uma plataforma para gestão de vários nós que têm um docker machine. Dá a possibilidade de adicionar nós que estejam alojados ou localmente, ou na Amazon Web Services, ou na DigitalOcean, entre outras.

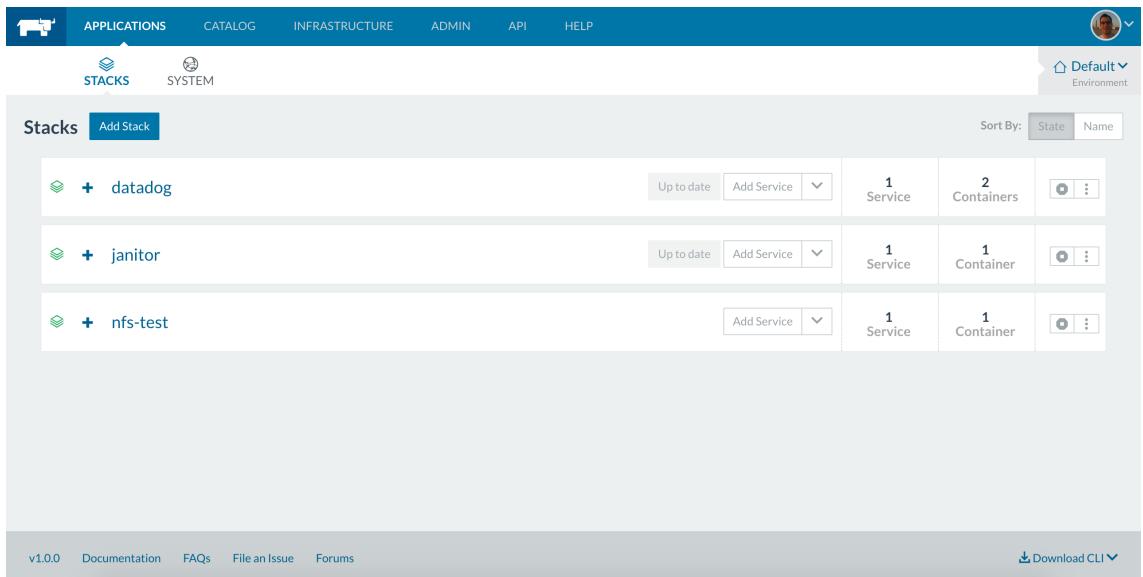


Figura 6: Página Inicial do Rancher

A configuração do Rancher é muito simples. Escolhe-se uma máquina que queremos que seja o nó principal que gere todos os outros nós, sendo que este deve ter a versão mais recente do Docker instalada.

Este nó será responsável por gerir outros nós que sejam adicionados à rede do Rancher, pode ser adicionado nós dentro de um Datacenter ou em Hosting Providers externos como o DigitalOcean ou a AWS.

Para autenticação no Rancher, está disponível várias hipóteses, GitHub, LDAP ou Basic Authentication.

Na página inicial, é mostrada ao utilizador a lista de Stacks, sendo que estas devem-se organizar por serviço/aplicação.

Figura 7: Access control with GitHub

Figura 8: Admin accounts

Ainda no painel de administração, pode-se ainda ver os processos que estão a correr, ver um log de ações na administração e ainda mudar algumas configurações.

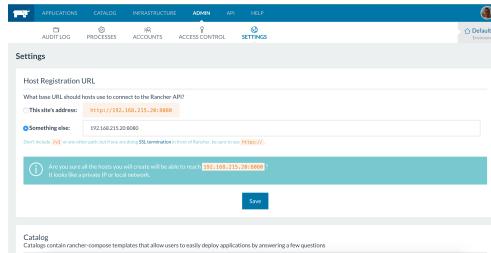


Figura 9: Definições

Audit Log						
Time	Event Type	Description	Environment	Resource Type	Identity	Auth & IP
2 hours ago	service:reconcile	Waiting for Environment: deleting agent_31 instance:Starting	Default	service:3d3		
2 hours ago	service:reconcile	Waiting for Environment: deleting agent_31 instance:Starting	Default	service:3d3		
2 hours ago	service:reconcile	Waiting for Environment: deleting agent_31 instance:Starting	Default	service:3d4		
2 hours ago	service:reconcile	Waiting for Environment: deleting agent_31 instance:Starting	Default	service:3d5		
2 hours ago	service:reconcile	Waiting for Environment: deleting agent_31 instance:Starting	Default	service:3d6		
2 hours ago	service:reconcile	Waiting for Environment: deleting agent_31 instance:Starting	Default	service:3d7		

Figura 10: Log de ações

O Rancher também tem uma API para o exterior, pelo que é dito na documentação, as features são primeiro desenvolvidas para a API e só depois aparecem na interface gráfica. São também um meio de comunicação para programas que podem ser desenvolvidos posteriormente com base na API do Rancher.

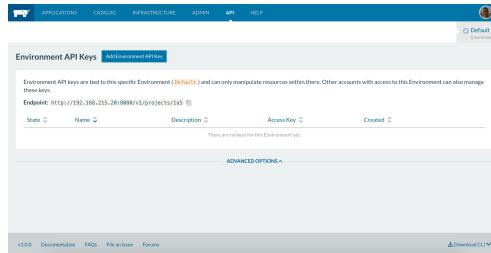


Figura 11: API Rancher



Figura 12: API Container

No painel da infraestrutura é possível ver os Hosts adicionados, containers ativo nos vários Hosts, Storage pools adicionados, certificados e Private Registries.

Podem ser adicionados Hosts do data center, locais ou de data centers como DigitalOcean, AWS entre outros.

Já os storage pools é possível adicionar storage com NFS.

Para gravar imagens locais, privadas, normalmente criam-se Private Registries.

State	Name	IP Address	Host	Image	Command
Offlining	convoy-nfs_1	10.0.2.156	girs02m1	rancher/nomad agent v0.7.0	volume agent nfs
Offlining	convoy-nfs_storagedpool_1	10.0.2.150	girs02m1	rancher/nomad agent v0.7.0	storagepool agent
Offlining	datadog_datadog-agent_1	10.0.2.129	girs02m1	datadog/docker-dd-agent:1.0.0.0	supervisord -c /etc/dd-agent/conf.d
Started Once	datadog_datadog-agent_datadog_1	None	girs02m1	datadog/docker-dd-agent:1.0.0.0	processes check latest
Offlining	janitor_cleanup_1	10.0.2.129	girs02m1	unihex/docker-cleanups:1.5.2	None
Offlining	naughty_nominet_1	10.0.2.170	girs02m1	rancher/nomad	Any Docker service, Nomad
Offlining	Network Agent	10.0.2.149.98	girs02m1	rancher/nomad instance v0.8.1	None
Offlining	ubutu-test_docker_1	10.0.2.151	girs02m1	ubuntuf14.0.4	drush dbstatus

Figura 13: Containers

Active	convoy-nfs
Hosts	girs02m1
Volumes	girs02m1
Mounts (Container Path)	/var/lib/docker/volumes/girs02m1/_data

Figura 14: Storage pools

Figura 15: Hosts

3 Armazenamento

3.1 RAID

Criação de um RAID

Configuração

Para se avaliar os diferentes níveis de RAID usou-se o OpenMediaVault que é um sistema operativo para gestão de um NAS baseado em Debian (Linux).

Para instalação do ambiente foi criado uma máquina virtual usando a Virtual Box com 4 discos com 1024MB.

Depois foram criados diferentes níveis de RAID, sendo estes o nível 0, 1, 5, 6 e 10. Observou-se para os diferentes níveis de RAID, qual é o espaço

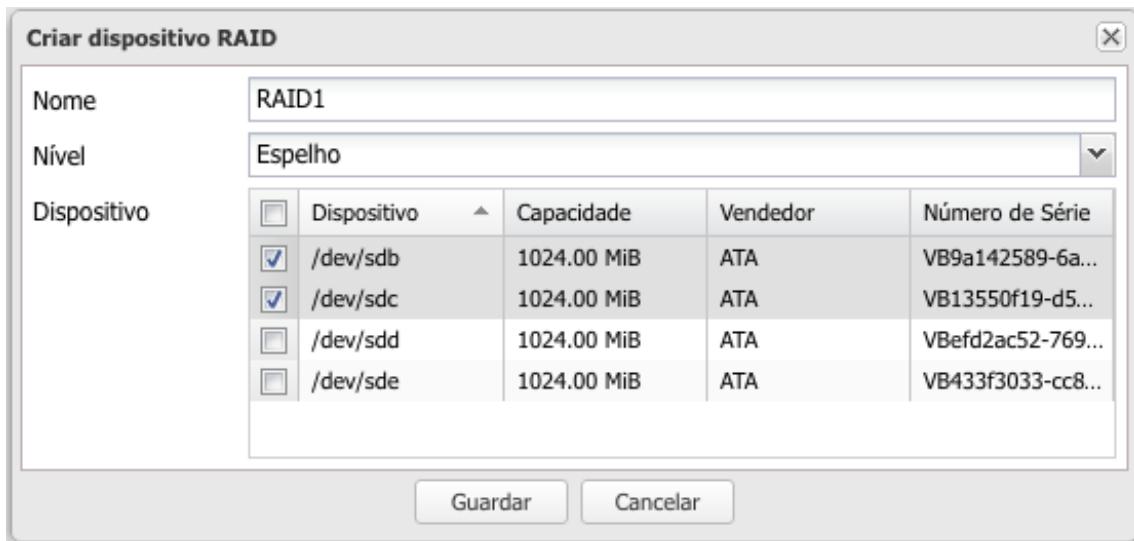


Figura 16: RAID 1

disponível e removendo os discos, qual é o número de falhas que podem existir para que os dados não sejam perdidos.

Tabela 1: Avaliação dos diferentes níveis de RAID

RAID	Número de discos	Espaço disponível	Número de falhas permitidas
0	2	2 GB	0
1	2	1023.44 MB	1 (N-1)
10	4	2 GB	2 (N/2)
5	3	2 GB	1
6	4	2 GB	1

3.2 NAS/SAN

Configuração do NAS

Na segunda parte do trabalho foi pedido que fosse realizado uma NAS/SAN, para isso começou-se por criar um RAID 0 e criou-se um volume com o RAID 0 criado com o nome "md1" com o sistema de ficheiros "ext3".

Girs:RAID0	/dev/md1	clean	Stripe	2.00 GiB	/dev/sdb /dev/sdc
------------	----------	-------	--------	----------	----------------------

Figura 17: RAID 0

Dispositivo	Rotulo	Ficheiro de siste...	Total	Disponivel	Utilizado	Montado	Referenciado	Estado
/dev/md0	Share	ext4	1.97 GiB	1.93 GiB	38.01 MiB	Sim	Sim	Ligado
/dev/md1	Share2	ext3	1.97 GiB	1.93 GiB	35.03 MiB	Sim	Não	Ligado
/dev/sda1		ext4	912.40 MiB	21.55 MiB	844.49 MiB	Sim	Sim	Ligado

Figura 18: Volume "md1"

Usando o Samba fez-se partilha do novo Volume criado como mostra a figura 19.

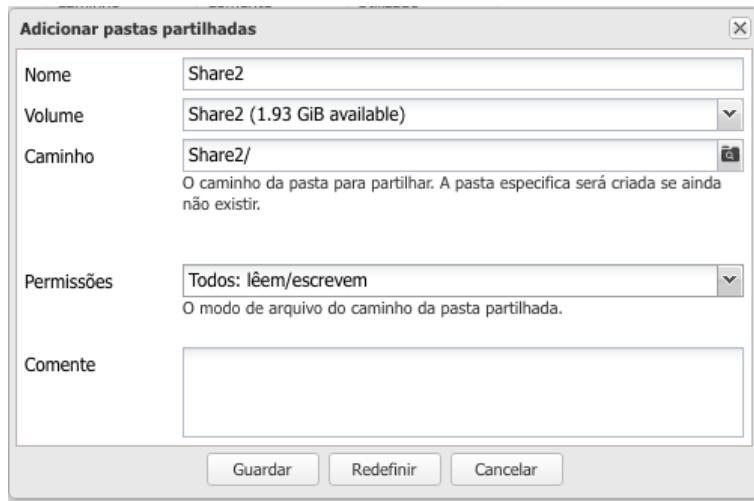


Figura 19: Adicionar o Volume para partilha

Depois montou-se a pasta partilhada como sendo um NAS num OS X como demonstra a figura 20.

Teste do NAS

Para testar se realmente estaria ou não a funcionar o NAS criado, criou-se um ficheiro na máquina virtual como demonstra a figura 21.

Após isso, verificou-se se no OS X o ficheiro também já existia, e de facto, existia, como demonstra a figura ??.

Benchmark do NAS Volume

Usando o "bonnie++" efetuaram-se 10 medições e calculou-se a média para obter resultados finais para a experiência, como mostram as tabelas 2, 3 e 4.

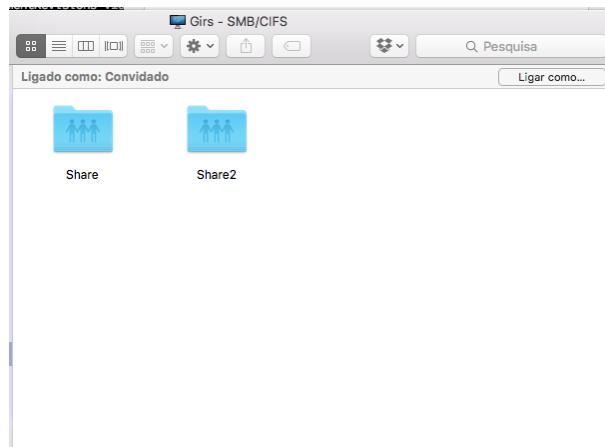


Figura 20: Pastas partilhadas

```
root@Girs:/media/64faae79-a270-490f-955d-ba7e8cca4268/Share2# ls
teste.txt
```

Figura 21: Adicionar o Volume para partilha



Figura 22: Ficheiro existente no OS X

Tabela 2: Resultados do benchmark usando bonnie++ (Parte 1)

Bonnie++ Version	Concurrency Level	Random Number Seed	File Size	Block Writes (K/s)	(% cpu)
1.97	1	1459450816	8G	88149	10
1.97	1	1459453831	8G	72789	6
1.97	1	1459452397	8G	73546	7
1.97	1	1459452673	8G	75701	
1.97	1	1459447167	8G	88738	9
1.97	1	1459446918	8G	72541	7
1.97	1	1459445918	8G	78299	7
1.97	1	1459449549	8G	75141	7
1.97	1	1459449100	8G	75521	7
1.97	1	1459449032	8G	80327	8
média	1.97	1459449740	8G	78075,2	7,5

No final concluiu-se que para o mesmo nível de concorrência e para a escrita e leitura de um ficheiro com o tamanho de 8GB que:

Tabela 3: Resultados do benchmark usando bonnie++ (Parte 2)

Reading/Rewriting Block (K/s)	(% cpu)	Block Reads (K/s)	(% cpu)	Seek (s/s)	(% cpu)
21758	6	47470	5	1000	31
32983	5	77413	8	940,2	27
38005	6	77339	7	1103	35
30252	6	71017	7	1060	35
32433	10	73999	7	1055	33
36601	6	74004	7	1004	32
32597	6	66784	7	1093	34
31529	6	71241	7	907,1	29
33710	6	74052	7	994,8	31
32089	5	73297	7	973,5	31
média	32195,7	6,2	70661,6	6,9	1013,06

Tabela 4: Resultados do benchmark usando bonnie++ (Parte 3)

Block Write Latency (ms)	Rewrite Latency (ms)	Block Read Latency (ms)	Seek Latency (ms)
4009	11369	11902	175
27544	11565	10906	3186
5024	25956	11947	1697
5512	10859	9862	168
4185	21276	10784	2173
4909	12442	11381	2292
5541	10699	11237	178
6399	11024	11982	169
5440	11958	13269	171
4428	34282	11390	182
média	7299,1	16143	11466
			1039,1

Tabela 5: Resultados finais da benchmark bonnie++

Block Writes (K/s)	78075,2	Seek (s/s)	1013,06
(% cpu)	7,5	(% cpu)	31,8
Reading/Rewriting Block (K/s)	32195,7	Block Write Latency (ms)	7299,1
(% cpu)	6,2	Rewrite Latency (ms)	16143
Block Reads (K/s)	70661,6	Block Read Latency (ms)	11466
(% cpu)	6,9	Seek Latency (ms)	1039,1

Esta tabela permitiu que se percebesse melhor os sistemas de RAID existentes. Entre eles existe, por exemplo o RAID 0, que é o ideal quando se procura uma melhor performance. Existe ainda para uma maior capacidade de armazenamento o JBOD e para melhor redundância, com melhor aproveitamento de discos, podendo falhar até dois, tem-se o RAID 6.

Foi ainda possível, através da benchmark Bonnie++, ter conhecimento acerca dos tempos de escrita e leitura numa pasta partilhada na rede pela NAS criada no trabalho.

3.3 Rancher Storage Pool

Usando uma máquina do laboratório, instalou-se uma imagem de Ubuntu Server e colocou-se NFS.

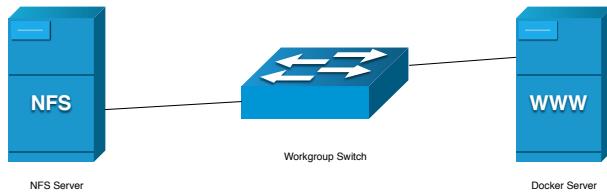


Figura 23: Arquitetura de servidores

Usando o Convoy NFS¹ foi montado o NFS do servidor paralelo no Rancher.

Depois de montado, o NFS, devem ser criados vários volumes que depois serão usados para os containers que serão instanciados. Esta solução permite que os ficheiros do container estejam num servidor à parte de storage, sendo mais fácil efetuar backups aos dados de todos os containers.

Os volumes criados só irão aparecer ativos quando forem usados num container.

Name	State	Name	Mounts (Container Path)
grind0nd	Active	volume2	/nfs/test_ubuntu_1:/volume

Figura 24: Convoy NFS Catalog

Figura 25: Volumes criados no Convoy NFS

Volume NFS num Container

Para testar a criação de um container com um Volume NFS, foi criado uma Stack "nfs-test" com um container com uma imagem "ubuntu:14.04.3" com um volume a fazer mount no diretório "/volume", apenas para efeito de teste.

Foi então criado um ficheiro "rafa" no diretório "/volume" com o conteúdo de "ola".

¹ <https://github.com/rancher/convoy>

> Shell: nfs-test_ubuntu_1

ProTip: Hold the Command key when opening shell access to launch a new window.

The screenshot shows a terminal window titled 'Shell: nfs-test_ubuntu_1'. The terminal displays a root shell session on a Docker container. The user runs several 'ls' commands to show the contents of the root directory and its subdirectories (bin, dev, home, lib64, mnt, proc, run, srv, var, boot, etc, lib, media, opt, root, sbin, sys, usr, volume). Then, the user navigates to the 'volume' directory ('cd volume') and creates a new file named 'rafa' ('touch rafa'). Finally, the user reads the contents of the 'rafa' file ('cat rafa'), which shows the word 'rafa'. A 'Close' button is visible in the bottom right corner of the terminal window.

```
root@c22fd1f6132b:~# ls
bin  dev  home  lib64  mnt  proc  run  srv  [redacted]  var
boot  etc  lib  media  opt  root  sbin  sys  usr  volume
root@c22fd1f6132b:~# cd media/
root@c22fd1f6132b:/media# ls
root@c22fd1f6132b:/media# media# cd ..
root@c22fd1f6132b:/# ls
bin  dev  home  lib64  mnt  proc  run  srv  [redacted]  var
boot  etc  lib  media  opt  root  sbin  sys  usr  volume
root@c22fd1f6132b:~# cd volume/
root@c22fd1f6132b:/volume# ls
rafa
root@c22fd1f6132b:/volume# cat rafa
ola
root@c22fd1f6132b:/volume#
```

Connected

Close

Figura 26: Criação de um ficheiro no diretório "volume"

Depois, para testar, acedeu-se à máquina NFS e listou-se o diretório partilhado com a máquina Docker, e observou-se que o ficheiro também tinha sido ali criado.

Figura 27: Máquina NFS

4 Monitorização

4.1 Centreon

Para a realização da parte de monitorização do projeto, foi utilizada a plataforma Centreon². Esta plataforma consiste numa ferramenta de supervisão e monitorização de uma rede, baseando-se no mecanismo de monitorização OpenSource, Nagios³.

The screenshot shows the Centreon web interface. At the top, there's a navigation bar with links for Home, Monitoring (which is selected), Reporting, Configuration, and Administration. Below the navigation is a green header bar with links for Status Details, Performances, Downtimes, and Event Logs. The main content area is titled 'Monitoring > Status Details > Hosts'. It features a search bar and filters for Host Status (All), Host, Status, Poller, and Hostgroup. A table lists one host: 'Centreon-Server' (IP: 127.0.0.1). The table includes columns for Hosts, Status, IP Address, Last Check, Duration, Tries, and Status information. The status information row shows 'OK - 127.0.0.1: rta 0.021ms, lost 0%'. At the bottom of the page, there are footer links for Documentation, Centreon Support, Centreon, and Github Project, along with a copyright notice: 'Copyright © 2005 - 2015'.

Figura 28: Lista de Hosts, ambiente Centreon

² <https://www.centreon.com/en/>

³ <https://www.nagios.org/>

Todos os testes de monitorização realizados, são serviços criados no host "Centreon-Server". Na imagem abaixo, é possível ver os testes criados:

Centreon-Server / Monitoring Server [127.0.0.1]				
	Service Statuses	Performances	Host Informations	Comments
Services				
Services	Status	Duration	Output	
octetsin	WARNING	33m 23s	SNMP WARNING - *53433425*	
ifOutOctets	OK	24m 53s	SNMP OK - 3337984404	
inteface1	OK	25m 53s	SNMP OK - 100	
inteface6	OK	26m 53s	SNMP OK - 10000	
Ping	OK	5d 22h 42m 22s	PING OK - Packet loss = 0%, RTA = 1.48 ms	

Figura 29: Lista de Serviços do Host "Centreon-Server"

Para cada serviço, foi criado um comando, que é o executado na máquina Centreon, que opera sobre o router, pois a máquina do grupo, que possui Docker, não disponibilizava as MIBS necessárias para a obtenção de dados interessantes, tendo por isto sido proposta pelo professor, a monitorização ao router.

Um exemplo da configuração de um comando é o seguinte:

Figura 30: Painel de configuração de um comando

Neste caso acima apresentado, o comando consiste num pedido de informação relativamente à quantidade de octetos que saem na interface 1.

Estes comandos criados, podem ser posteriormente utilizados por serviços criados para um host ou grupo de hosts. A configuração de um serviço consiste no seguinte:

The screenshot shows the 'Modify a Service' configuration interface. Key fields include:

- Linked with Hosts:** Centreon-Server
- Description:** ifOutOctets
- Service Template:** SNMP-Windows-CPU
- Service Check Options:** Check Command: ifOutOctets
- Args:** No argument found for this command
- Service Scheduling Options:**
 - Check Period: 24x7
 - Max Check Attempts: 1
 - Normal Check Interval: * 60 seconds
 - Retry Check Interval: * 60 seconds
 - Active Checks Enabled: Yes (radio button selected)
 - Passive Checks Enabled: No
 - Is Volatile: No

Figura 31: Painel de configuração de um serviço

Para configurar o serviço que fornece a informação acerca dos octetos que saem da interface 1, seleciona-se o conjunto de hosts que possuirá o serviço a ser criado. Neste serviço, escolhe-se o comando criado previamente.

Depois de todos os serviços criados é possível então verificar a informação que recebemos de cada um.

Service octetsin		Monitoring Server 127.0.0.1
Centreon-Server		
Status Details		
Service Status	WARNING	
Status information		SNMP WARNING - *363937721*
Extended status information		
Performance Data		IF-MIB::ifInOctets.1=363937721c
Current Attempt	3 / 3	
State Type	HARD	
Last Check Type	Active	
Last Check	2016/04/21 - 15:09:39	
Next Scheduled Active Check	2016/04/21 - 15:14:39	
Latency	0.094 seconds	
Check Duration	0.071408 seconds	
Last State Change	2016/04/21 - 14:19:09	
Current State Duration		
Last Service Notification		
Current Notification Number	0	
Is This Service Flapping?	No	
Percent State Change	4.934211 %	
In Scheduled Downtime?	No	
Last Update	2016/04/21 - 15:11:08	

Figura 32: Serviço com alerta "Warning"

No exemplo acima, na criação do comando para este serviço, definiu-se um valor, que caso fosse ultrapassado, lançaria um alerta de "warning". Neste caso foi na quantidade de octetos que entram na interface 1.

Quando um serviço corre sem qualquer problema, obtém-se uma mensagem de "ok":

Service interface6		Monitoring Server 127.0.0.1
Centreon-Server		
Status Details		
Service Status	OK	
Status information	SNMP OK - 10000	
Extended status information		
Performance Data	IF-MIB::ifHighSpeed.6=10000	
Current Attempt	1 / 3	
State Type	HARD	
Last Check Type	Active	
Last Check	2016/04/21 - 15:10:39	
Next Scheduled Active Check	2016/04/21 - 15:15:39	
Latency	0.038 seconds	
Check Duration	0.103564 seconds	
Last State Change	2016/04/21 - 14:25:39	
Current State Duration		
Last Service Notification		
Current Notification Number	0	
Is This Service Flapping?	No	
Percent State Change	0 %	
In Scheduled Downtime?	No	
Last Update	2016/04/21 - 15:11:13	

Figura 33: Serviço com alerta "OK"

É possível ainda em cada serviço, ter um gráfico com o histórico do estado do serviço, ao longo de um determinado intervalo de tempo:

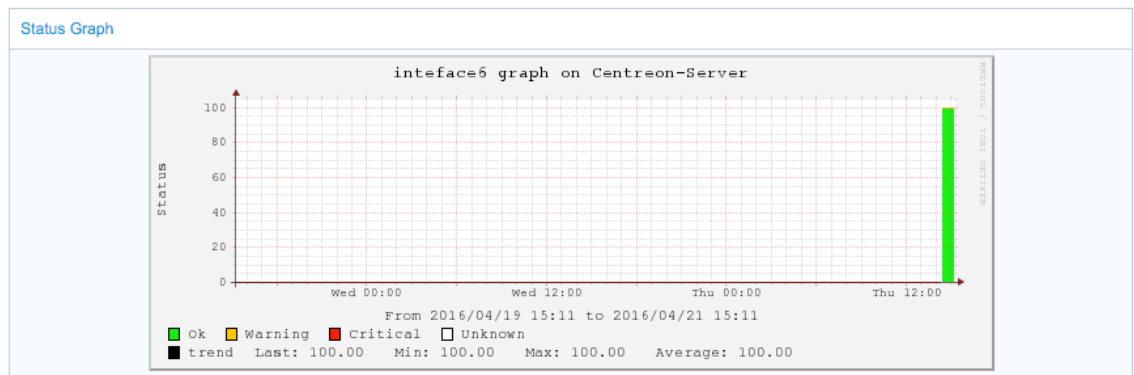


Figura 34: Gráfico do serviço com alerta "OK"

5 Datadog

Datadog é também um sistema de monitorização. Foi utilizado para monitorizar a máquina que contém o ambiente Docker. Para isso, foi instalado na máquina, através do Rancher:

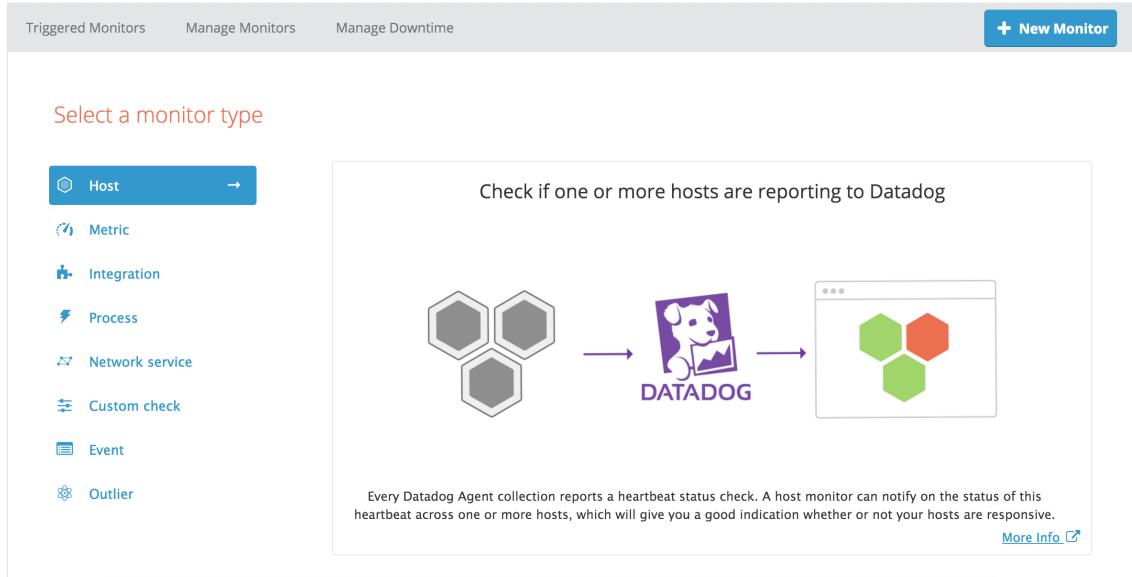


Figura 35: Seleção do sistema a ser monitorizado

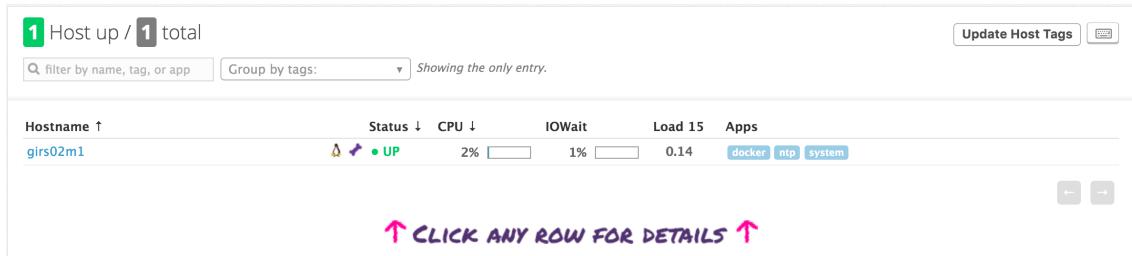


Figura 36: Host que está a ser monitorizado

Foi selecionada a máquina "girs02m1", para ser monitorizada. Quando é selecionada, é apresentada uma dashboard onde é possível ver os eventos que ocorreram na mesma, num intervalo de tempo.

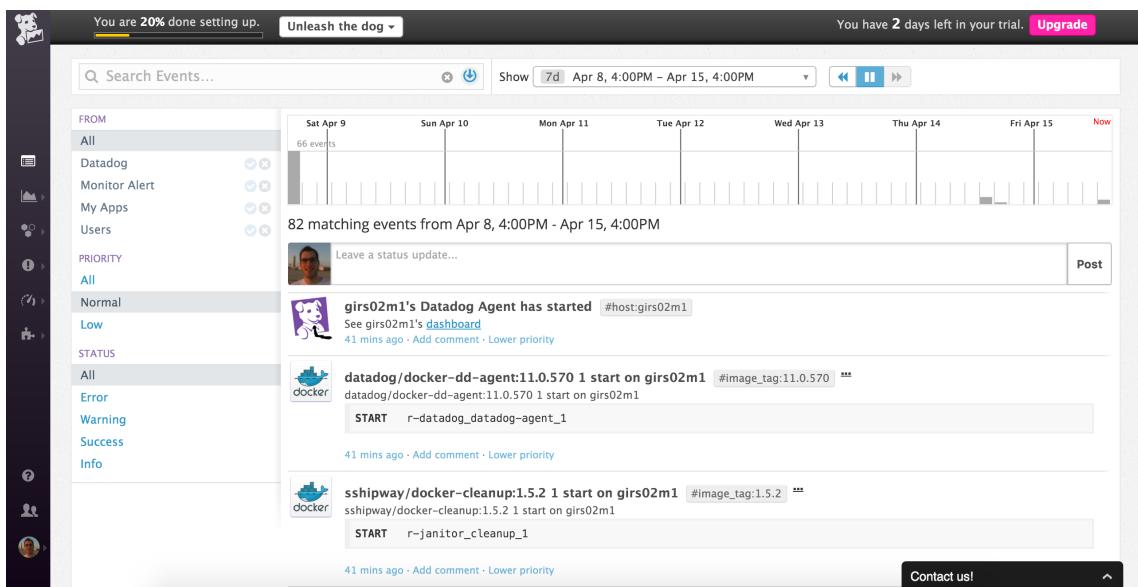


Figura 37: Dashboard Datadog

Um outro ponto, bastante relevante desta plataforma, são os gráficos disponibilizados com informação relativamente à memória utilizada, espaço ocupado, número de containers a funcionar, utilização do cpu, entre outros:

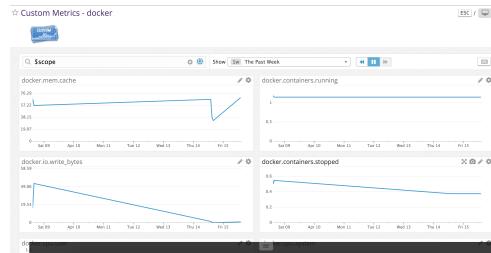


Figura 38: Gráficos Métricas I

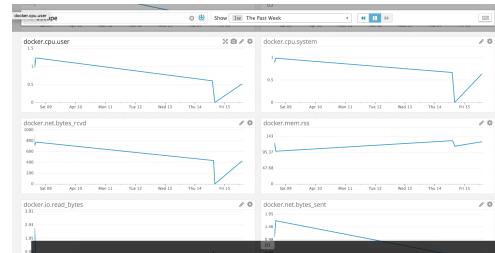


Figura 39: Gráficos Métricas II

6 Load balancer

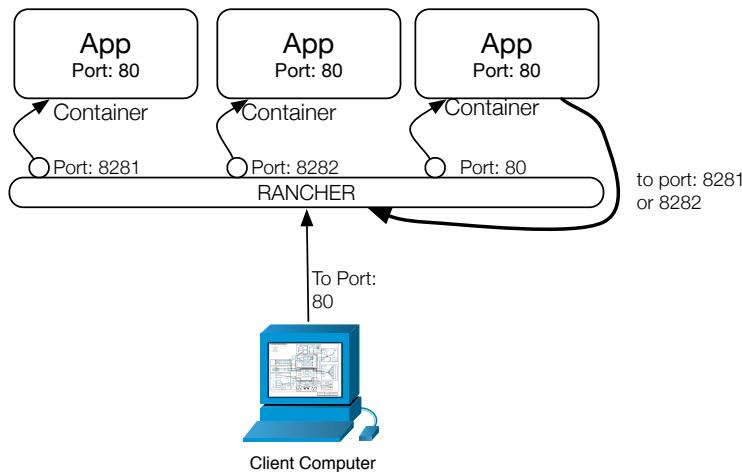


Figura 40: LoadBalancer Simples com NGINX

A nossa solução para fazer o balanceamento passou pela construção de uma Stack com 3 containers, dois que têm a mesma aplicação a correr na porta 80 mapeada para portos diferentes do Host e outro container que tem NGINX à escuta também num porto diferente.

Para criar a aplicação fez-se as seguintes ações:

```
# apt-get update
# apt-get install -y apache2 php5 libapache2-mod-php5 vim
# cd /var/www/html/
# rm index.html
# vim index.php
# service apache2 start
```

```
<?php

$start_script = gettimeofday();

const INTERVAL = 0.04;
$dummy = 0;
$cycles = 1000;

$rand = rand(50, 100);
for($j = 0; $j < $rand; $j++){
    $start = gettimeofday();
    for($i=0; $i<$cycles; $i++){
```

```

    $dummy += $i*$i;
}

$stop = gettimeofday();

$spentTime = ($stop["sec"]-$start["sec"]) +
    ($stop["usec"]-$start["usec"]) / 1000000.0;
$cycles = ($cycles / $spentTime) * ($target_load / 100.0) * INTERVAL;
usleep((1000000 * INTERVAL - $spentTime*1000000));
}

$stop_script = gettimeofday();

echo "<pre>";
echo "start:_";
print_r($start_script);
echo "\n_stop:_";
print_r($stop_script);
echo "\n_done!_\n";
echo "hostname:_";
echo gethostname();
echo "</pre>";

?>

```

6.1 Nginx como Load Balancer

Ao aceder pelo seu computador, para a porta do LoadBalancer, este irá reencaminhar o pedido para outro container e pode seguir várias políticas para fazer este reencaminhamento:

Por defeito

Por defeito o algoritmo de balanceamento usado é "round-robin", todos os pedidos são processados independentemente da carga e são redirecionados de igual forma para os containers disponíveis na configuração.

Uma configuração possível para a nossa solução:

```

upstream 192.168.215.20 {
    server 192.168.215.20:8281;
    server 192.168.215.20:8282;
}

server {
    location / {
        proxy_pass  http://192.168.215.20;
    }
}

```

```
    }
}
```

Least connected

Outra forma de fazer load balancing é com o least-connected (menos-conectado). Este permite controlar a carga nas instâncias de forma justa em que alguns dos pedidos levam mais tempo para serem concluídos.

Com este método de balanceamento, Nginx vai tentar não sobrecarregar um servidor que esteja ocupados com pedidos excessivos, distribuindo os novos pedidos para um servidor que esteja menos ocupado.

Uma configuração possível para a nossa solução:

```
upstream 192.168.215.20 {
    least_conn;
    server 192.168.215.20:8281;
    server 192.168.215.20:8282;
}

server {
    location / {
        proxy_pass http://192.168.215.20;
    }
}
```

Session persistence

Com round-robin ou least-connected, a sessão do cliente pode ser destruída se não existir um meio de persistência destas. Não há garantia de que o mesmo cliente será sempre direcionado para o mesmo servidor.

Se existir necessidade de permanecer o cliente num dado servidor para garantir a persistência da sessão, é necessário usar o mecanismo ip-hash de平衡amento de carga.

Com este mecanismo, o IP do cliente é usado como chave da tabela de hashing para determinar qual é servidor que deve ser selecionado para os pedidos dos clientes. Este método garante que os pedidos do mesmo cliente sejam sempre direcionados para o mesmo servidor exceto quando o servidor já não está disponível.

Uma configuração possível para a nossa solução:

```
upstream 192.168.215.20 {
    ip_hash;
    server 192.168.215.20:8281;
    server 192.168.215.20:8282;
}
```

```

server {
  location / {
    proxy_pass http://192.168.215.20;
  }
}

```

Weight

Esta opção permite enviar os utilizadores para os servidores com mais precisão e alocar peso com peso certo para determinadas máquinas.

Nginx permite que se redirecione o tráfego para uma máquina específica com mais proporção ou com menos proporção.

Uma configuração possível para a nossa solução:

```

upstream 192.168.215.20 {
  server 192.168.215.20:8281 weight=1;
  server 192.168.215.20:8282 weight=2;
}

server {
  location / {
    proxy_pass http://192.168.215.20;
  }
}

```

Por defeito, o peso é 1. Com o peso de 2, o container na porta 8282 irá receber duas vezes mais tráfego do que o container no porto 8281.

Max Fails

De acordo ⁴ com a configuração por defeito do round robin, o nginx irá continuar a enviar pedidos para o container, mesmo que o container/servidor não estejam a responder. Com a diretiva max fails é possível prevenir automaticamente detetando servidor que não estejam a responder por um tempo determinado.

Existem dois comandos que podem ser usados: max_fails e fail_timeout. Max fails refere-se ao número máximo de tentativas falhadas ao tentar conectar-se ao servidor antes de se considerar inativo. Fail_timeout é o tempo que é necessário para considerar um servidor intativo. Quando esse tempo expirar, irá ser eliminado da tabela, quando voltar a estar disponível, irá voltar a estar na tabela.

Uma configuração possível para a nossa solução:

⁴<https://www.digitalocean.com/community/tutorials/how-to-set-up-nginx-load-balancing>

```

upstream 192.168.215.20 {
    server 192.168.215.20:8281 max_fails=3 fail_timeout=15s;
    server 192.168.215.20:8282 weight=2;
}

server {
    location / {
        proxy_pass http://192.168.215.20;
    }
}

```

6.2 Load balancing tendo em conta a carga do CPU dos containers

Pretende-se usando SNMP obter a carga de cada container em tempo real de forma a enviar o pedido para o container com a carga de CPU menor.

Carga de CPU de um container

Para obter a carga do CPU de um container foi usado um script bash para obter a informação sobre o estado dos containers. É feito o parse da carga de CPU dada pelo comando **docker stats**.

O container é identificado pela porta que ele usa no host, que neste caso é a 8281.

```

#!/usr/bin/env bash
PORTS=$(docker ps | grep ">80/tcp" | grep "0.0.0.0:8281" \
| awk '/_/{print $10}' | sed -r 's/[0.0.0.0:]+//g' \
| sed -r 's/->80/tcp//') # ports
CONTAINER=$( docker ps | grep ">80/tcp" | grep "0.0.0.0:8281" \
| awk '/_/{print $1}') # containers
CPU_CONTAINER=$(docker stats --no-stream $CONTAINER \
| awk '/_/{print $2}' | grep "%" | sed -r 's/%//')
echo ${CPU_CONTAINER}

```

SNMP

Para que o snmp tivesse acesso ao daemon do Docker foi necessário executar o seguinte comando:

```

# sudo usermod -aG docker snmp
# sudo service snmpd restart

```

Já a configuração do SNMP foi normal, usando a versão 3, foi criado um utilizador e uma password para o mesmo:

```
createUser bootstrap MD5 temp_password DES
```

Foi também necessário colocar o SNMP disponível em todas as interfaces:

```
agentAddress udp:161, udp6:[::1]:161
```

Na secção "EXTENDING THE AGENT", foi colocado dois comandos, um "app1" e outro "app2" que serão encarregados por retornar a percentagem de CPU usada pelo container.

```
extend    app1    /bin/sh /path/to/app1.sh
extend    app2    /bin/sh /path/to/app2.sh
```

Usando então um snmpget é possível então obter a percentagem pretendida.

```
# snmpget -v 3 -u authOnlyUser -l authPriv -a MD5 -A temp_password -x DES -X temp_password IP.IP.IP.IP 'NET-SNMP-EXTEND-MIB::nsExtendOutputFull."app1"'
```

Obtendo a carga dos containers usando C

Usando o popen executa-se o comando SNMP, o código apresentado no relatório não é o completo.

```
/* Open the command for reading. */
fp = popen("snmpget -v 3 -u authOnlyUser -l authPriv -a MD5
-A temp_password -x DES -X temp_password IP.IP.IP.IP
'NET-SNMP-EXTEND-MIB::nsExtendOutputFull.\"app1\"', "r");
if (fp == NULL) {
    printf("Failed to run command\n");
    exit(1);
}
```

Depois faz-se o parse da string obtida para inteiro:

```
char app1[10];

int pos = 57;
int i = pos;

while(path[i]!=0){
    app1[i-pos] = path[i];
    i = i + 1;
}

int app1_val = parseToInt(app1, 0, i-pos-1);
```

De seguida faz-se a comparação.

Costumização do NGINX



Figura 41: Solução final

Na porta 80 estará o NGINX modificado que tem em consideração a carga do CPU de cada container. Na porta 8281 estará a app1, na porta 8282 a app2 e na porta 8283 um loadbalancer NGINX não modificado.

Para se realizar a costumização do NGINX foi inicialmente instalado as bibliotecas necessárias e feito o git clone do repositório do NGINX.

```
# apt-get update
# apt-get install -y git vim build-essential libcurl4-openssl-dev libpcre3 libpcre3-dev
libpcrecpp0 libssl0.9.8 libssl-dev zlib1g zlib1g-dev lsb-base openssl libssl-dev
libgeoip1 libgeoip-dev google-perf-tools libgoogle-perf-tools-dev libperl-dev
libgd2-xpm-dev libatomic-ops-dev libxml2-dev libxslt1-dev python-dev snmp snmp-mibs-downloader
# cd /
# git clone https://github.com/nginx/nginx.git
# vim nginx/src/http/ngx_http_upstream_round_robin.c # mais em baixo
estará focada a alteração do código
# cd nginx
# ./auto/configure --sbin-path=/usr/local/nginx/nginx --conf-path=/etc/nginx/nginx.conf
--pid-path=/var/run/nginx.pid --with-debug --with-select_module --with-poll_module
--error-log-path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log
# make && make install
# mkdir /etc/nginx/sites-available
# mkdir /etc/nginx/sites-enabled
# vim /etc/nginx/nginx.conf # add in http section: include /etc/nginx/sites-available/*;
# vim /etc/nginx/sites-available/default

upstream 192.168.215.20 {
    server 192.168.215.20:8281;
    server 192.168.215.20:8282;
}

server {
```

```

location / {
    proxy_pass  http://192.168.215.20;
}
}

```

Para iniciar e terminar o NGINX, é usado os seguintes comandos:

```

# /usr/local/nginx/nginx # start NGINX
# kill -QUIT $cat/var/run/nginx.pid # stop NGINX

```

No container irá existir um daemon que estará sempre em execução e o seu trabalho será obter os a carga dos containers por SNMP e guardar numa variável numa zona partilhada para que o NGINX depois consiga aceder a essa variável sem ter de esperar pela resposta do SNMP.

Durante o nosso desenvolvimento também foi possível perceber que o SNMP responde sempre da mesma forma durante 3 ou 4 segundos, aparentemente fazendo cache dos pedidos efetuados, dificulta assim ter dados sobre o CPU atualizados ao segundo, sendo que o ganho terá de ser considerado para intervalos superiores a este.

No ficheiro `nginx/src/http/ngx_http_upstream_round_robin.c` foi modificada a função `ngx_http_upstream_get_peer` para obter os dados guardados na zona partilhada:

```

static ngx_http_upstream_rr_peer_t *
ngx_http_upstream_get_peer(ngx_http_upstream_rr_peer_data_t *rrp)
{
    time_t                                now;
    uintptr_t                             m;
    ngx_int_t                             total;
    ngx_uint_t                            i, n, p;
    ngx_http_upstream_rr_peer_t   *peer, *best;

    now = ngx_time();

    best = NULL;
    total = 0;

#if (NGX_SUPPRESS_WARN)
    p = 0;
#endif
    // https://www.cs.cf.ac.uk/Dave/C/node27.html

    int shmid;
    key_t key;

```

```
int *shm, *s;

/*
 * We need to get the segment named
 * "5678", created by the server.
 */
key = 5678;

/*
 * Locate the segment.
 */
if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
    perror("shmget");
    exit(1);
}

/*
 * Now we attach the segment to our data space.
 */
if ((shm = shmat(shmid, NULL, 0)) == (int *) -1) {
    perror("shmat");
    exit(1);
}

/*
 * Now read what the server put in the memory.
 */
s = shm;

ngx_uint_t ix = *s;

for (peer = rrp->peers->peer, i = 0;
     peer;
     peer = peer->next, i++)
{
    if (best == NULL || i==ix) {
        best = peer;
        p = i;
        break;
    }
}

rrp->current = best;
```

```

n = p / (8 * sizeof(uintptr_t));
m = (uintptr_t) 1 << p % (8 * sizeof(uintptr_t));

rrp->tried[n] |= m;

best->current_weight -= total;

if (now - best->checked > best->fail_timeout) {
    best->checked = now;
}

return best;
}

```

Para o daemon responsável por obter a informação sobre os containers através de SNMP foi elaborado o seguinte código:

```

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>

#define SHMSZ      27

// gcc get-snmp.c -o server-get-snmp.out && ./server-get-snmp.out &> output
// ps aux | grep "get"

int parseToInt(const char* s, int start, int stop) {
    unsigned long long int m = 1;
    int ret = 0;
    int i = stop;

    for (; i >= start; i--) {
        if(isdigit(s[i])){
            ret += (s[i] - '0') * m;
            m *= 10;
        }
    }

    return ret;
}

```

```
}

int main(int argc, char const *argv[]) {
    // https://www.cs.cf.ac.uk/Dave/C/node27.html

    int shmid;
    key_t key;
    int *shm, *s;

    /*
     * We'll name our shared memory segment
     * "5678".
     */
    key = 5678;

    /*
     * Create the segment.
     */
    if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    /*
     * Now we attach the segment to our data space.
     */
    if ((shm = shmat(shmid, NULL, 0)) == (int *) -1) {
        perror("shmat");
        exit(1);
    }

    /*
     * Now put some things into the memory for the
     * other process to read.
     */
    s = shm;

    while(1){
        printf("Getting app_1_CPU_status....\n");

        FILE *fp;
        char path[1035] = "";

```

```

/* Open the command for reading. */
fp = popen("snmpget -v 3 -u authOnlyUser -l
authPriv -a MD5 -A temp_password -x DES -X
temp_password 192.168.215.20
'NET-SNMP-EXTEND-MIB::nsExtendOutputFull.\\"app1\\\"", "r");
if (fp == NULL) {
    printf("Failed to run command\n");
    exit(1);
}

/* Read the output a line at a time - output it. */
while (fgets(path, sizeof(path)-1, fp) != NULL) {
    printf("%s", path);
}

/* close */
pclose(fp);

// parse what we need
char app1[10];

int pos = 57;
int i = pos;

while(path[i]!=0){
    app1[i-pos] = path[i];
    i = i + 1;
}

int app1_val = parseToInt(app1, 0, i-pos-1);

printf("VALUE_APP1: %d (%)\n", app1_val);

printf("Getting app2 CPU status....\n");
FILE *fp2;
char path2[1035];

/* Open the command for reading. */
fp2 = popen("snmpget -v 3 -u authOnlyUser -l authPriv
-a MD5 -A temp_password -x DES -X
temp_password 192.168.215.20
'NET-SNMP-EXTEND-MIB::nsExtendOutputFull.\\"app2\\\"", "r");
if (fp2 == NULL) {

```

```

    printf("Failed_to_run_command\n");
    exit(1);
}

/* Read the output a line at a time - output it. */
while (fgets(path2, sizeof(path2)-1, fp2) != NULL) {
    printf("%s", path2);
}

/* close */
pclose(fp2);

// parse what we need
char app2[10];

i = pos;

while(path[i]!=0){
    app1[i-pos] = path[i];
    i = i + 1;
}

int app2_val = parseToInt(app2, 0, i-pos-1);

printf("VALUE_APP2:_%d_(%)\n", app2_val);

int ix = 0;

if(app1_val>app2_val){
    ix = 1;
}

*s = ix;
sleep(4);
}

return 0;
}

```

Benchmark do NGINX comparando com o normal

Usando o Apache Benchmark, sabendo que cada pedido demora em média 5 a 6 segundos, foram realizados 500 pedidos com concorrência de 30 pedidos.

Tabela 6: Comparaçāo entre testes

	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Teste 6	Teste 7	Teste 8	Média
Normal	161.498	192.892	181.070	189.433	173.298	171.766	176.420	185.243	178,952
Modificado	107.544	124.692	109.118	117.975	110.223	105.965	118.421	127.008	115,118

Tempo médio por pedido (ms)

Apesar da modificāo do NGINX nāo ter sido completa de forma a retirar todo o códigō desnecessário e acelerar ainda mais os resultados obtidos, estes foram interessantes e melhores em comparação com o módulo normal.

7 Conclusão

A realização deste trabalho, permitiu que se percebesse melhor, sistemas de virtualização, armazenamento, RAID, NFS, monitorização e de load balancing.

Este trabalho permitiu ainda, a colocação em prática dos vários tópicos apresentados ao longo da componente teórica da disciplina.