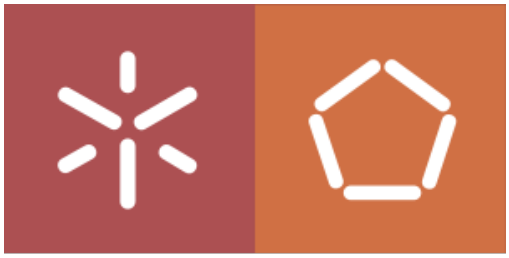

Universidade do Minho
Escola de Engenharia

Mestrado em Engenharia Informática

Perfil Engª de Aplicações

Infraestrutura de Centro de Dados

Trabalho Prático

1º Ano, 1º Semestre

Ano letivo 2020/2021

Diogo Lopes

PG42823

Fábio Gonçalves

PG42827

Joel Carvalho

PG42837

Diogo Ferreira

PG42824

Conteúdo

Lista de Figuras

Siglas e Acrónimos

DRBD - Distributed Replicated Block Device

iSCSI - Internet Small Computer System Interface

IP - Internet Protocol

HAC - High Availability Cluster

VM - Virtual Machine

1. Introdução

O desenvolvimento deste projeto recai, na integra, no âmbito dos conteúdos abordados na Unidade Curricular de Infraestrutura de Centro de Dados. As infraestruturas, necessitam de cumprir vários requisitos desde a solidez até à elevada disponibilidade e escalabilidade. Caso algum destes fracasse, isto poderá provocar múltiplas mazelas, por vezes algumas delas irreparáveis.

Com o propósito de garantir esses requisitos, o principal objetivo da implementação deste Trabalho Prático consiste no planeamento, configuração, análise de desempenho e operações de infraestruturas de elevada disponibilidade com a aplicação Wiki.js. Ao longo do desenvolvimento deste trabalho, identificamos os pontos críticos de falha e desempenho, descrevemos a arquitetura implementada e concluímos com a avaliação do desempenho da arquitetura proposta e das políticas de balanceamento e tolerância a faltas utilizadas.

1. Objetivos

O Plano de Trabalho para este para a elaboração deste Projeto resume-se essencialmente aos três tópicos seguintes:

- Planear e Operacionalizar uma instalação da plataforma *Wiki.js*;
- Garantir alta disponibilidade e balanceamento de carga;
- Avaliar o desempenho da instalação, realizando testes de carga e contemplar o impacto dos mecanismos de balanceamento e alta disponibilidade introduzidos.

2. Descrição da Arquitetura e Componentes da Aplicação *Wiki.js*

Wiki.js é um motor wiki totalmente personalizável e modular, escrito inteiramente em *JavaScript*. Possui um conjunto de ferramentas que permite uma integração bastante rápida com qualquer sistema existente. Posto isto, a aplicação encontra-se dividida em *frontend* e *backend* com a possibilidade de utilizar diferentes bases de dados, tendo sido utilizada neste trabalho prático, a base de dados PostgreSQL para o armazenamento dos dados. O *backend* foi desenvolvido em *Node.js* com recurso à *framework Express.js*, nomeadamente desenvolvido em *JavaScript*. O *frontend* foi desenvolvido recorrendo à *framework Vue.js*.

Características

1. Performance - Incrivelmente rápido, o *Wiki.js* é construído tendo o desempenho em mente;
2. Customizável - Totalmente editável, como a aparência, incluindo um modo claro e escuro;
3. Disponibilidade - *Wiki* público, completamente privado ou uma mistura de ambos;
4. Escalável - Desde *Raspberry Pi* a uma VM de alto desempenho na *Cloud*, o *Wiki.js* faz uso inteligente dos recursos disponíveis.

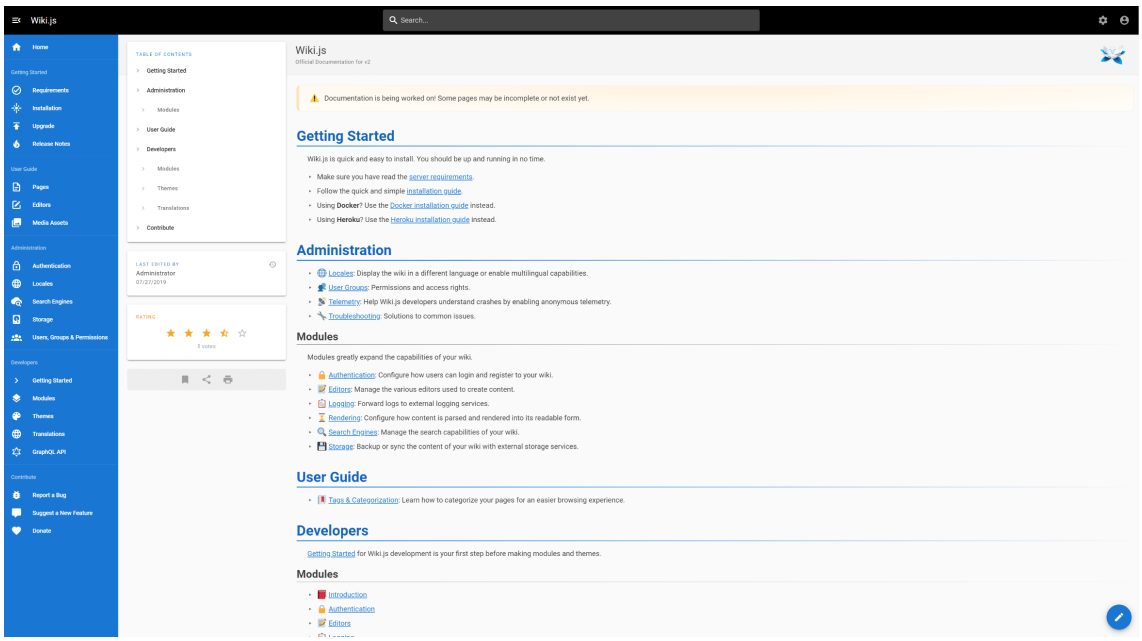

Figura 1 Página Inicial Wiki.js

3. Pontos Críticos de Falha e Desempenho

Qualquer serviço que apresente apenas uma réplica de cada serviço, representa um possível ponto de falha, uma vez que quando qualquer um destes serviços fica indisponível, a aplicação fica igualmente indisponível para os utilizadores. Assim sendo, todos os componentes são pontos críticos de falha, servidores *Web*, servidores aplicacionais e a base de dados.

Tendo em conta o tipo de aplicação estudada, e o facto de esta apresentar um possível *bottleneck* na base de dados, dado que todas as operações efetuadas necessitam desta, é possível perder acesso às informações lá contidas, ou até termos um acesso demasiado lento tornando assim a aplicação inutilizável.

4. Arquitetura Implementada

1. Armazenamento

Para garantirmos armazenamento de informação de alta disponibilidade e distribuído, utilizamos o serviço de replicação DRBD em duas máquinas em modo primário para realizarmos a replicação dos dados. Nestas máquinas implementamos também iSCSI multipath de modo a facilitar o acesso a estas, e em caso de *failover*, dado que as duas máquinas correm como primárias automaticamente selecionar a máquina que ainda se encontra disponível.

2. Cluster Base de Dados

De modo a garantirmos uma base de dados PostgreSQL de alta disponibilidade implementamos um *High Availability Cluster* muito semelhante ao apresentado no guião 5 das aulas práticas. Este cluster é constituído por dois nós que utilizam as ferramentas *iSCSI* e *multipath* para aceder às duas máquinas referidas na secção anterior. Neste *cluster* implementámos apenas dois recursos, um relacionado com o *FileSystem* e outro relacionado com a própria base de dados PostgreSQL. É aqui que acaba a semelhança com o guião prático, dado que a plataforma Google Cloud não suporta IPs virtuais que foram utilizados na implementação desenvolvida nas aulas práticas. De tal modo que para substituir este recurso utilizamos outra ferramenta disponibilizada pela própria plataforma, *Failover for Internal TCP/UDP Load Balancing*.

3. Load Balancing for Failover

Tal como foi referido anteriormente a Google Cloud Platform não suporta IP's virtuais, assim a alternativa encontrada pelo nosso grupo para fazer o redirecionamento do tráfego quando o HAC da base de dados é migrado por alguma razão, foi implementar *Failover for Internal TCP/UDP Load Balancing,* uma ferramenta disponibilizada pela GCP.

Este recurso funciona com base em grupos de instâncias (grupos de máquinas virtuais) e em *healthchecks*. Temos um grupo principal para o qual o tráfego será redirecionado normalmente e um grupo *failover* que funcionará como *backup* para o grupo principal. No nosso caso em particular nos dois grupos de instâncias apenas temos uma máquina, como tal não utilizaremos as funcionalidades de *load balancing* oferecidas por esta ferramenta. Este recurso avalia o estado das máquinas através de *healthchecks*, no nosso caso são pedidos TCP à porta 5432, a porta utilizada pela base de dados *PostgreSQL*, conforme as respostas que recebe considera uma VM saudável ou não. No nosso caso em particular em qualquer momento apenas uma das duas máquinas é considerada saudável

pois temos a base de dados a correr em apenas uma máquina, nunca nas duas. Quando a máquina principal por qualquer razão se torna indisponível, por breves momentos enquanto o *cluster* faz a migração dos recursos para outra máquina, nenhuma VM é considerada saudável e só apenas quando recebe resposta aos *healthchecks* é que começa a direcionar o tráfego para essa máquina. A plataforma continua sempre a enviar *healthchecks* a todas as máquinas e quando a máquina no grupo principal se torna disponível volta a redirecionar o tráfego para esta.

4. Servidores Aplicacionais

Com o objetivo de obtermos alta disponibilidade e consequentemente também melhor performance na camada aplicacional, implementamos dois servidores aplicacionais. Estes comunicam com a base de dados através do *load balancer* referido anteriormente.

5. Servidores Web

Tal como na camada anterior a maneira de obter alta disponibilidade nesta camada foi ter duas réplicas do serviço que fazem *load balancing* do tráfego para os servidores aplicacionais.

6. *Load Balancing*

Dado que temos mais do que um servidor web, implementamos um *global load balancer* que redireciona e faz o balanceamento do tráfego da internet para as nossas instâncias Nginx. Esta é uma ferramenta fornecida pela GCP e como tal é gerido pela plataforma que assegura alta disponibilidade e garante que em caso de falha é reiniciado ou substituído.

7. Arquitetura das Máquinas Virtuais

Características	DRBD_1 DRBD_2	vm-postgres1vm- postgres2	wiki-inst1 wiki-inst2	nginx-inst-1 nginx-inst-2
Tipo	*e2-medium*	*e2-medium*	*e2-medium*	*e2-small*
CPU	2	2	2	2
Memória	4 GB	4 GB	4 GB	2 GB
Sistema Operativo	centos-8	centos-8	ubuntu-2004	ubuntu-2004

Tabela 1 Arquitetura das Máquinas Virtuais

8. *Overview* do Sistema

Com o intuito de realizar todos os objetivos propostos, o planeamento da estrutura baseou-se no seguinte esquema.

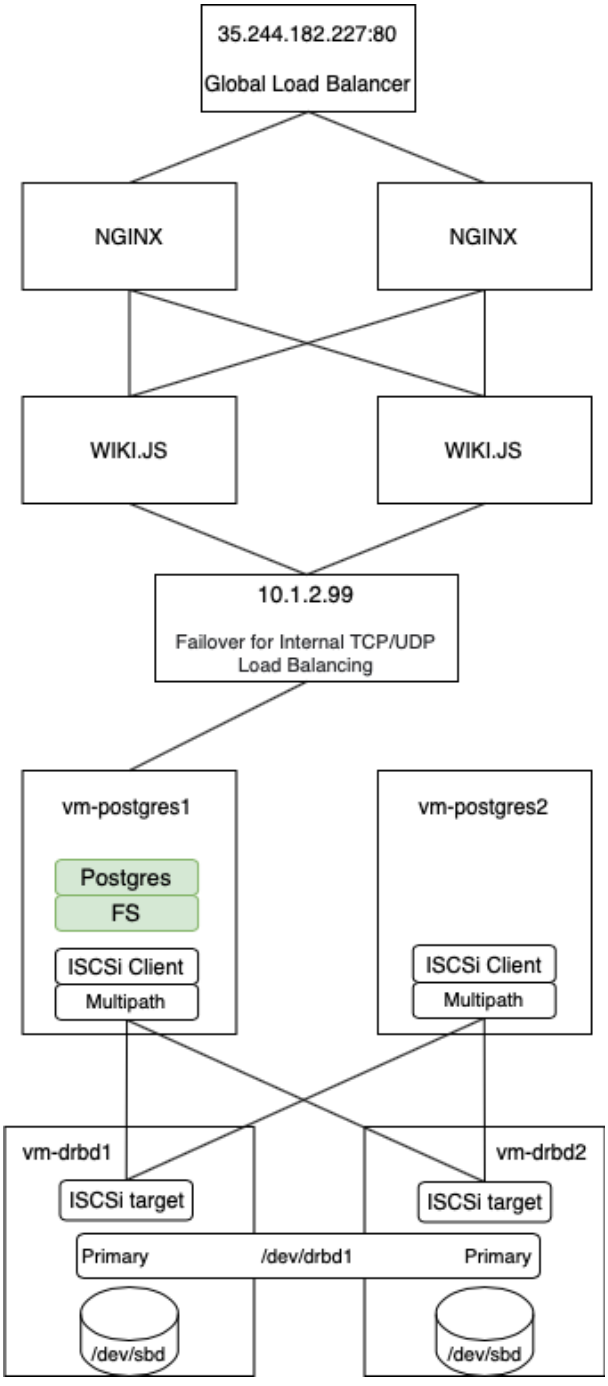

Figura 2 Overview do Sistema

Tal como podemos observar na figura anterior os pontos críticos de falha anteriormente referidos foram todos eliminados. Temos mais que uma réplica dos

servidores web e aplicacionais e como tal a falha de uma máquina de qualquer destes serviços não tornam a aplicação inutilizável como anteriormente, o *High Availability Cluster* e a replicação de dados feita pela ferramenta DRBD garantem que a base de dados se encontra sempre acessível. Estas mudanças além de garantir alta disponibilidade aumentam também a performance do sistema.

Na arquitetura sugerida fazemos balanceamento de carga em dois níveis, tal como foi dito anteriormente o *global load balancer* faz balanceamento da carga externa pelos dois servidores Nginx e estes por sua vez fazem balanceamento de carga pelos dois servidores aplicacionais.

A solução atual permite um fácil escalonamento dos servidores web e aplicacionais, mas tal escalonamento pode criar um *bottleneck* na base de dados visto que apenas temos uma instância a correr. Como tal, para aumentar o número de servidores web e aplicacionais e evitar este problema a base de dados teria de ser replicada.

5. Avaliação do desempenho da arquitetura proposta e das políticas de balanceamento e tolerância a faltas utilizadas.

Para realizarmos a avaliação do desempenho desta arquitetura recorremos à plataforma ***Apache JMeter***, que, por sua vez, é uma das ferramentas mais utilizadas para medir a performance de um determinado sistema, oferecendo uma interface gráfica para a criação dos testes. Esta ferramenta permite definir várias definições, tais como o número de *threads*, o número de vezes que cada *thread* é executada e o intervalo entre cada execução.

Situamos o número de *threads* em 100, o número de vezes que cada *thread* é executada em 100 e o intervalo em 10s. De forma a testar o desempenho desta arquitetura optamos por fazer os seguintes pedidos:

- ***HomePage***: Pedido para aceder à página inicial do *Wiki.js;*
- ***All pages***: Pedido para receber todas as páginas criadas;
- ***Create Comment***: Pedido para inserir um comentário novo numa página;
- ***Create Groups***: Pedido para inserir um novo grupo de utilizadores;
- ***Create Page***: Pedido para inserir uma nova página na aplicação;
- ***Create User***: Pedido para inserir um novo utilizador na aplicação;
- ***Get Comments***: Pedido para receber todos os comentários de uma determinada página;
- ***Get Groups***: Pedido para receber todos os grupos já inseridos pelo administrador;

- ***Get Users***: Pedido para receber todos os utilizadores já inseridos pelo administrador;
- ***Login (Admin)***: Pedido para realizar login com os dados do administrador;
- ***Login (User)***: Pedido para realizar login com os dados de um utilizador;

Para a apresentação dos resultados decidimos apresentar os gráficos resultantes, corridos através da linha de comandos, antes e depois da implementação da arquitetura do sistema. Decidimos que a granularidade dos gráficos se iria situar nos 2 segundos, de modo a obtermos registos de 2 em 2 segundos.

Teste 1

1. Estatísticas

Na seguinte figura 3, conseguimos observar as estatísticas associadas a um sistema muito simples, sem uma arquitetura distribuída. O tempo de execução situou-se nos 7 minutos, um tempo consideravelmente maior em comparação com o sistema, que tem uma arquitetura implementada. Podemos também perceber, que a média aritmética de todas as respostas neste sistema é muito alta, cerca de 2395 ms, comparativamente com o outro sistema, cerca de 370 ms. Relativamente ao *throughput*, número de pedidos por segundo que o servidor recebeu, observamos que existe uma grande diferença, neste caso temos cerca de 27 transações por segundo, enquanto que no outro sistema, obtemos cerca de 78 transações por segundo.

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	**11000**	**0**	**0.00%**	**2395.43**	**46**	**18992**	**1656.00**	**5643.70**	**7017.00**	**10411.21**	**26.49**	**1611.73**	**23.71**
All Pages	1000	0	0.00%	2654.43	50	10504	2062.00	5818.50	6734.50	9508.91	2.44	2.40	1.00
Create Comment	1000	0	0.00%	913.88	46	10867	615.50	1997.90	2338.15	5533.20	2.43	2.99	2.79
Create Groups	1000	0	0.00%	4917.30	84	13479	4566.00	9776.10	10793.75	13041.58	2.43	1.89	2.50
Create Page	1000	0	0.00%	1873.11	53	7176	1492.50	3829.90	4906.15	5773.55	2.43	1.75	4.30
Create User	1000	0	0.00%	1827.76	51	7231	1502.00	3653.20	4428.80	5632.99	2.43	1.68	3.34
GET Comments	1000	0	0.00%	3388.01	55	11171	3032.50	6562.60	7521.70	10497.91	2.44	22.92	2.24
GET Groups	1000	0	0.00%	3832.26	342	13901	3232.00	7313.90	8508.85	10018.76	2.42	1571.29	2.05
GET Users	1000	0	0.00%	2102.68	49	7273	1705.00	4715.00	5484.85	6287.96	2.43	3.11	2.11
HomePage	1000	0	0.00%	2414.34	93	10412	2008.50	4802.60	5668.15	7505.69	2.44	7.55	0.29
Login (Admin)	1000	0	0.00%	1376.27	50	14926	877.00	2751.90	4054.90	13165.23	2.43	3.08	1.66
Login (User)	1000	0	0.00%	1049.64	46	18992	643.50	2265.00	3029.90	6150.97	2.43	3.08	1.66

Figura 3 Estatísticas sem arquitetura

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	**11000**	**1**	**0.01%**	**370.37**	**46**	**7037**	**263.00**	**704.00**	**951.00**	**2156.97**	**79.92**	**5060.74**	**80.71**
All Pages	1000	0	0.00%	237.22	49	1529	196.00	465.80	581.00	749.98	7.35	11.16	2.98
Create Comment	1000	1	0.10%	301.55	60	1323	251.50	562.50	664.95	988.93	7.40	10.85	9.84
Create Groups	1000	0	0.00%	432.23	98	2962	370.00	771.50	927.90	1268.45	7.40	11.02	8.96
Create Page	1000	0	0.00%	316.43	63	1597	275.00	582.80	686.95	894.83	7.40	10.61	14.45
Create User	1000	0	0.00%	326.33	62	2051	273.50	622.80	729.95	986.77	7.40	10.40	11.54
GET Comments	1000	0	0.00%	375.00	65	2259	308.00	715.40	875.55	1459.77	7.43	16.28	8.17
GET Groups	1000	0	0.00%	1088.16	252	7037	817.00	2202.50	2846.35	4477.16	7.41	5037.14	7.64
GET Users	1000	0	0.00%	316.41	62	2136	273.00	572.90	694.95	934.91	7.42	17.67	7.80
HomePage	1000	0	0.00%	263.36	48	1838	212.50	481.90	639.75	917.67	7.34	23.10	0.83
Login (Admin)	1000	0	0.00%	209.51	46	2627	164.00	380.00	481.00	993.95	7.35	5.94	4.98
Login (User)	1000	0	0.00%	207.83	46	2821	167.00	386.80	476.90	655.89	7.37	5.92	5.00

Figura 4 Estatísticas com arquitetura

Nesta figura, podemos consultar toda a informação relativa ao sistema que tem uma arquitetura bem distribuída e resiliente. Existem grandes diferenças, não só as já referidas no tópico anterior, mas também a nível do tempo de execução dos testes, visto que este apenas demorou cerca de 2 minutos, em comparação com os 7 minutos do outro sistema, sem uma arquitetura.

2. *Over time*

Entrando em mais detalhe, relativamente aos tempos decorridos nos testes, é notoriamente evidente nestes dois gráficos (ver figura 5 e figura 6) a discrepância que existe na escala da média do tempo de resposta. Neste primeiro sistema, temos uma escala que anda entre os 47 ms e os 13057 ms, enquanto que no outro sistema, os valores andam entre os 50 ms e os 2022 ms.

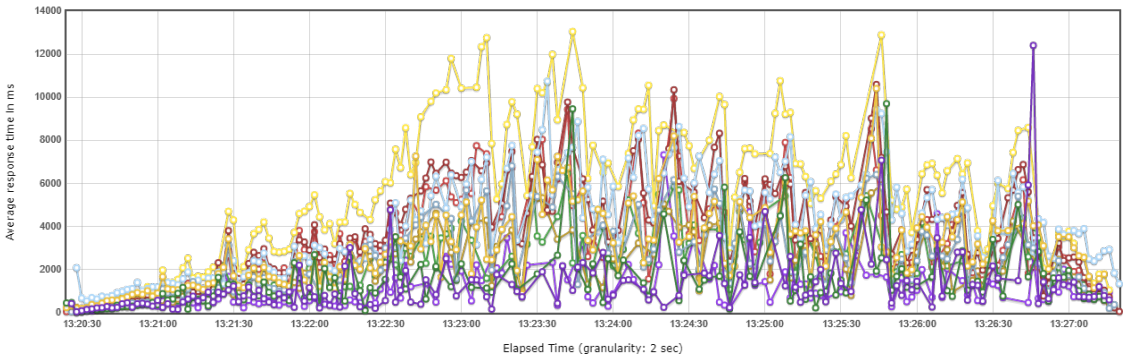

Figura 5 Over time sem arquitetura

Como observado na figura 6, temos tempos de resposta, mais favoráveis, e com muito menos oscilações no decorrer do tempo, o que por sua vez significa que temos um sistema bem distribuído, resiliente e eficaz no que toca a testes de carga.

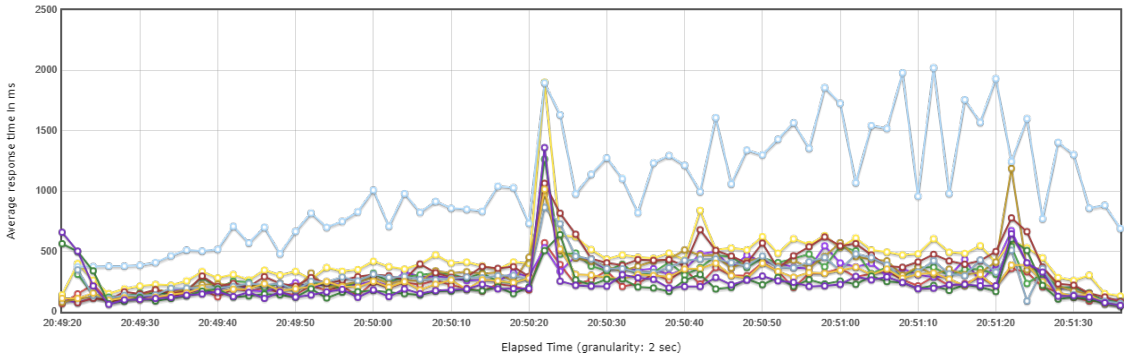

Figura 6 Over Time com arquitetura

Quanto à latência no decorrer dos testes, conseguimos perceber que esta segue a mesma tendência que o ponto acima referido, ou seja, no sistema distribuído (figura 8) temos uma escala que se situa entre os 50 ms e os 1903 ms, enquanto que no sistema sem uma arquitetura, esta situa-se entre os 50 ms e os 13000 ms.

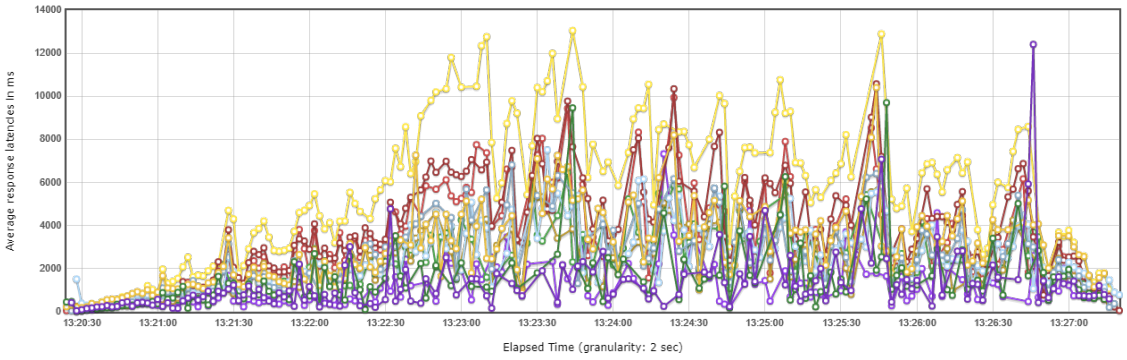

Figura 7 Latência Over time sem arquitetura

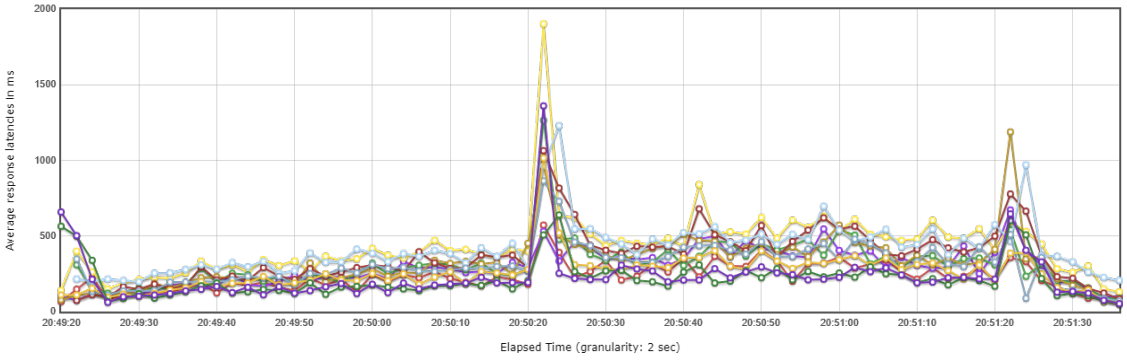

Figura 8 Latência Over time com arquitetura

Os gráficos apresentados na Figura 9 e Figura 10 representam o tempo de resposta em função do número de *threads*. Estes permitem identificar quando a aplicação passa a ter uma performance inferior ao esperado, ou seja, o tempo de resposta para o número de utilizadores começa a aumentar. Como podemos observar, no sistema sem uma arquitetura, temos inicialmente um aumento exponencial, valor o qual vem sendo reduzido consoante o tempo, mas de forma muito reduzida. Quanto ao número de *threads* ativas, conseguimos também observar que, neste tipo de sistema o valor é consideravelmente maior, cerca do dobro em comparação com o sistema com uma arquitetura.

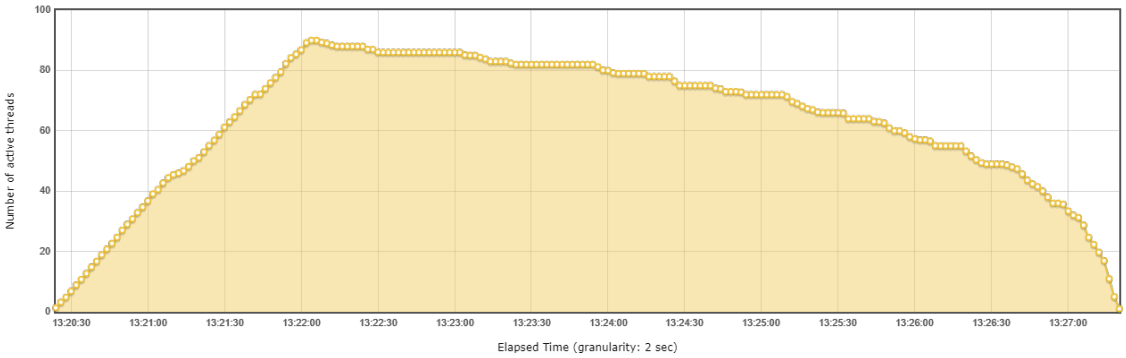

Figura 9 Active threads over time sem arquitetura

Este último sistema (ver figura 10), apresenta um número máximo de *threads* ativas, de cerca de 50, o que representa metade do outro sistema. Olhando agora para a linha do tempo, conseguimos visualizar um comportamento típico de um sistema distribuído, ou seja, ao longo do tempo vamos tendo mais pedidos, e por sua vez, a carga no sistema vai aumentar, o que representa um acumular de *threads* ativas, não obstante que o número de transações, como iremos ver a seguir, mantém se muito constante ao longo deste tempo, o que significa uma forte resiliência deste sistema.

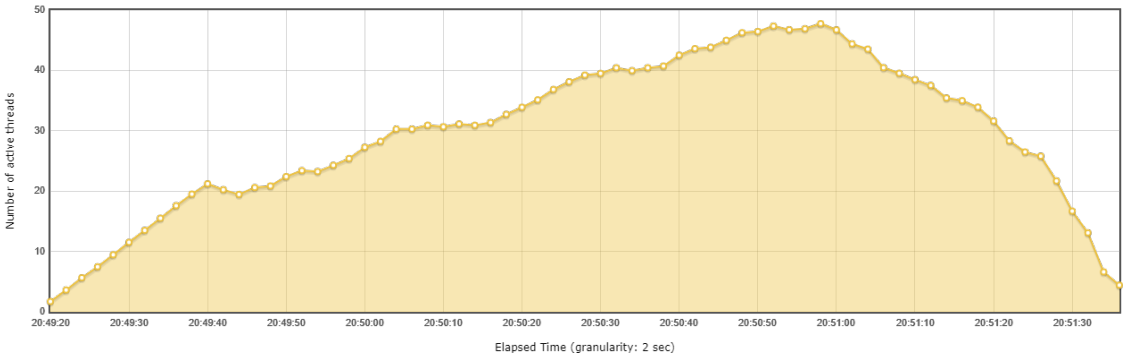

Figura 10 Active threads over time com arquitetura

3. *Throughput*

Analisando agora, com mais detalhe, o número de pedidos por segundo que o servidor recebeu, podemos constatar que, este primeiro sistema, apresenta valores muito inconstantes ao longo do tempo.

Analisando o número de transações por segundo, (figura 11) a escala do número de transações, situa-se entre 1 e 11 transações por segundo no decorrer dos cerca de 7 minutos de teste.

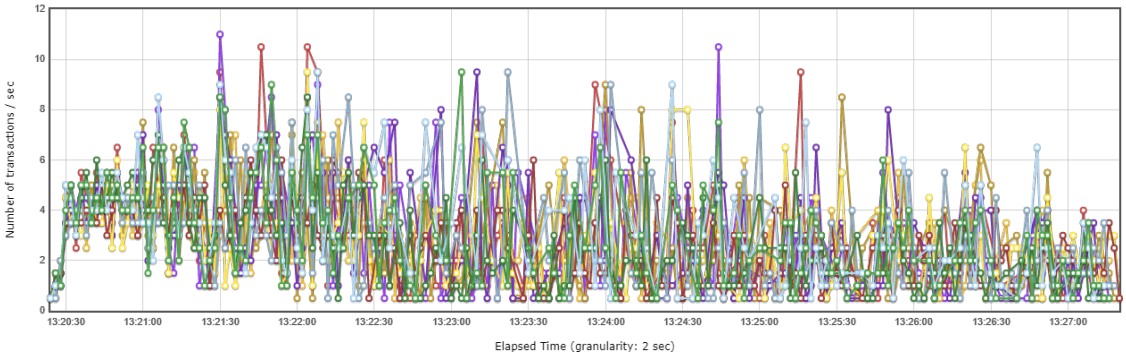

Figura 11 Transações por segundo sem arquitetura

Já neste tipo de sistema, conseguimos observar (ver figura 12) valores mais constantes, ou seja, menos oscilatórios, que se situam entre as 1 e as 13 transações por segundo no decorrer dos cerca de 2 minutos de teste.

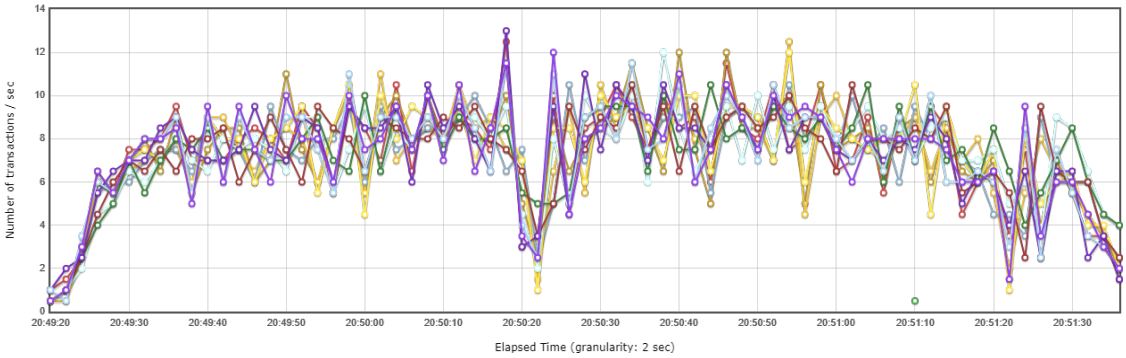

Figura 12 Transações por segundo com arquitetura

Nas figuras abaixo implícitas estão presentes os tempos de resposta em função do número de pedidos por segundo. Podemos verificar que neste primeiro sistema (figura 13), a mediana do tempo de resposta apresenta, não só valores substancialmente maiores, comparativamente com o do sistema da figura 14, mas também em função do número global de pedidos por segundo e por fim valores mais oscilantes, o que representa um sistema mais instável.

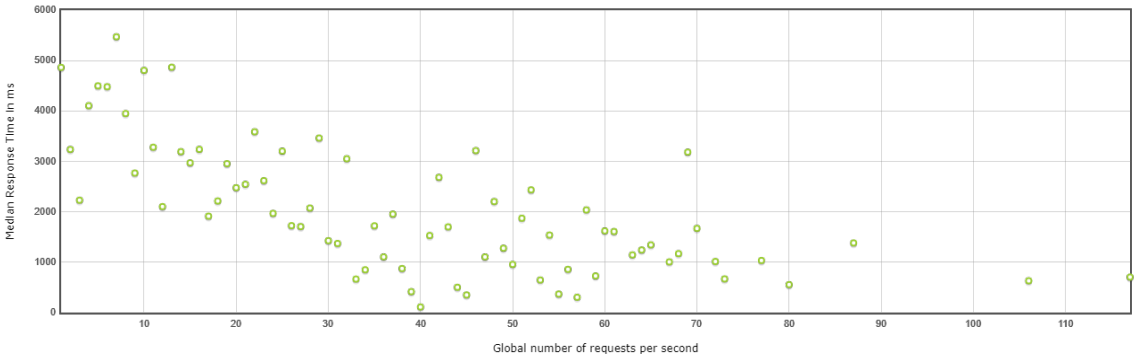

Figura 13 Response time VS Request sem arquitetura

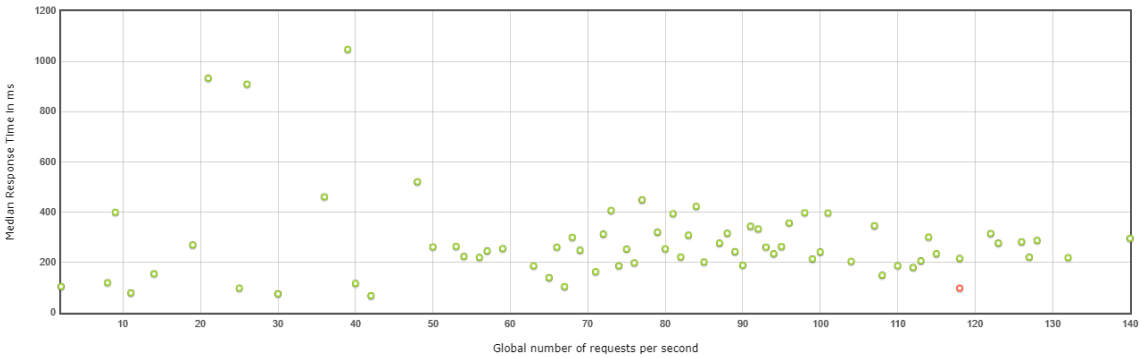

Figura 14 Response time VS Request com arquitetura

4. *Response times*

Por fim, analisando o tempo de resposta nos respetivos percentis, visualizámos que o sistema sem arquitetura apresenta valores máximos de cerca de 20.000 ms, enquanto que o outro sistema apresenta valores máximos de 7.000 ms em função do respectivo percentil. Posto isto, podemos afirmar que o sistema que possui uma arquitetura definida, apresenta valores muito mais representativos de um sistema distribuído, resiliente e eficaz.

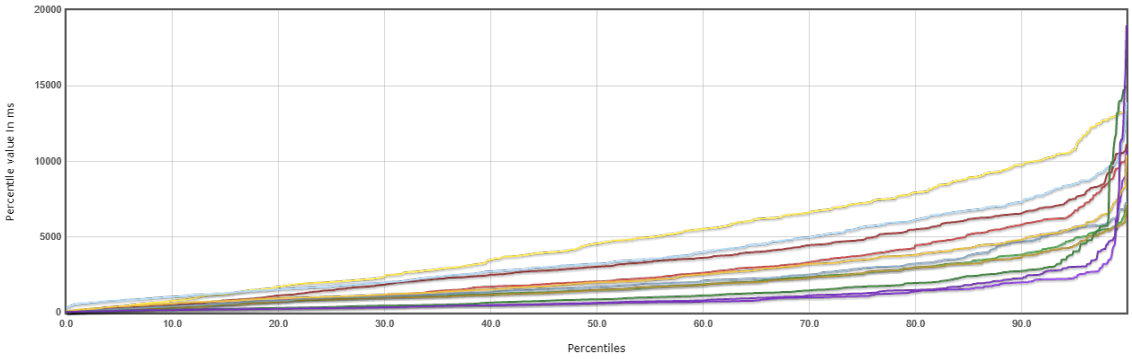

Figura 16 Response times sem arquitetura

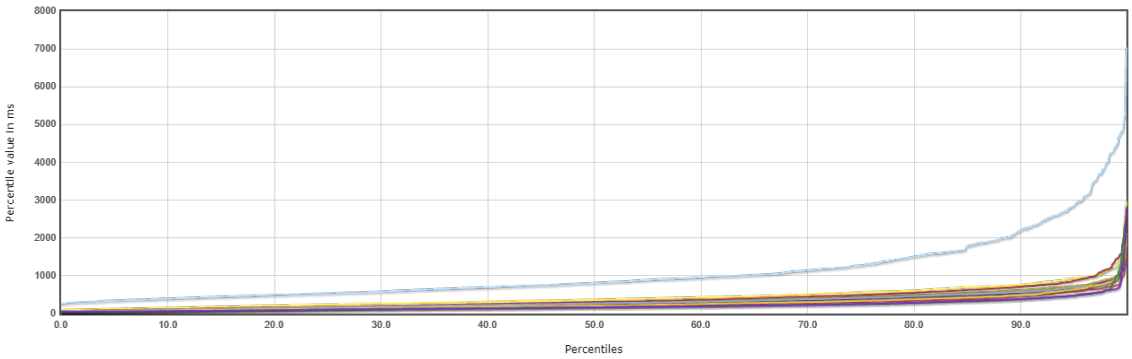

Figura 15 Response times com arquitetura

Teste 2

Posto estes resultados, decidimos também testar a arquitetura implementada, com as 2 instâncias de wiki ligadas, com a mesma arquitetura, mas apenas com 1 instância de wiki ligada.

1. Over time

Olhando então para o tempo de resposta ao longo do decorrer dos testes, podemos afirmar que tendo as duas instâncias de wiki ligadas, conseguimos uma melhor performance, pois no gráfico em baixo temos valores que rondam desde os 0 ms até aos 7000 ms com apenas uma instância de wiki, enquanto que na arquitetura com as duas instâncias tínhamos valores máximos de 2000 ms.

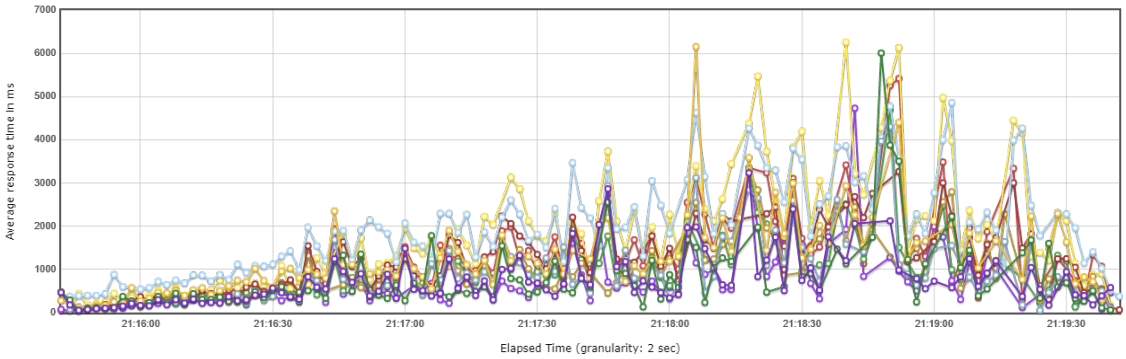

Figura 17 Over time com uma instância

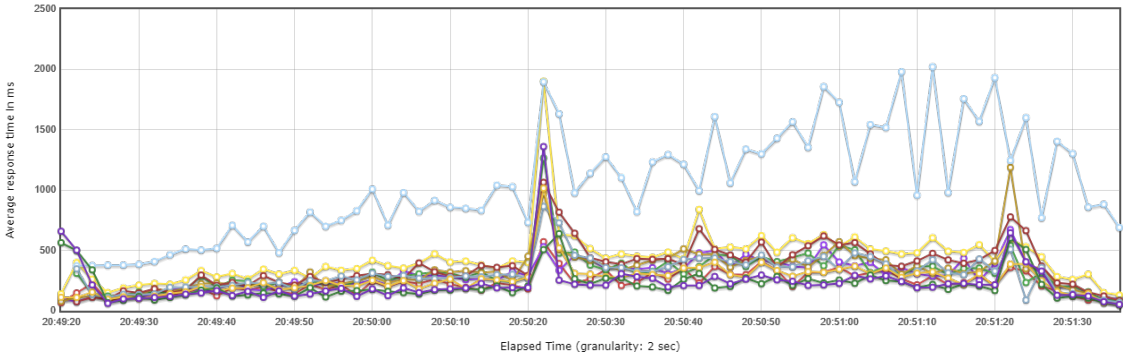

Figura 18 Over time com duas instâncias

2. *Throughput*

Relativamente às transações por segundo no decorrer dos testes, mais uma vez conseguimos perceber que temos mais desempenho com as duas instâncias ligadas, visto que com apenas uma arquitetura temos valores que se situam com mais frequência entre as 2 e as 8 transações por segundo.

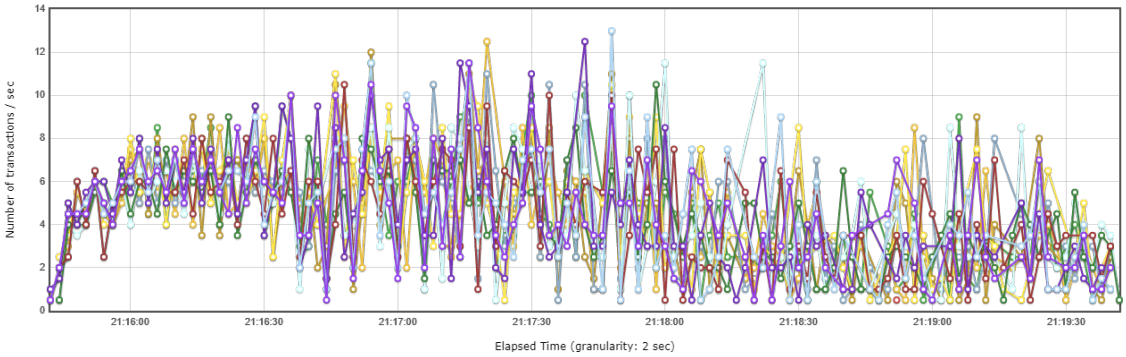

Figura 19 Transações por segundo com uma instância

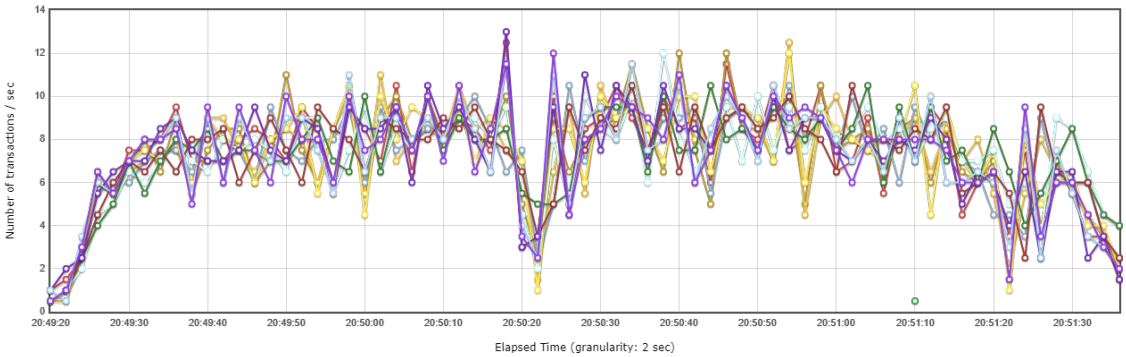

Figura 20 Transações por segundo com duas instâncias

Relativamente ao tempo de resposta em função do pedido, conseguimos perceber que, os valores demonstrados na figura dos testes da arquitetura com apenas 1 instância aparentam ser menos oscilatórios, mas o intervalo da mediana do tempo de resposta é consideravelmente maior.

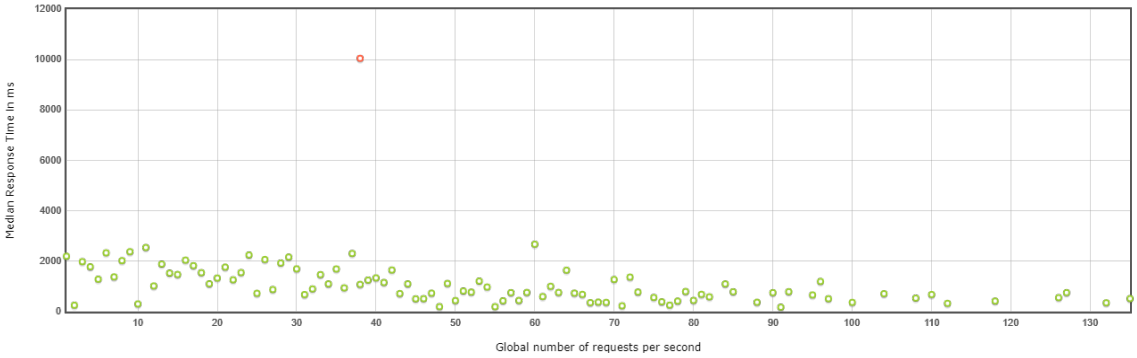

Figura 22 Response time VS Request com uma instância

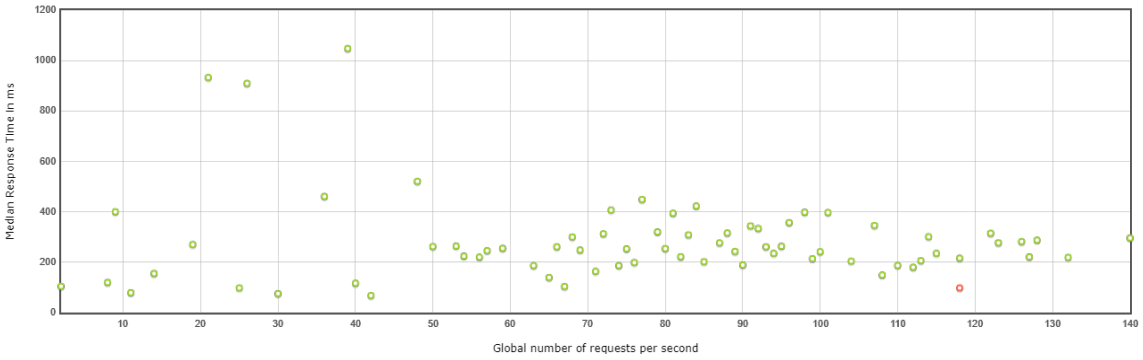

Figura 21 Response time VS Request com duas instâncias

3. *Response times*

Por último, analisando o tempo de resposta, conseguimos perceber, mais uma vez, que sem dúvida alguma termos duas instâncias a funcionar com a camada aplicacional, conseguimos tirar um melhor desempenho e por sua vez uma maior satisfação com o cliente. Como demonstram as figuras abaixo, com apenas uma instância da aplicação, temos valores de tempo de resposta mais altos, o que indica que o utilizador esperou mais tempo em comparação com a outra arquitetura que possuía as duas instâncias de wiki ligadas.

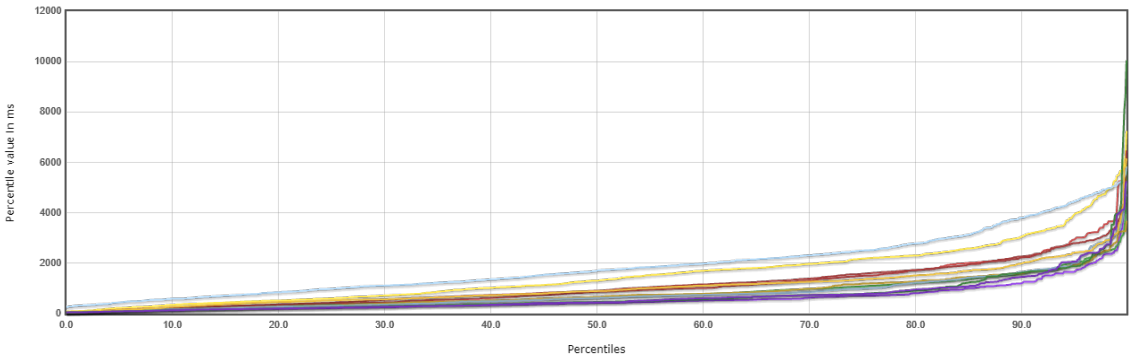

Figura 23 Response time com uma instância

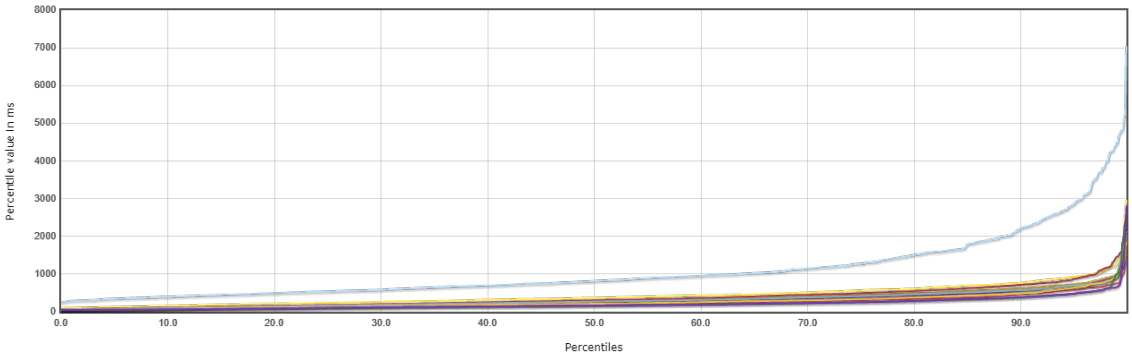

Figura 24 Response time com duas instâncias

Observando os gráficos dos três tipos de arquiteturas, com apenas uma única instância de wiki, uma outra com duas instâncias de wiki e, por último com apenas uma única instância com todas as camadas instaladas, concluímos que sem incerteza alguma, a arquitetura com 2 instâncias wiki é que apresenta melhores resultado a nível de desempenho. Relativamente à arquitetura com apenas uma instância de wiki, conseguimos uma melhoria de desempenho bastante significativa em comparação com a instância que tinha todas as camadas instaladas.

6. Conclusão

O planeamento de uma Infraestrutura é uma parte fundamental para o bom funcionamento de uma aplicação, de modo a que esta consiga responder com sucesso às exigências do utilizador. Com a realização deste Trabalho Prático foi possível aplicar os diversos conhecimentos que nos foram sendo passados ao longo de todo o semestre, nomeadamente na Unidade Curricular de Infraestrutura de Centro de Dados. Deste modo, foi possível consolidar determinadas noções, como por exemplo sobre a alta disponibilidade de infraestruturas e escalabilidade das mesmas, pois após uma análise de performance de diversos componentes, foi possível planear uma topologia que promove alta disponibilidade e escalabilidade.

As maiores dificuldades encontradas foram ao nível de estabelecer a comunicação entre certas camadas da nossa arquitetura, dado que nem todos os recursos com os quais tínhamos trabalhado estão disponíveis na *Cloud*. Nomeadamente o *Load Balancer* para a camada aplicacional, visto que não conseguimos utilizar o que foi lecionado pelo docente, tivemos que utilizar o disponibilizado pelo Google.

A arquitetura implementada apresenta um maior desempenho em comparação com a arquitetura que não tinha qualquer tipo de distribuição e, com a arquitetura que apenas tinha uma única instância de wiki. Com esta arquitetura implementada, permite-nos assim uma maior rapidez nas respostas dos pedidos e uma maior redução da percentagem de erro, fazendo com que mais utilizadores naveguem na aplicação em simultâneo e com uma taxa de sucesso mais alta. No que diz respeito ao débito e à latência dos pedidos dos utilizadores, os valores são muito mais favoráveis em comparação com as outras arquiteturas e por isso concluímos que com uma arquitetura bem distribuída e resiliente conseguimos obter uma enorme melhoria no desempenho geral de toda a infraestrutura.

Para concluir, como trabalho futuro de modo a melhorar a escalabilidade da nossa plataforma seria implementar réplicas da base de dados de modo a termos mais do que uma instância deste serviço disponível eliminando assim este possível *bottleneck*.