Mestrado em Engenharia Informática

Aplicações e Serviços de Computação em Núvem
Grupo 20

Relatório Trabalho Prático

Aluno/a:	*Número:*
Ana Luísa Carneiro	PG46983
Ana Rita Peixoto	PG46988
Francisco José Vilão Peixoto	PG47194
João Carlos Peixoto Freitas	PG47850
Luís Miguel Pinto	PG47428





Janeiro 2022

Conteúdo

1. Introdução

Neste trabalho realizado no âmbito da UC de Aplicações e Serviços de Computação em Núvem, foi proposta a automatização da instalação da aplicação Wiki.js. Complementarmente, também foi efetuada a sua caracterização, análise, monitorização e avaliação.

Deste modo, o presente relatório pretende expor todo o trabalho efetuado, detalhando todos os passos que compuseram a sua realização e justificações das decisões tomadas pelo grupo.

Para a primeira etapa do trabalho, é possível observar qual a arquitetura e componentes do sistema, o seu padrão de distribuição, pontos de falha e operações críticas. A segunda fase do trabalho consistiu no *deployment* da aplicação com recurso ao *kubernetes* e aos serviços fornecidos pelo *Google Cloud Platform*, efetuando a instalação no menor número de passos possível. Por fim, a terceira e quarta fase do trabalho, dizem respeito às ferramentas de monitorização e avaliação, contemplando a análise das ferramentas mais adequadas, a sua instalação e também testes que simulem comportamentos de utilizadores, tudo isto de forma automática.

2. WikiJs

O Wiki.js é um mecanismo *wiki* (página *web* que geralmente permite sua alteração por qualquer pessoa que possua um *browser*) executado em Node.js e escrito em *JavaScript*. Trata-se de um *software* livre lançado sob a Licença Pública Geral Affero GNU, encontrando-se disponível como uma solução auto-hospedada ou através da instalação de "clique único" no mercado *DigitalOcean* e *AWS*.

3. Arquitetura e Componentes

A arquitetura do sistema segue essencialmente um padrão multi camada, ou seja, temos a camada de interface para os utilizadores, a camada de serviço da aplicação e finalmente uma camada responsável pela persistência dos dados. Em adição, conseguimos identificar ainda componentes relacionados com ferramentas de monitorização e *benchmarking* da aplicação. Assim, através da arquitetura permitimos a separação dos componentes e suas respetivas funcionalidades, ainda que as mesmas consigam comunicar e propagar o seu estado entre cada elas. Na figura seguinte encontra-se representada a arquitetura implementada, juntamente com os respetivos componentes que a caracterizam.

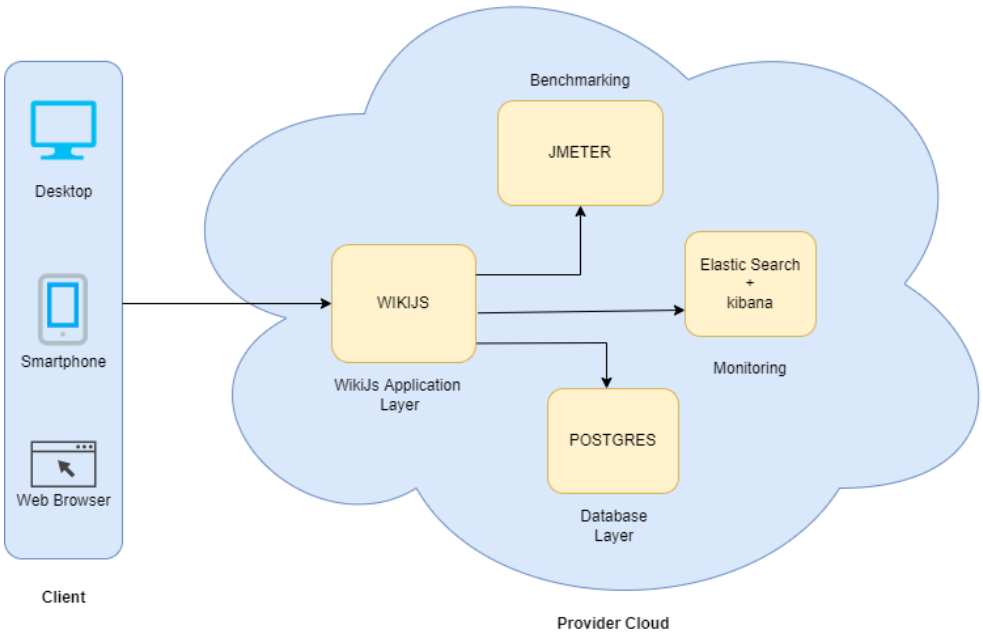

Figura 3.1: Arquitetura Implementada

O servidor da aplicação (*middleware*) recebe os pedidos provenientes dos utilizadores sendo a sua função dar resposta aos mesmos, podendo para isto usufruir dos dados presentes na base de dados caso seja necessário. Com o *deployment* desta aplicação visionamos permitir o acesso por um número elevado de clientes, sem comprometer a qualidade, disponibilidade e escalabilidade do serviço, de preferência sem nenhum ponto de falha. Dito isto, de forma a cumprir com os requisitos estipulados, para além dos componentes já mencionados, necessitamos de introduzir os seguintes componentes:

- **Elasticsearh** - motor de busca distribuído que permite fazer buscas de texto completo. Permite ainda, num sistema *cluster-based*, tornar as procuras na base de dados mais eficientes.
- **KIBANA** - *plugin* de visualização e exploração de dados para o Elasticsearch. Fornece recursos de visualização de conteúdo para um *cluster* Elasticsearch. Os utilizadores podem criar diversos gráficos sobre os dados.
- **JMETER** - é uma ferramenta de auxilio na realização testes de carga a sistemas computacionais.

Posteriormente, nas demais secções, iremos tornar a abordar estes últimos componentes, fornecendo um maior nível de detalhe acerca do seu propósito e da sua importância.

4. Componentes e Operações Críticas

Em aplicações distribuídas e desenvolvidas em contexto de *cloud*, é necessário ter em conta a existência de pontos de falha, ou seja, componentes que podem comprometer a viabilidade do sistema; e também operações cujo desempenho é crítico, isto é, operações que devem ser otimizadas de forma a minimizar o *delay* permitindo respostas em tempo útil aos pedidos feitos pelos utilizadores.

Tratando-se de uma aplicação distribuída, estas aplicações devem também ser capazes de suportar um grande número de utilizadores e é vital que mesmo que um componente do sistema falhe isso seja transparente aos olhos do utilizador.

4.1 Pontos Únicos de Falha

É importante identificar pontos únicos de falha (SPOFs) do sistema, de modo a prepara-lo para reagir a essas falhas. Dados os componentes existentes no sistema, podemos identificar os seguintes pontos de falha:

- **Servidor aplicacional:** é crucial para o sistema que exista pelo menos um servidor aplicacional ativo. No entanto, caso apenas exista um único servidor e este falhe, o sistema perde a capacidade de operar. De forma a mitigar este problema, é boa prática a utilização de vários servidores aplicacionais e de um *proxy* que atue como balanceador de carga.

- **Base de Dados:** tal como no servidor aplicacional, a existência de uma única instância da base de dados torna-se num ponto de falha e, para evitar este problema, devemos efetuar réplicas da base de dados por vários servidores.

4.2 Operações Críticas

Uma operação com desempenho crítico trata-se de qualquer operação que necessite de ser otimizada ao máximo para funcionar com *delay* mínimo.

Consideram-se operações com desempenho crítico :

- **Pesquisa Global** - numa primeira vista pode parecer uma operação trivial e insignificante. No entanto, quando temos em conta a escalabilidade da solução, faz sentido que num grande volume de dados possa ser um problema. Quando temos uma aplicação como o WikiJS, onde são criados e editados milhares de ficheiros diariamente, é fundamental que o sistema de pesquisa seja rápido o suficiente para encontrar um ficheiro em tempo útil. Esta é também, para nós, uma operação com desempenho crítico, uma vez que nenhum utilizador gosta de ficar muito tempo à espera de uma resposta à sua pesquisa.
- **Armazenamento Ficheiros** - O armazenamento de ficheiros é uma operação fundamental para o bom funcionamento da aplicação, pois é a base do sistema. Desta forma, caso haja falha nesta componente o utilizador não vai conseguir armazenar nenhum ficheiro. Uma forma de contornar este problema poderá passar por acrescentar redundância ao sistema de ficheiros. Numa aplicação que lida com tantos ficheiros, isto permitirá reduzir os *downtimes* provocados pela falha deste serviço, bem como aumentar a disponibilidade do mesmo, tornando o serviço mais eficiente no acesso aos ficheiros. Assim sendo, os componentes envolvidos nesta operação que podem limitar mais o seu desempenho envolvem a **base de dados** e o **servidor aplicacional**.

5. Padrões de distribuição

Um padrão de distribuição é um esquema que visa a distribuição de componentes e serviços por várias entidades de forma a implementar uma aplicação a ser usada por utilizadores de forma remota. Com estes padrões permitimos acomodar vários utilizadores através de técnicas de replicação de componentes permitindo aumentar o nível de disponibilidade e escalabilidade do serviço. Na seguinte imagem encontra-se a distribuição das componentes implementadas neste projeto assim como as ferramentas que foram utilizadas para essa implementação - *Kubernetes* e *Docker*.

Para a criação do padrão que podemos ver na imagem 5.1 utilizou-se o *Kubernetes da Google Cloud* (GKE) que nos permitiu a criação de um *cluster* de três máquinas virtuais, uma que corresponde ao *master* que vai atribuir a carga de trabalho entre as restantes

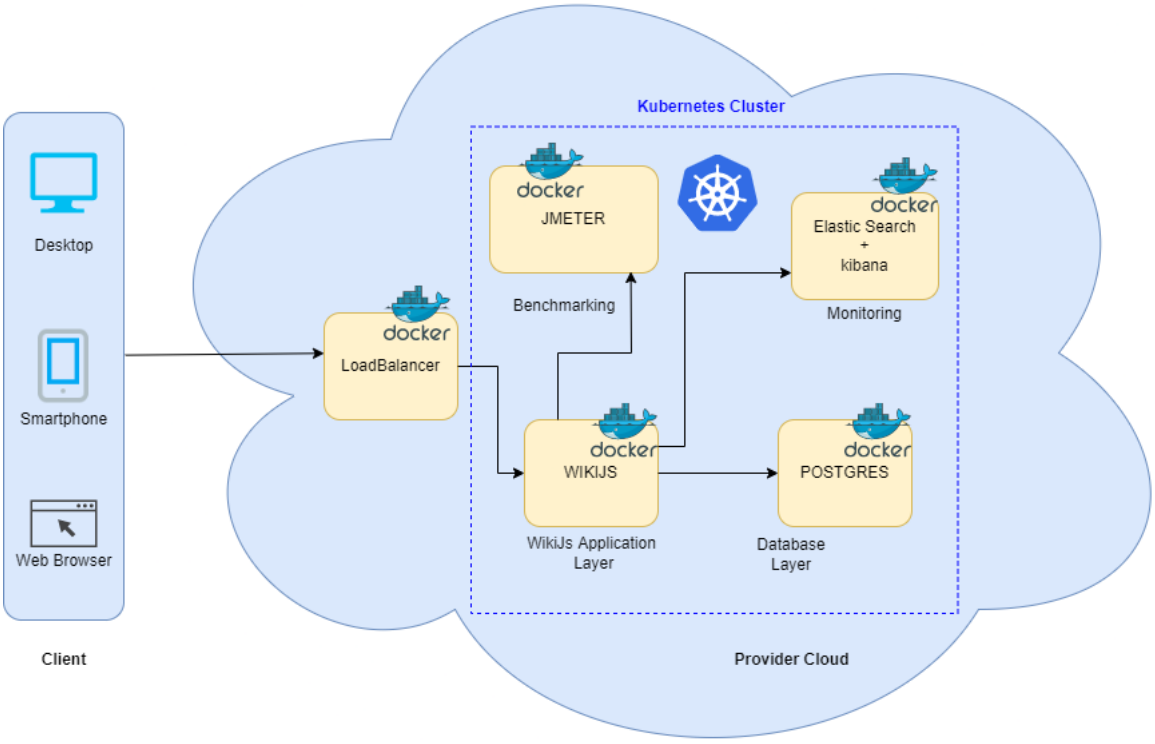

Figura 5.1: Padrão Distribuição com *Docker* e *Kubernetes*

máquinas a qual designamos por *workers*. Nos *workers* foram criados *containers* onde instalamos componentes do projeto como *Wikijs* e *Postgres*. Esta criação de *containers* independentes teve como objetivo isolar cada componente de forma a garantir uma gestão eficiente e organizada dos mesmos.

O serviço ***wikijs*** tem como objetivo fornecer a base aplicacional do projeto e por isso há necessidade de garantir a disponibilidade deste, ou seja, é necessário que caso haja aumento do número de utilizadores na aplicação esta consiga responder a todos os pedidos. Para além disso é necessário que caso haja falha na base aplicacional exista um serviço *backup* que responda aos pedidos dos utilizadores. Para que se permita a escalabilidade e disponibilidade da aplicação então é necessário a replicação da componente uma vez que desempenha um serviço crítico.

O serviço ***postgres*** tem como objetivo fornecer a camada de dados de apoio à base aplicacional e por isso é igualmente necessário garantir alta disponibilidade e escalabilidade da aplicação. Desta forma, é necessário a replicação da componente já que também este é um serviço crítico e por isso caso haja falha numa base de dados exista uma outra replicada de *backup* .

O serviço de **monitorização** (*elasticsearch* ou *kibana*) como não é uma componente essencial para o sistema não necessita de replicação pois não há necessidade de se este estar sempre disponível. O mesmo acontece para o serviço de ***benchmarking*** (JMeter) pois não apresenta operações de desempenho crítico.

Com a criação do *cluster* a partir do *kubernetes* permitimos mais facilmente a replicação de serviços entre os *workers* uma vez que o *master* vai distribuir internamente a carga de trabalho entre as máquinas. Desta forma permitimos uma aplicação escalável e confiável.

De forma a que haja **balanceamento de carga** entre as várias replicações dos serviços críticos então cria-se um *load balancer* que vai balancear os pedidos do utilizadores para as várias replicações. Desta forma conseguimos manter a escalabilidade e performance do sistema pois os pedidos serão reencaminhados para a replicação do serviço que tiver com menos carga. Com a utilização do GKE para a criação do *cluster* o balanceamento de carga já se encontra feito automaticamente através da criação de um serviço do tipo *LoadBalancer* para fazer o *deployment* do *wikijs*.

6. Automatização

A automatização do processo de *deployment* da aplicação *wikijs* utilizou as ferramentas GKE (*Google Kubernetes Engine*) e GCP (*Google Cloud Platform*), através de *playbooks* e da linguagem *ansible*. Nas seguintes secções encontra-se a explicação de todo o processo de instalação e *deployment* das várias componentes utilizadas no projeto desde a criação do *cluster* até à instalação das ferramentas de *benchmarking* e monitorização passando pelo *deployment* da aplicação *Wikijs*.

A utilização da ferramenta GKE deve-se à sua simplicidade a implementar outros parâmetros do trabalho, como o *LoadBalancer* e as ferramentas de monitorização e *benchmarking*.

6.1 Criação do *Cluster Kubernetes*

A primeira etapa do trabalho consistiu na criação de um *cluster kubernetes* com um número arbitrário de nodos. Para este trabalho consideraram-se 3 nodos, para que o *kubernetes* possa distribuir a carga e efetuar o seu balanceamento. Ao efetuar a criação de uma instância *cluster* no *google cloud*, são criadas também máquinas virtuais automaticamente que suportam os nodos criados aquando do *cluster*.

Para isso, recorreu-se a um ficheiro *playbook* que utiliza o módulo *gcp_container_cluster* para efetuar a criação do *cluster* e recorreu-se ao módulo *gcp_container_node_pool* para efetuar a criação dos nodos associados ao *cluster* anterior.

6.2 Instalação do *Wikijs* e *Postgres*

Para o *deployment* remoto e automatizado do *Wikijs* e da base de dados *Postgres* foi utilizado três ficheiros:

- ***postgres-pv.yaml***: Neste ficheiro encontra-se a implementação da criação do

volume de dados que vai ser usado para persistir os dados criados com a aplicação *wikijs*. Neste caso estamos atribuir 5 Gi de volume para armazenar e persistir os dados.

- ***postgres-deployment.yml***: Este ficheiro tem como objetivo fazer a implementação da base de dados *postgres* que vai ser utilizada pela aplicação *wikijs* através da porta 5432. Esta BD também vai utilizar o volume de dados que foi persistido no ficheiro *postgres-pv.yaml* e um conjunto de parâmetros de forma a criar o utilizador e correspondente *password*, assim como o nome da BD que vai armazenar os dados da aplicação.

- ***wikijs-deployment.yml***: Neste ficheiro encontra-se toda a instalação da aplicação *Wikijs*. Para isso criou-se um serviço do tipo *LoadBalancer* que permite a exportação da aplicação para o exterior do *cluster*, através da porta 3000. Para o *deployment* da aplicação foi necessário a conexão com a BD *postgres* criada com o ficheiro *postgres-deployment.yml* utilizando a porta, utilizador e password definidas nesse mesmo ficheiro.

Os três ficheiros foram implementados remotamente no *cluster* recorrendo ao *kubectl*. Esta instalação pode ser efetuada a partir do *localhost* ou de uma máquina virtual da GCP, desde que seja feita a conexão com o *google cloud* com recurso a comandos *gcloud*. Desta forma conseguiu-se fazer a instalação tanto da aplicação como da componente de base de dados a partir do acesso remoto ao *cluster kubernetes*.

Para que seja possível a exportação da aplicação para o exterior da VM foi necessário a criação de uma regra de *firewall* para abrir a porta 3000 a todos os IPs.

6.3 Criação das VMs

Caso queiramos criar o *cluster* de *kubernetes* a partir de uma VM em vez do *localhost* podemos criar de forma automática uma instância de máquina virtual na *google cloud*. Para isso, procedeu-se à elaboração de um *playbook* que efetua a criação de um número arbitrário de VMs conforme as configurações que forem indicadas (tipo de máquina, imagem, zona, projeto e tamanho do disco).

É de realçar que para a criação das VMs foi necessário recorrer ao módulo *gcp_compute _instance* do *Ansible*. Além disso, de forma a conseguir efetuar a criação das VMs na plataforma da *Google Cloud*, foi necessário ter em consideração a chave do projeto de ASCN obtidas na *google console*, para fornecer acesso ao projeto.

Todo este processo de criação das VMs foi feito remotamente utilizando um *host* externo à *Google Cloud*.

6.4 Instalação da Monitorização

Para o processo de monitorização foi utilizado o serviço do *Elasticsearch*. Para a instalação desta ferramenta, foram utilizados *scripts* que efetuam o seu *deployment* de

forma automática sobre o *cluster* do GKE, que contém as instâncias monitorizadas.

7. Monitorização e Métricas

De forma a monitorizar o comportamento e os recursos da aplicação recorremos aos serviços da *Elastic Cloud* e da *Google Cloud*.

Na figura seguinte destacamos as diferentes métricas escolhidas para monitorização na *dashboard* da ferramenta utilizada.

7.1 Ferramentas Utilizadas

Elasticsearch serve como motor de busca e análise de dados, de forma a permitir efetuar pesquisas eficientes sobre os dados a explorar.

Kibana permite analisar e explorar o comportamento dos recursos em tempo real através de uma *dashboard* configurável, onde temos acesso a uma interface gráfica que facilita bastante o processo de análise.

FileBeats permite coleccionar e enviar vários tipos de *logs* de diferentes servidores para o *Elasticsearch*, possibilitando assim analisar *logs* de vários servidores de forma centralizada.

Metrics Explorer - *Google Cloud Platform* apresenta ferramentas do *Cloud Monitoring* para visualizar e monitorizar dados sobre serviços e gestão de recursos (CPU,RAM,I/O) que estão a ser utilizados.

7.2 Métricas Utilizadas

Métricas de Performance do Sistema correspondem a dados relativos à utilização de CPU, memória, utilização de disco, *requests* ao serviço e ainda o tempo de resposta do mesmo. Estas métricas são importantes de modo a identificar possíveis pontos de falha, de modo a permitir uma melhor gestão dos recursos, que se pode refletir em modificações na arquitetura por parte dos administradores. Na figura 7.1 podemos estas métricas aplicadas ao *cluster* de GKE criado no projeto.

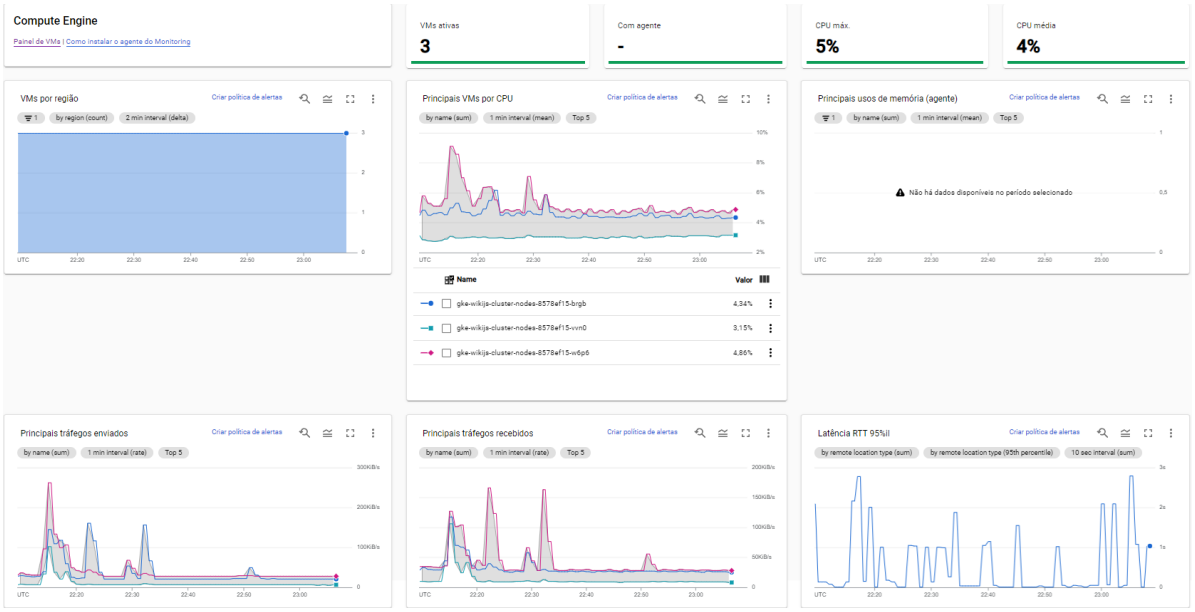

Figura 7.1: Monitorização - Gestão de Recursos (utilização do CPU)

Métricas de Performance da Aplicação Nesta secção utilizamos a *framework Filebeat* que nos permite analisar os registos de tudo o que se passa por de trás dos *nodes* que compõem o *cluster engine*. Através destes *logs* é possível, entre outros, obter o número de pedidos que cada um dos *nodes* da aplicação recebe conforme ilustrado na Figura-7.2.

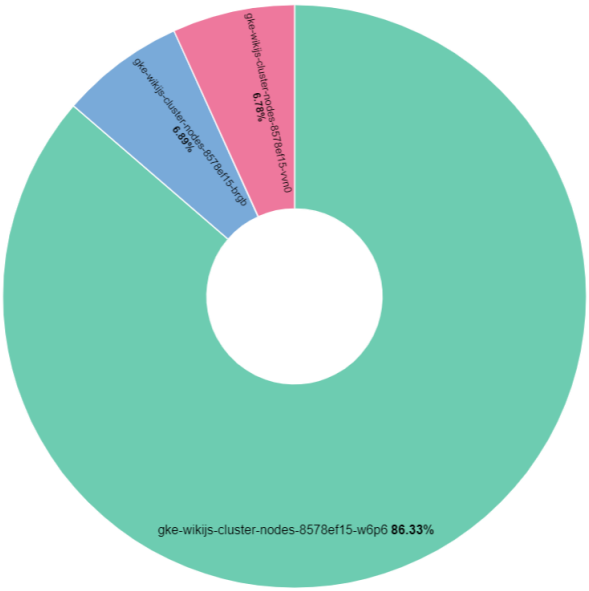

Figura 7.2: Carga de trabalho dos *nodes* da aplicação

8. Avaliação e Testes

Para efetuar avaliação e testes do *WikiJs* recorreu-se à ferramenta **BlazeMeter**, que facilitou bastante a gravações dos testes usando o *browser*, visto que permite simular e gravar todos os passos necessários a registar nos ficheiros que irão ser usados pelo **JMeter**.

Recorrendo à ferramenta **JMeter** pretende-se avaliar diversas ações no contexto da aplicação, nomeadamente o *log in* e a criação de uma entrada de texto na plataforma do *WikiJs*.

A ferramenta permite a realização de testes de performance, que simulam operações sobre o sistema que têm um maior custo no desempenho, visto simular uma grande afluência de Utilizadores (*Threads*) na aplicação.

Fazendo recurso ao *JMeter* que permite definir o número de clientes usados nos testes, executam-se as operações pré-definidas. Após os testes é gerada uma *dashboard*, onde se observa as diferentes métricas avaliadas (tempo de resposta, pedidos executados, etc).

8.1 Teste 1 - *Homepage*

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**800**	**0**	**0.00%**	**4126.94**	**244**	**19194**	**1667.50**	**13260.00**	**14996.95**	**18053.07**	**41.04**	**23505.95**	**28.98**
http://35.223.72.104:3000/	100	0	0.00%	12975.67	5033	19194	13416.00	17838.60	18609.35	19193.42	5.13	11752.98	14.49
http://35.223.72.104:3000/-0	100	0	0.00%	1787.06	428	2883	1746.00	2754.80	2817.40	2882.56	26.29	82.28	9.83
http://35.223.72.104:3000/-1	100	0	0.00%	1205.74	304	2509	1193.00	1839.50	2056.55	2508.16	17.31	168.76	7.20
http://35.223.72.104:3000/-2	100	0	0.00%	1275.42	257	2357	1275.00	1858.40	2207.55	2356.51	22.36	49.05	9.11
http://35.223.72.104:3000/-3	100	0	0.00%	1279.40	244	2353	1283.00	1865.90	2206.65	2352.55	22.88	33.16	9.32
http://35.223.72.104:3000/-4	100	0	0.00%	1960.90	605	6894	1854.00	2746.70	3332.90	6881.18	11.37	724.36	4.70
http://35.223.72.104:3000/-5	100	0	0.00%	1344.33	294	2689	1351.50	1950.50	2277.50	2685.90	16.72	89.23	6.78
http://35.223.72.104:3000/-6	100	0	0.00%	11187.00	2279	17736	11763.50	15265.90	17006.85	17735.81	5.25	11583.93	2.11
Test	100	0	0.00%	12975.67	5033	19194	13416.00	17838.60	18609.35	19193.42	5.12	11727.71	14.46

Figura 8.1: Resultado de conexão à *homepage* por parte de 100 *Threads*.

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**1600**	**0**	**0.00%**	**7986.72**	**477**	**32708**	**3355.00**	**24519.80**	**26727.75**	**29791.30**	**48.78**	**27938.27**	**34.44**
http://35.223.72.104:3000/	200	0	0.00%	24830.70	11696	32708	25326.50	28953.60	30441.25	32267.12	6.10	13969.13	17.22
http://35.223.72.104:3000/-0	200	0	0.00%	2965.31	477	4513	3068.50	4238.70	4391.05	4500.82	36.72	114.92	13.73
http://35.223.72.104:3000/-1	200	0	0.00%	2299.92	617	4964	2247.50	3932.60	4240.40	4948.34	21.91	213.61	9.12
http://35.223.72.104:3000/-2	200	0	0.00%	2743.87	810	4559	2735.00	4108.60	4324.50	4536.73	24.72	54.23	10.07
http://35.223.72.104:3000/-3	200	0	0.00%	2743.54	1039	4733	2671.00	4127.20	4327.15	4557.79	26.65	38.62	10.85
http://35.223.72.104:3000/-4	200	0	0.00%	3612.40	1376	7172	3561.50	4938.40	5503.45	6900.92	18.61	1185.18	7.69
http://35.223.72.104:3000/-5	200	0	0.00%	2834.12	1039	5460	2776.50	4207.20	4468.10	5250.45	21.78	116.20	8.83
http://35.223.72.104:3000/-6	200	0	0.00%	21863.93	10772	30698	22557.50	26345.60	28381.00	30372.61	6.19	13659.51	2.49
Test	200	0	0.00%	24830.71	11696	32708	25326.50	28953.60	30441.25	32267.12	6.09	13945.33	17.19

Figura 8.2: Resultado de conexão à *homepage* por parte de 200 Threads.

8.2 Teste 2 - *Login* e Criar um *Post* no *WikiJs*

Os testes relativos ao *log in* na plataforma bem como a criação de um novo *post* no *WikiJs* foram realizados para 100, 200 e 300 Utilizadores, todas as informações retiradas dos testes encontram-se em anexo.

Primeiramente vão ser apresentadas as estatísticas para 200 e 300 Utilizadores e consegue-se perceber que a taxa de erro é extremamente baixa.

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**8596**	**2**	**0.02%**	**1616.34**	**15**	**30640**	**283.00**	**2972.30**	**4864.15**	**24940.42**	**38.12**	**4116.37**	**32.72**
Test	200	2	1.00%	26712.45	14496	34077	26765.00	30935.10	31651.60	33319.39	5.84	13647.19	135.86
http://35.223.72.104:3000/logout	200	2	1.00%	415.67	15	628	441.50	460.00	470.90	601.35	1.29	1.04	1.03
http://35.223.72.104:3000/login	200	0	0.00%	22977.70	9509	30640	23189.00	27402.30	28133.85	29869.36	6.51	14905.16	18.41
http://35.223.72.104:3000/	200	0	0.00%	603.28	567	1636	594.00	618.90	634.60	684.67	7.17	38.96	23.91
http://35.223.72.104:3000/login-6	200	0	0.00%	20781.12	6620	29433	21332.00	25130.70	26391.25	28523.33	6.62	14589.30	2.66
http://35.223.72.104:3000/e/en/home	200	0	0.00%	590.20	569	683	586.00	606.90	617.00	654.85	5.49	24.55	18.34
http://35.223.72.104:3000/login-5	200	0	0.00%	2715.21	990	5093	2864.00	3836.90	4008.05	4881.79	29.58	157.82	11.99
http://35.223.72.104:3000/login-4	200	0	0.00%	3707.92	1385	10729	3738.50	5140.80	5627.70	6933.54	17.79	1132.89	7.35
http://35.223.72.104:3000/login-3	200	0	0.00%	2555.78	1006	4428	2693.50	3741.10	3911.10	4124.72	35.34	51.22	14.39
http://35.223.72.104:3000/en/home-0	200	0	0.00%	306.07	288	390	302.00	320.00	329.00	386.76	3.87	12.13	1.47
http://35.223.72.104:3000/en/home-1	200	0	0.00%	234.03	133	361	276.00	287.00	291.95	357.78	3.87	1.49	1.94
http://35.223.72.104:3000/en/home-2	200	0	0.00%	224.47	135	341	275.00	286.00	290.95	311.92	3.87	1.48	1.90
http://35.223.72.104:3000/login-2	200	0	0.00%	2578.43	991	4567	2719.50	3702.10	3914.90	4368.10	30.46	66.81	12.40
http://35.223.72.104:3000/en/home-3	200	0	0.00%	260.01	135	361	280.00	290.90	299.85	360.80	3.87	1.48	1.90
http://35.223.72.104:3000/login-1	200	0	0.00%	2209.80	882	4096	2303.00	3538.80	3650.85	3876.91	35.66	347.68	14.84
http://35.223.72.104:3000/en/home-4	200	0	0.00%	278.51	139	361	280.00	289.90	298.90	360.83	3.87	1.49	1.93
http://35.223.72.104:3000/login-0	200	0	0.00%	2195.43	465	3498	2421.00	3307.60	3410.50	3483.92	45.20	97.24	17.13
http://35.223.72.104:3000/en/home-5	200	0	0.00%	279.18	139	361	280.00	292.00	300.80	357.83	3.88	1.49	1.90

Figura 8.3: Estatísticas relativas a 200 *Threads*.

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**12894**	**3**	**0.02%**	**2128.05**	**3**	**39512**	**284.00**	**4201.50**	**7473.50**	**33422.35**	**91.51**	**9885.96**	**78.55**
Test	300	3	1.00%	34508.16	18246	42967	35054.00	39343.90	40508.05	42036.73	6.96	16272.24	161.87
http://35.223.72.104:3000/logout	300	3	1.00%	420.04	3	605	442.00	473.90	505.70	558.93	3.96	3.18	3.15
http://35.223.72.104:3000/login	300	0	0.00%	30778.52	12773	39512	31439.50	35925.80	36797.15	38535.14	7.59	17374.59	21.46
http://35.223.72.104:3000/	300	0	0.00%	624.95	571	2800	593.00	664.00	718.95	1533.61	9.03	49.03	30.09
http://35.223.72.104:3000/login-6	300	0	0.00%	27802.72	9626	39117	28345.00	33531.40	34652.10	36040.51	7.67	16908.32	3.08
http://35.223.72.104:3000/e/en/home	300	0	0.00%	605.26	560	798	591.50	640.80	707.70	789.97	7.53	33.69	25.16
http://35.223.72.104:3000/login-5	300	0	0.00%	3833.93	1412	11795	3754.50	5439.00	5755.20	6763.70	21.59	115.18	8.75
http://35.223.72.104:3000/login-4	300	0	0.00%	5298.32	1917	13307	5078.00	7644.20	8927.85	11896.73	19.77	1258.77	8.16
http://35.223.72.104:3000/login-3	300	0	0.00%	3667.27	1074	8268	3561.50	5316.00	5595.90	6678.44	30.20	43.77	12.30
http://35.223.72.104:3000/en/home-0	300	0	0.00%	314.39	287	531	304.00	349.90	377.95	435.75	4.66	14.61	1.77
http://35.223.72.104:3000/en/home-1	300	0	0.00%	206.26	134	347	157.00	287.90	299.00	336.97	4.67	1.79	2.34
http://35.223.72.104:3000/en/home-2	300	0	0.00%	251.27	134	334	278.50	297.00	304.90	324.94	4.66	1.78	2.29
http://35.223.72.104:3000/login-2	300	0	0.00%	3684.98	1078	8312	3662.00	5322.20	5607.30	8233.70	30.20	66.24	12.30

Figura 8.4: Estatísticas relativas a 300 *Threads*.

Com a realização dos testes para diferentes números de Utilizadores conseguiu-se compreender que não há uma degradação significativa no Tempo de Resposta com o aumento de Utilizadores. Com 300 Utilizadores a percentagem de erros em relação a 100 Utilizadores é bastante similar.

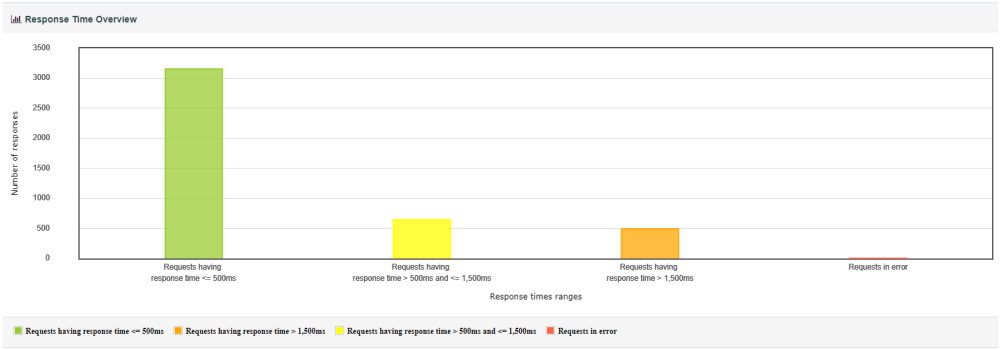

Figura 8.5: Tempo de Resposta relativo a 100 *Threads*.

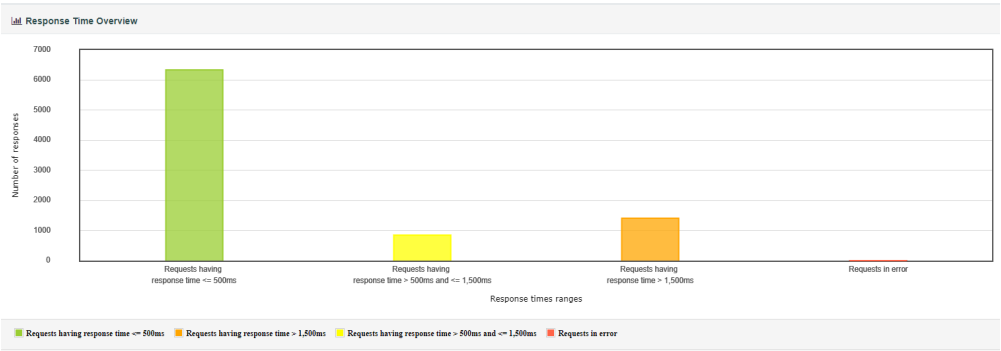

Figura 8.6: Tempo de Resposta relativo a 200 *Threads*.

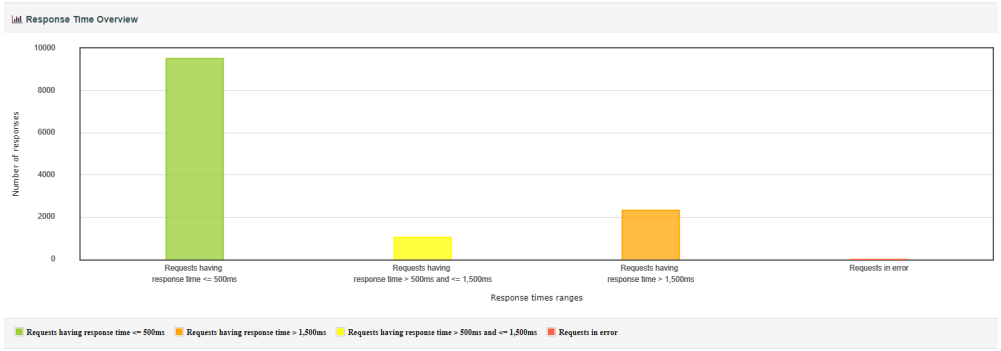

Figura 8.7: Tempo de Resposta relativo a 300 *Threads*.

8.3 Monitorização durante os Testes

De forma a conseguir entender melhor o custo na performance que se obteve através dos testes, apontou-se duas categorias de monitorização de performance que demonstram que a aplicação, mesmo no teste com 300 Utilizadores, não ultrapassou os 20% de Utilização de CPU.

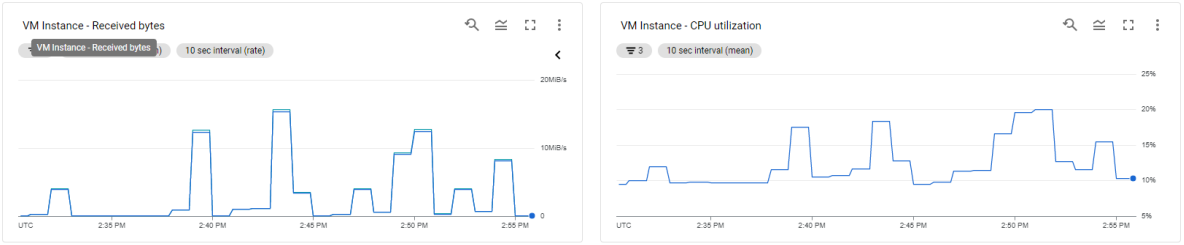

Figura 8.8: Dados relativos à monitorização durante a realização dos testes.

9. Conclusão

Dado por concluído o trabalho prático, fará sentido apresentar uma visão crítica, refletida e ponderada do trabalho realizado.

No espetro positivo, consideramos relevante destacar o facto do *deployment* do *WikiJs* estar funcional e operacional, recorrendo ao uso de funcionalidades do *google cloud* como o GKE, e considerar funcionalidades extra como o balanceamento de carga através de um *LoadBalancer*.

Por outro lado, em termos de melhorias ou *upgrades* no nosso projeto, consideramos que seria benéfico implementar hierarquia na base de dados, considerando uma base de dados primária que estaria a suportar o programa e as secundárias para dar resposta, caso a primária falhe.

Em suma, consideramos que o balanço do trabalho é positivo, as dificuldades sentidas foram superadas e os requisitos propostos foram cumpridos.