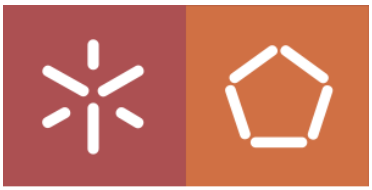


Universidade do Minho
Escola de Engenharia

MESTRADO EM ENGENHARIA INFORMÁTICA

APLICAÇÕES E SERVIÇOS DE COMPUTAÇÃO EM NUVEM

TRABALHO PRÁTICO

INSTALAÇÃO E ANÁLISE DA APLICAÇÃO "WIKI.JS"

Grupo 07

ANA FILIPA PEREIRA - PG46978

CAROLINA SANTEJO - PG47102

DIOGO RISCA – PG47845

PHILIPE CORTEZ – PG44422

RAQUEL COSTA - PG47600

ÍNDICE

I. INTRODUÇÃO

Neste trabalho prático da unidade curricular de Aplicações e Serviços de Computação em Nuvem, foi pedido ao grupo que instalasse a aplicação *Wiki.js* recorrendo a serviços da *Google Cloud*. Além disto, e recorrendo ao conhecimento adquirido ao longos das aulas práticas e teóricas, pretendeu-se que todo o processo de instalação fosse o mais automatizado possível, tendo-se, para isto, utilizado a ferramenta *Ansible*.

Neste projeto, pretendia-se, também, que a aplicação fosse escalável, ou seja, que respondesse de forma eficiente mesmo quando o número de clientes aumenta. Para garantir isto, o grupo utilizou a metodologia de *autoscaling*. Por outro lado, o *Wiki.js* deveria permitir o balanceamento de carga para evitar pontos de contenção de desempenho, sendo que para este ponto decidimos recorrer ao serviço *Google Kubernetes Engine*, que efetua este processo de forma automática.

Além disto, utilizou-se a ferramenta *Jmeter* e os serviços de monitorização da Google *Cloud Platform* para efetuar a avaliação da aplicação.

Assim sendo, ao longo deste relatório será detalhadamente explicado todas as decisões tomadas relativas ao processo de instalação, monitorização e avaliação da aplicação *Wiki.js*.

II. ARQUITETURA E COMPONENTES DA APLICAÇÃO

A aplicação Wiki.js é um software criado com o intuito de disponibilizar um ambiente responsivo e customizável de criação e edição de páginas de documentação, segundo uma política de *open source*. Isto permite facilitar os processos colaborativos de escrita e edição das equipas de trabalho uma vez que têm à sua disposição, além de um ambiente de escrita bastante produtivo, várias ferramentas disponíveis.

A *Wiki.js* é, desta forma, uma aplicação bastante poderosa, desenvolvida em *Node.js*, escrita em *JavaScript* e com diversas vantagens das quais realçamos:

- Interface intuitiva e fácil de usar por todos os membros da equipa;
- Possui mecanismos para revisão e controlo de versões;
- Suporte para várias linguagens diferentes;
- Existência de funcionalidades de pesquisa, sendo que em alguns casos será necessário recorrer a componentes *third-paty* como o *ElasticSearch* e *Azure Search;*
- Wiki.js oferece módulos de armazenamento com suporte completo para *backup* e sincronização de conteúdo;
- Existência de um sistema de gestão de utilizadores;
- Possibilidade de efetuar comentários em páginas ou *posts.*

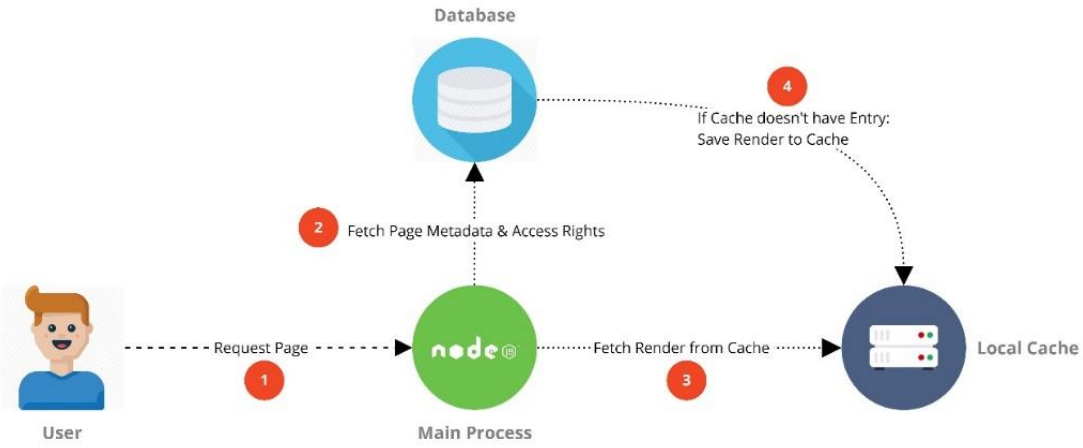


FIGURA 1 - REQUEST "VER PÁGINA"

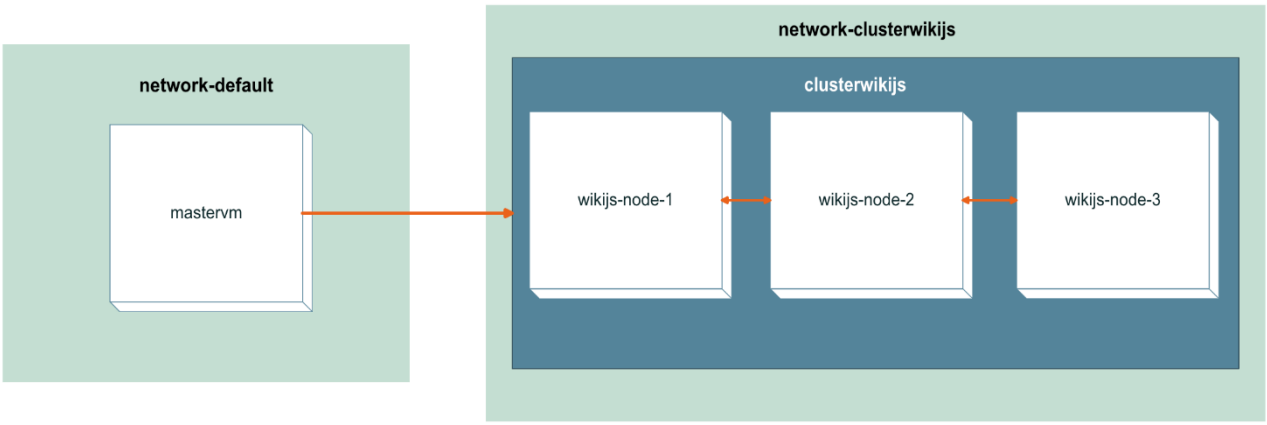
Na figura 1, podemos ver um diagrama que representa o pedido de ver uma página. Neste caso, e como acontece na maioria dos *requests*, é feita, em primeiro lugar, a comunicação com o *frontend* e o *Node.js*. Este último componente, verifica se o conteúdo que necessita se encontra em *cache*, e, caso não esteja, comunica com a base de dados.

É de realçar que o software da aplicação não se encontra dividido em *frontend* e *backend*. Por outro, é fornecido suporte para diversas bases de dados, sendo que o grupo escolheu utilizar neste projeto a *PostgreSQL*.

III. INFRAESTRUTURA DA SOLUÇÃO

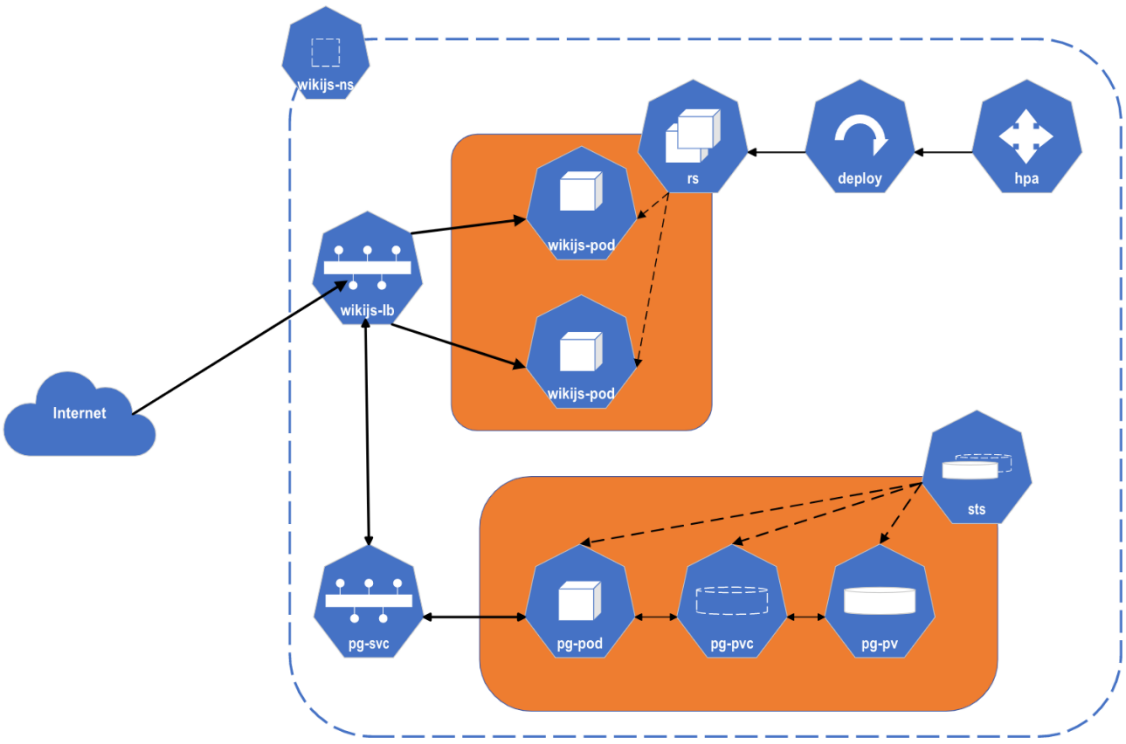
1. DESCRIÇÃO DA INFRAESTRUTURA NECESSÁRIA PARA CORRER O CLUSTER

O *cluster* é composto por três instâncias de VM do tipo e2-medium, sendo que cada máquina possui 4Gb de memória RAM e 2 vCPU's, todas as máquinas se encontram na mesma rede (network-clusterwikijs) e por razões financeiras todas as três instâncias se encontram na mesma *region* (europe-west1), todas maquinas do cluster estão configuradas para se autorrepararem isso é ainda que sejam desligadas mediante a um erro ou ação humana as mesmas irão reiniciar e subir o ambiente kubernetes necessário para que o cluster funcione em potencia máxima. Para além disso temos também a máquina denominada vmmaster a qual foi provisionada com o intuito de nos oferecer um ambiente estável e confiável a partir do qual possamos executar os playbooks do ansible a qual se encontra numa rede diferente, porém possui as mesmas configurações de hardware das máquinas do cluster.



2. DESCRIÇÃO DOS OBJETOS DO CLUSTER

Para além das máquinas virtuais que compõem o cluster temos vários componentes internos do cluster necessários para que tudo funcione corretamente, a imagem abaixo demonstra quais são esses componentes e como cada um deles esta conectado.



O banco de dados e gerenciado por um StatefulSet que fica a cargo de garantir que sempre haja ao menos um Pod do banco de dados a correr e que esse esteja ligado a um volume persistente onde são armazenados os dados de forma efetiva, por sua vez a app wikijs e gerenciada por um Deployment o qual fica responsável por assegurar que ao menos um Pod do wikijs esta a correr, reiniciando os Pod's em caso de erro, a função do Deployment e semelhante a do StatefulSet exceto que o Deployment não tem como assegurar a ligação entre um Pod e um volume persistente.

Foi configurado um HorizontalPodAutoscaler para o Deployment do wikijs que ira criar até dez novos Pod's do wikijs caso os recursos de processamento dos Pod's existentes não sejam sufientes para lidar com a demanda, o HorizontalPodAutoscaler também ira destruir os pods criados uma vez que os mesmos não sejam mais necessários.

Devido a sua natureza efémera e dinâmica os Pod's são criados e destruídos a todo momento o que faz com que Pod's com diferentes endereços de IP surjam a todo instante, para assegurar a comunicação entre os Pod's do wikijs e do banco de dados, foram criados dois serviços a saber pg-svc e wikijs-lb sendo que o primeiro foi associado a todos os Pod's do banco de dados e o segundo a todos os Pod's da wikijs, os serviços mantem dados a respeito do endereço dos Pod's aos quais estão associados bem como o endereço de outros serviços de forma que são capazes de gerenciar as comunicações internas entre os Pod's aos quais estão associados e demais serviços

Para fazer com que a aplicação wikijs seja accessivel ao mundo exterior fora do cluster o serviço wikijs-lb foi configurado como ExternalLoadBalancer provendo assim um IP publico através do qual a aplicação pode ser acessada e ainda faz o papel de balanceador de carga que distribui por igual as requisições para os Pod's wikijs.

3. OPERAÇÕES CRÍTICAS

Um ponto crítico de falha é qualquer componente de um sistema que, caso falhe, provoca a falência do sistema. Durante o planeamento de um sistema, a alternativa para evitar a existência de pontos de falha passa por adicionar componentes redundantes e por replicar as partes críticas desse sistema.

No caso de sistemas *Wiki,* grande parte das operações são realizadas por editores ou escritores sobre a base de dados. Desta forma, caso se verifique a inacessibilidade aos dados, todo o sistema ficará comprometido pelo que a base de dados é um ponto crítico de falha. O *frontend* da aplicação é também um single *point of failure*, uma vez que o sustento que a aplicação tem na sua vertente web sofrerá um grande impacto em caso de falha.

IV. INSTALAÇÃO E CONFIGURAÇÃO DA APLICAÇÃO

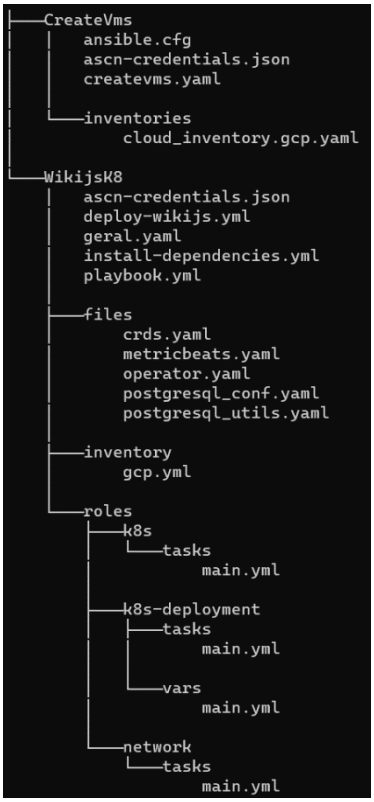
1. DESCRIÇÃO DAS FERRAMENTAS UTILIZADAS

Alguns dos objetivos deste projeto consistiram em fazer o provisionamento e o *deployment* da aplicação Wiki.js. Ambos estes processos, quando feitos passo a passo e manualmente tornam-se repetitivos e morosos pelo que surgiu a necessidade de recorrer a alguma ferramenta que permitisse automatizar estes processos. Desta forma, o grupo decidiu utilizar a ferramenta *Ansible* já abordada nas aulas da unidade curricular. O *Ansible* é uma ferramenta *open source* simples, mas poderosa que permite, além de outras funcionalidades interessantes, configurar e fazer o *deploy* das aplicações nos ambientes de desenvolvimento, de testes e de produção, sendo isto feito a partir de uma única máquina. Por outro lado, o Ansible possui componentes bastante uteis tais como *inventory*, *tasks*, *module*, *role* ou *playbook*, sendo este último um script escrito em YAML, no qual se definem as tarefas a aplicar em targets específicos.

Por outro lado, e visto que foi pedido que, no projeto, se utilizasse a *Google Cloud Platform*, o grupo decidiu tirar partido de serviços disponibilizados pela plataforma e que foram bastante uteis na realização do trabalho. O *Google Kubernetes Engine* foi um dos serviços usados. O GKE oferece um ambiente pronto para fazer *deployment*, gerenciar e escalar aplicações com *containers* usando a infraestrutura do Google. Este ambiente consiste em várias máquinas (*Compute Engine instances*) que agrupadas formam um *cluster*. Além disto o GKE fornece operações para monitorizar e gerir esses mesmos *clusters*. Desta maneira, o *Google Kubernetes Engine* permite o escalonamento automático de *pods* e *clusters* (*autoscaling*) baseado em diferentes métricas, a distribuição de carga automática pelos *pods* (*load balancing*) e uma maior segurança, dado que os *clusters* do GKE têm suporte nativo à Política de rede do *Kubernetes* para restringir o tráfego usando regras de *firewall* no nível dos *pods*.

2. INSTALAÇÃO AUTOMÁTICA DA APLICAÇÃO

2.1 DESCRIÇÃO GERAL



Para automatizar a instalação da aplicação com as ferramentas referidas foram utilizados diversos ficheiros *yaml.* Em primeiro lugar, podemos destacar a pasta *CreateVms* onde se encontram os ficheiros necessários para criar uma instância na *Google Cloud*. Por outro lado, na pasta *WikijsK8* encontram-se todos os ficheiros necessários tanto para a criação de um cluster na Google Cloud como para a instalação da aplicação *Wikijs* dentro do mesmo.

2.2 CRIAÇÃO DA VMMASTER

A primeira fase da instalação consistiu na automatização da criação de uma instância, à qual demos o nome de *mastervm* na *Google Cloud*. Para isso será necessário correr o *playbook* '*createvms.yaml*' utilizando o *ansible.*

Esta máquina será do tipo '*e-small',* isto é, possui 2 vCPU's e 2GB de memória *RAM,* tem como sistema operativo o Ubuntu e terá um disco de 20GB. No final da sua criação será copiado todo o conteúdo da pasta WikijsK8 para dentro da mesma.

O grupo teve a necessidade de criar esta máquina virtual, uma vez que é a partir dela que será possível analisar as operações realizadas no cluster e, desta forma será sempre possível acede-la a partir de qualquer dispositivo.

2.3 INSTALAÇÃO DAS DEPENDÊNCIAS

Após a criação da *mastervm,* e antes de passar para a instalação da aplicação, foi requerida a instalação de algumas ferramentas e bibliotecas.

Em primeiro lugar foi necessária a instalação manual, tanto do *ansible* como de algumas das suas bibliotecas, nomeadamente a *cloud.common*, *community.kubernetes* e *kubernetes.core*.

De seguida será necessário correr o *playbook 'install-dependencies.yml'* que irá fazer a instalação automática do *kubernetes* e de algumas bibliotecas do *python*.

A instalação do *kubernetes* na *mastervm* será útil para fazer a monitorização do cluster.

2.4 CRIAÇÃO DO CLUSTER

A terceira fase da instalação consistiu na criação do cluster na *Google Kubernetes Engine* onde será feito o *deployment* do *Wikijs*.

Desta forma correndo o *playbook 'playbook.yml'* com o ansible, irá criar um cluster com uma *pool* de nodos associada. Inicialmente esta *pool* será constituída por 3 nodos, no entanto será possível escalar automaticamente até 10 nodos. Cada máquina será do tipo *e2-medium* e terá 20GB de espaço em disco. No final será também executado um comando automaticamente que conecta a *mastervm* ao cluster.

2.5 *DEPLOYMENT* DA APLICAÇÃO

Tendo o cluster devidamente criado e configurado, foi possível passar para a ultima fase que consistiu no *deployment* do *Wikijs.* Para automatizar este processo foi utilizada a biblioteca *kubernetes.core.k8s* do *ansible*.

Em primeiro lugar para fazer a instalação do wikijs foram criados 3 objetos. O objeto principal será do tipo *Deployment* que é onde será indicada a imagem da aplicação, o número de replicas e outras variáveis necessárias. Para ser possível aceder à aplicação através do exterior foi também necessário criar um *Service* do tipo *LoadBalancer* com a porta 3000 que é a porta *default* do *Wikijs.* Por ultimo, foi criado um escalonador automático (*HorizontalPodAutoscaler*), que será responsável por aumentar e diminuir o numero de réplicas de acordo com o número de pedidos à aplicação.

De seguida, como o *Wikijs* é uma aplicação que requer a utilização de uma base de dados, foi escolhida a *PostgreSQL.* Desta forma, foi criado um objeto do tipo *StatefulSet,* onde se indicou a imagem e as variáveis de acesso à base de dados. Cada réplica terá associado um volume persistente com 20GB. De seguida, foi necessário também criar um *Secret,* que é onde serão guardadas todas as credenciais de acesso à base de dados. Por fim, para que a aplicação tenha acesso à BD é gerado um serviço que escuta na porta 5432

V. AVALIAÇÃO DE DESEMPENHO

A avaliação de desempenho é uma etapa fundamental deste trabalho prático uma vez que é através dela que se observa o comportamento da aplicação Wiki.js quando submetida a grandes quantidades de pedidos de utilizadores feitos em simultâneo. A análise da aplicação é feita tendo-se em conta diversas métricas sendo que as selecionadas pelo grupo foram o *response time*, *CPU usage time*, *abort rate* e *throughput*.

1. FERRAMENTAS UTILIZADAS

De forma a ser possível obter métricas e gráficos que permitissem analisar o comportamento da aplicação em diferentes cenários, o grupo recorreu não só à ferramenta *Jmeter* já falada nas aulas da UC, mas também os serviços de monitorização disponibilizados pela *Google Cloud*.

1.1 JMETER

Como já foi referido, a avaliação de desempenho foi feita recorrendo à ferramenta *JMeter*, que permite efetuar testes de carga sobre a aplicação em estudo e que simulam pedidos de clientes. Para realizar estes testes, foi preciso definir alguns campos num *Http Request*, nomeadamente, o número de *threads* (*users* que interagem com a aplicação), o *ramp-up period* e o *loop count*. É de realçar que para avaliar o comportamento do *Wiki.js* foi importante ter em conta as várias páginas que podem ser pedidas pelos *users*. Isto acontece uma vez que num cenário real esta será a utilidade que mais vai ser usada pelos utilizadores da aplicação.

Com o *Jmeter*, conseguimos obter informações relativas às métricas de *response time*, *abort rate* e *throughput*.

1.2 METRICS EXPLORER

Para efetuar a avaliação do Wiki.js o grupo usou, também, o *Metrics Explorer* sendo esta uma ferramenta de monitorização disponibilizada pela *Google Cloud.* Com esta ferramenta é possível obter gráficos detalhados relativos a diferentes métricas. O grupo usou o *Metrics Explorer* para obter informações relativas ao *CPU usage time*.

VI. RESULTADOS FINAIS

Primeiramente, foi definido um conjunto de páginas para avaliar e monitorizar, de acordo com as várias métricas escolhidas e os testes estipulados. Para tal, o grupo escolheu as páginas que melhor se aplicam num cenário real onde existem inúmeros utilizadores da aplicação. Assim sendo, as seguintes páginas são aquelas que o grupo considerou que seriam as mais manuseadas por utilizadores da aplicação.

É também de notar, que durantes os testes realizados, considerou-se o seguinte fator, de modo a comparar os vários resultados obtidos para cada página:

- Variação do número de pedidos simultâneos, isto é, *threads* ou clientes concorrentes, definindo o seguinte conjunto de valores: 10, 100, 500, 1000, 8000, 15000

1. TESTES E RESULTADOS

1.1 LOGIN

	Nº Threads					
Tempo médio de resposta (ms)	**10**	**100**	**500**	**1000**	**8000**	**15000**
	66	127	1232	3018	15998	29323

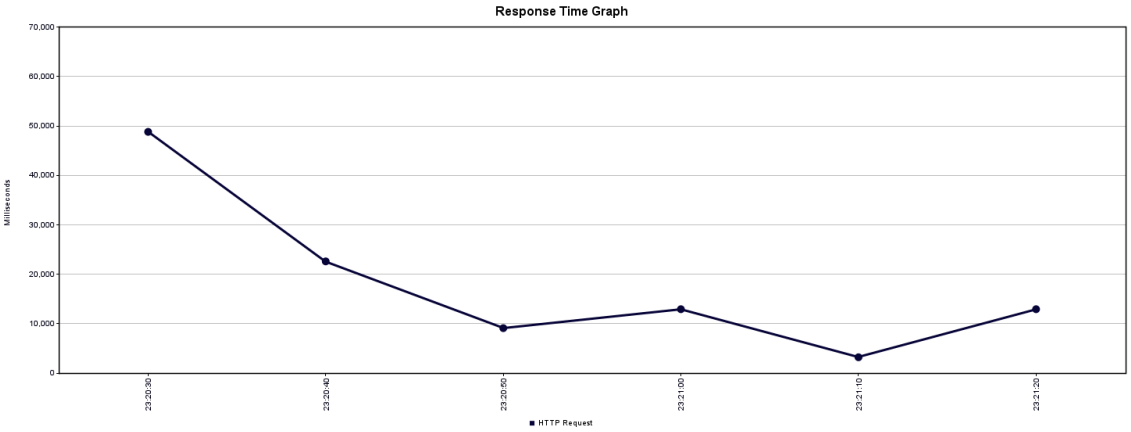
À medida que aumenta o número de pedidos, como era de esperar, o tempo de resposta vai aumentando uma vez que a aplicação tem de responder a cada vez mais pedidos em simultâneo.

	Nº Threads					
Abort Rate (%)	**10**	**100**	**500**	**1000**	**8000**	**15000**
	0.00%	0.00%	0.00%	0.00%	38.15%	64.94%

Como se pode verificar na tabela, até ao nº de threads igual a 100 não houve qualquer taxa de erro nos pedidos, ou seja, a aplicação foi capaz de atender a todos. A partir de 8000 threads, a nossa aplicação atingiu um limite em que não foi capaz de responder a alguns pedidos efetuados, uma vez que atingiu o número máximo de *pods,* ou seja, não foi possível escalar mais a aplicação.

	Nº Threads					
Throughput (requests/ sec)	**10**	**100**	**500**	**1000**	**8000**	**15000**
	3.8	23.2	28.9	301.9	188.3	153.0

Como se pode verificar na tabela, à medida que os pedidos aumentam o debito também aumenta, no entanto a partir dos 800 verificou-se uma diminuição dos valores. Isto deveu-se ao facto de à medida que são feitos mais pedidos são também criadas novas réplicas da aplicação, o que vai compensar esse mesmo aumento. No entanto a partir das 1000 threads, como se pode verificar na imagem abaixo atingiu-se o limite de *pods* (10), ou seja, serão feitos cada vez mais pedidos com o mesmo número de réplicas o que levou à diminuição do débito.



1.2 CRIAR PÁGINA

	Nº Threads					
Tempo médio de resposta (ms)	**10**	**100**	**500**	**1000**	**8000**	**15000**
	116	200	779	2038	5123	-

	Nº Threads					
Abort Rate (%)	**10**	**100**	**500**	**1000**	**8000**	**15000**
	0.00%	0.00%	0.09%	3.64%	9.10%	-

	Nº Threads					
Throughput (requests/ sec)	**10**	**100**	**500**	**1000**	**8000**	**15000**
	84.6	491.3	596.7	437.0	90.8	-

Tal como se pode verificar pelos resultados obtidos, nesta página obtemos resultados com um comportamento bastante semelhante ao da página login. Por um lado, é possível observar que o *troughput* é maior nesta página atual, o que significa que o servidor tem maior capacidade para lidar com os pedidos que lhe são feitos.

Além disso, vê-se também que a taxa de erro é menor do que na página de login.

VII. CONCLUSÃO E TRABALHOS FUTUROS

A realização deste trabalho prático permitiu consolidar todo o conhecimento adquirido ao longo das aulas teóricas e práticas da unidade curricular.

O grupo considera que fez um bom trabalho na medida em que conseguiu automatizar bastante todo o processo de provisionamento e deployment da aplicação Wiki.js, além de ter tido em conta não só a necessidade de haver balanceamento de carga, mas também a capacidade da aplicação manter uma boa performance com o aumento do número de clientes.

Por outro lado, existem algumas melhorias que poderão ser implementadas em trabalhos futuros tais como utilizar outras ferramentas de avaliação como por exemplo Selenium , avaliar a aplicação recorrendo a mais métricas ou ainda utilizar as bases de dados disponibilizadas pela Google Cloud Platform (ou outros serviços interessantes).