



Universidade do Minho
Escola de Engenharia

Aplicações e Serviços de Computação em Nuvem
Grupo 23
2023/2024

Tomás Cardoso Francisco pg54263
Afonso Xavier Cardoso Marques pg53601
Maria Eugénia Bessa Cunha pg54042
Ana Filipa da Cunha Rebelo pg53624

Dezembro 2023



Índice

1	Introdução	4
2	Laravel.io	4
3	Arquitetura	5
4	Automatização	6
4.1	Criação da imagem <i>Docker</i>	6
4.2	Criação do <i>Cluster Kubernetes</i>	6
4.3	Criação dos nodos	6
4.4	Criação dos pods, serviços e monitorização	7
5	Métricas de avaliação	8
6	Avaliação e Testes	11
6.1	Considerando a instalação base proposta pelo grupo para a Tarefa 1:	11
6.2	Com base nas respostas dadas para as questões anteriores:	12
7	Conclusão	13

List of Figures

1	Logótipo da Laravel	4
2	Arquitetura proposta	5
3	Cluster	6
4	Nodos	7
5	Carga de Trabalho (<i>Pods</i>)	7
6	Serviços	7

1 Introdução

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

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

A primeira etapa do trabalho consistiu no *deployment* da aplicação com recurso ao *Ansible* e aos serviços fornecidos pelo *Google Cloud Platform*, efetuando a instalação no menor número de passos possível. A segunda fase do trabalho consistiu no *deployment* da aplicação recorrendo aos *kubernetes* e aos serviços suportados pelo *Google Cloud Platform* com o intuito de otimizar o desempenho, escalabilidade e resiliência da aplicação. Por fim, a terceira e última etapa do trabalho envolve a seleção, instalação e testes automatizados das ferramentas de monitorização e avaliação. Isso inclui escolher as melhores ferramentas, instalá-las e realizar testes automáticos que simulem a interação dos utilizadores.

2 Laravel.io

O **Laravel.io** é uma comunidade *online* essencial para desenvolvedores que trabalham com o *framework PHP*, *Laravel*. Funciona como um fórum ativo, onde os utilizadores compartilham conhecimento, encontram soluções para problemas, acedem a recursos educativos e se mantêm atualizados sobre eventos e notícias relevantes para o *Laravel*. Além disso, oferece oportunidades de colaboração em projetos, *networking* e é um espaço aberto para aprendizagem contínua. É uma plataforma vital para a comunidade de desenvolvedores *Laravel*.



Figure 1: Logótipo da Laravel

3 Arquitetura

A arquitetura do sistema adota um paradigma multicamada, caracterizado pela estruturação em diferentes níveis funcionais. Essa abordagem compreende diferentes camadas, das quais se destacam a interface destinada ao utilizador (Laravel.IO) e uma camada dedicada à persistência dos dados (MySQL). Tal configuração proporciona uma clara separação de responsabilidades e funcionalidades entre os elementos do sistema, permitindo a interação eficiente entre a interface de utilizador e os processos de armazenamento e manipulação de dados.

Esta estrutura *multilayer* não apenas facilita a manutenção e evolução do sistema, mas também promove uma maior flexibilidade e escalabilidade, garantindo uma experiência robusta e adaptável aos requisitos e demandas do ambiente operacional.

Dito isto, para atender aos requisitos estabelecidos, além dos componentes já mencionados, é necessário introduzir os seguintes elementos:

- JMETER - é uma ferramenta de auxílio na realização de testes de carga a sistemas computacionais.
- Google Monitoring - é uma plataforma de monitorização oferecida pelo Google Cloud Platform (GCP). Esta ferramenta oferece uma visão abrangente do desempenho e do estado de várias camadas de uma aplicação hospedada no GCP.

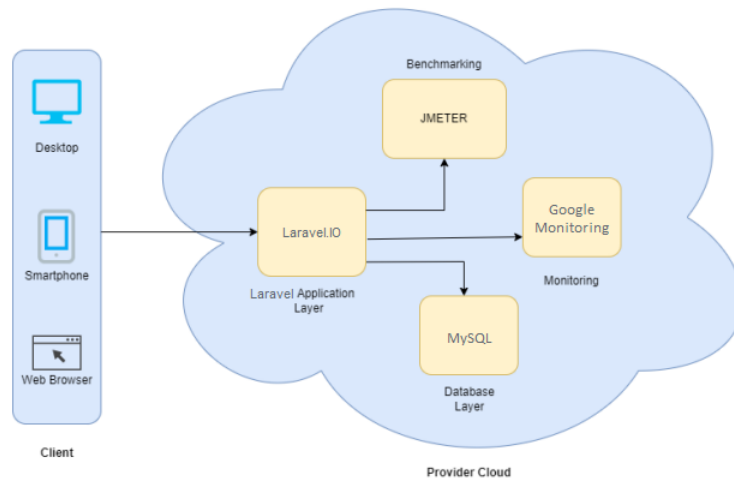


Figure 2: Arquitetura proposta

4 Automatização

A automatização do processo de *deployment* da aplicação **Laravel.io** levou à utilização das ferramentas *GKE* (*Google Kubernetes Engine*) e *GCP* (*Google Cloud Platform*), através de *playbooks* escritos em YAML para executar em Ansible. Nas seguintes secções encontra-se a explicação de todo o processo de instalação e *deployment* das várias componentes utilizadas no projeto.

4.1 Criação da imagem *Docker*

A primeira etapa do trabalho consistiu na criação da imagem *Docker* para a aplicação Laravel.io. Para tal o grupo criou um Dockerfile que descreve passo a passo as configurações e instalações necessárias para criar um ambiente funcional para a aplicação começando com a definição da imagem base como Ubuntu 22.04 e procedendo com as seguintes etapas:

- Atualização e Instalação de Dependências: As instruções RUN foram utilizadas para atualizar o sistema, instalar o PHP 8.2, Node.js, Composer, Git, MySQL Client e outras dependências essenciais.
- Configuração do ambiente Laravel: O Dockerfile copia o repositório da aplicação Laravel.io, configura o ficheiro de ambiente .env e gera uma chave única para a aplicação Laravel usando o comando `"php artisan key:generate"`.
- Gestão de Dependências da Aplicação: As dependências do Laravel foram geridas usando o Composer e o Node Package Manager (npm). O Dockerfile executa os comandos `"composer install"` para as dependências PHP e `"npm install && npm run build"` para as dependências JavaScript da aplicação.
- Configuração Final e Exposição de Porta: Um script de inicialização foi copiado para dentro da imagem e configurado como o comando padrão a ser executado quando a imagem é iniciada. Além disso, a porta 80 foi exposta para permitir o acesso à aplicação.

4.2 Criação do *Cluster Kubernetes*

A segunda fase do trabalho envolveu a criação de um *cluster Kubernetes* com um número arbitrário de nodos. Ao criar uma instância de *cluster* no *Google Cloud*, ocorre automaticamente a geração de containers que sustentam os nodos vinculados ao referido *cluster*.

Para realizar essa etapa, foi utilizado um ficheiro *playbook* (*gke-cluster-create.yml*) que emprega o módulo *GCP Container Cluster* para efetuar a criação do *cluster*, e o módulo *GCP Container Node Pool* para realizar a criação dos nodos associados a esse *cluster*.

<input type="checkbox"/> Status	Name ↑	Location	Fleet ?	Number of nodes	Total vCPUs	Total memory
<input checked="" type="checkbox"/>	ascn-cluster	us-central1-a	REGISTER	2	4	4 GB

Figure 3: Cluster

4.3 Criação dos nodos

Para executar com sucesso a criação dos nodos na plataforma *Google Cloud*, foi necessário levar em consideração a chave do projeto da *ASCN*, obtida na *Google Console*, a fim de fornecer acesso ao projeto.

Todo o processo de criação dos nodos foi realizado de forma remota, utilizando um *host* externo à *Google Cloud*. Esse método permitiu a execução eficiente das tarefas planeadas, facilitando a gestão e configuração das instâncias virtuais de maneira flexível e adaptada às necessidades específicas do projeto.

Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	gke-ascn-cluster-default-pool-3e7d1994-02r9	us-central1-a		gke-ascn-cluster-default-pool-	10.128.15.216 (nic0)	34.135.91.140 (nic0)	SSH ▾
✓	gke-ascn-cluster-default-pool-3e7d1994-p0zv	us-central1-a		gke-ascn-cluster-default-pool-	10.128.15.215 (nic0)	34.41.175.42 (nic0)	SSH ▾

Figure 4: Nodos

4.4 Criação dos pods, serviços e monitorização

Finalmente, é possível prosseguir para o *deployment* da aplicação, na qual são criados dois pods e dois serviços: um para a aplicação Laravel e outro para a sua base de dados. Isto permite descentralizar a carga da aplicação, distribuindo-a por vários containers, obtendo assim um melhor balanceamento desta.

O processo de criação destes serviços é feito de forma automatizada, com recurso ao playbook *laravelio-deploy.yml* que



Name ↑	Status	Type	Pods	Fleet ?	Namespace	Cluster
laravel-deployment	✓ OK	Deployment	1/1	 ascn-23-24	laravel	ascn-cluster
mysql-deployment	✓ OK	Deployment	1/1	 ascn-23-24	db	ascn-cluster

Figure 5: Carga de Trabalho (*Pods*)


Name ↑	Status	Type	Endpoints	Pods	Namespace	Clusters
laravel-service	✓ OK	External load balancer	34.173.218.185:80 	1/1	laravel	ascn-cluster
mysql-service	✓ OK	Cluster IP	10.0.9.37	1/1	db	ascn-cluster

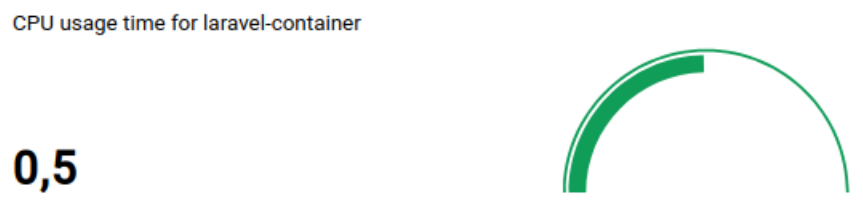
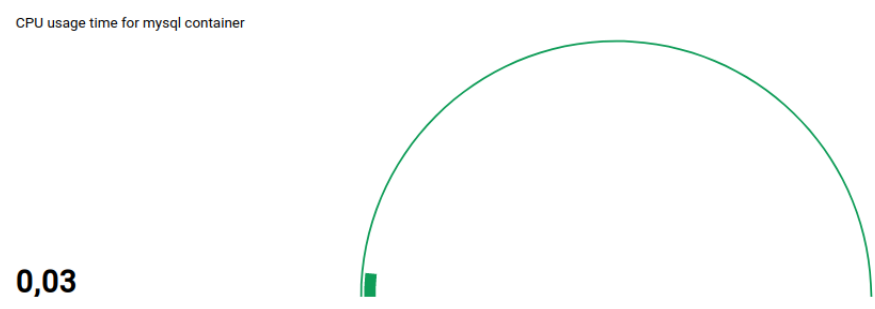
Figure 6: Serviços

Por último criou-se um *playbook* (*monitoring.yml*) que lança uma *dashboard* para monitorizar o cluster do projeto. Este *playbook* recorre a comandos do módulo *kubectl* e *gcloud monitoring*.

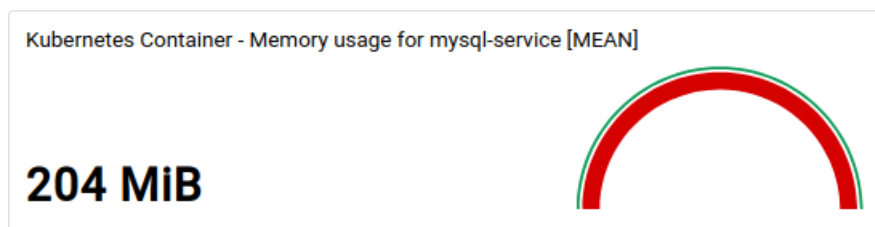
5 Métricas de avaliação

Tendo em conta o problema e as métricas disponibilizadas pelo Google Monitoring, resolvemos utilizar as seguintes métricas, sendo que as que se considerou mais pertinentes no problema atual:

- Tempo de utilização do CPU nos containers do MySQL e Laravel: Esta métrica fornece insights sobre a carga de trabalho no processamento das consultas da base de dados (MySQL) e da aplicação (Laravel). O alto tempo de utilização da CPU pode indicar sobrecarga ou demanda excessiva, influenciando diretamente o desempenho



- Utilização de memória pelos containers MySQL e Laravel: A monitorização da utilização de memória é crucial para garantir que os recursos estejam a ser utilizados de maneira eficiente. O alto uso de memória pode levar a gargalos de desempenho e até mesmo a falhas se não houver memória suficiente para operações importantes.



Kubernetes Container - Memory usage for laravel-service [MEAN]

83 MiB



- Utilização do CPU para os nodos do cluster: Observar a utilização de CPU nos nodos do cluster oferece uma visão geral do uso de recursos no ambiente. Isso é essencial para garantir que a capacidade de processamento esteja adequada e para identificar qualquer nodo que esteja sobrecarregado.

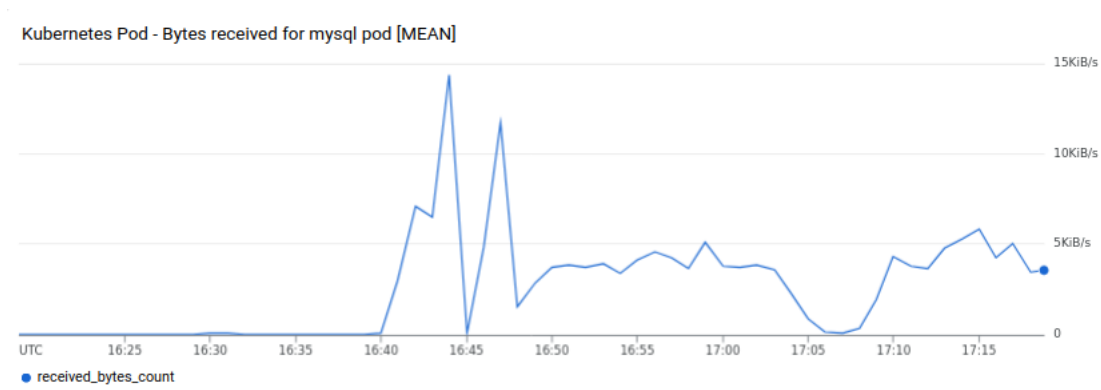
Kubernetes Node - CPU allocatable utilization for both nodes [MEAN]



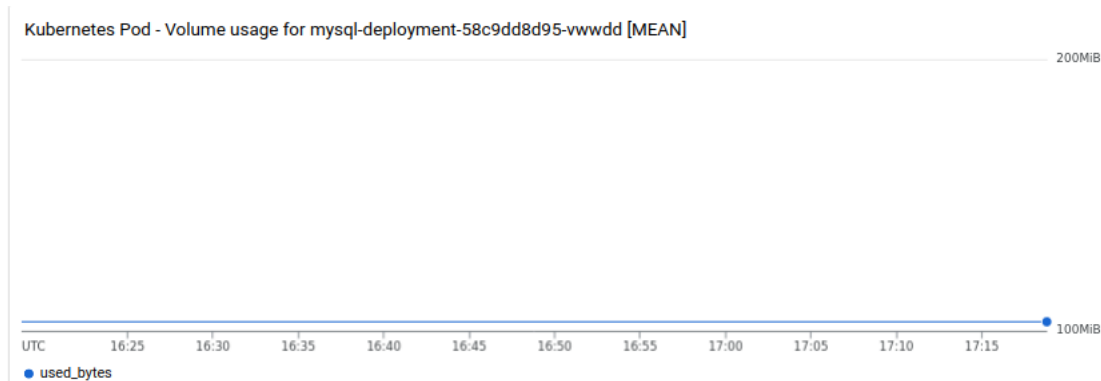
- Bytes transmitidos e recebidos do MySQL Pod: Monitorizar o volume de dados transmitidos e recebidos pelo pod do MySQL é essencial para entender o tráfego de rede. Grandes volumes podem indicar uma alta atividade de comunicação, o que pode afetar o desempenho e a latência da base de dados.

Kubernetes Pod - Bytes transmitted for mysql pod [MEAN]





- Volume usado do MySQL Pod: Observar o volume de armazenamento utilizado pelo pod do MySQL é crucial para garantir que haja espaço suficiente para armazenar dados. Isso pode prevenir falhas devido à falta de espaço em disco.



Estas métricas foram escolhidas porque oferecem uma visão abrangente do desempenho, uso de recursos e saúde dos componentes críticos da infraestrutura (como a base de dados e aplicação). Através delas, é possível identificar gargalos, antecipar problemas e otimizar a infraestrutura para garantir um funcionamento mais estável e eficiente da aplicação.

6 Avaliação e Testes

Com vista a poder responder às questões do enunciado e para poder efetuar testes de desempenho da implementação desenvolvida, foi necessário criar testes que permitissem avaliar os diferentes componentes e funcionalidade da aplicação, tendo em conta a possível variação de utilizadores que possam estar a utilizar. Para tal, recorreu-se à ferramenta JMeter, que permite efetuar testes de carga/stress através de um ficheiro de input, assim como simular um grande fluxo de utilizadores na aplicação.

Posteriormente, já com os ficheiros de teste obtidos, abriu-se os ficheiros no JMeter e iniciou-se os testes, sendo atribuídos diferentes valores de threads na aba "Thread Group", permitindo assim submeter a plataforma a uma maior ou menor carga. É de reforçar que a avaliação experimental da aplicação foi combinada com as métricas de monitorização com o objetivo de extrair observações mais completas e incisivas sobre os resultados observados. Infelizmente não conseguimos obter resultados concretos o que fez com que a discussão das questões do enunciado ficasse mais carente.

6.1 Considerando a instalação base proposta pelo grupo para a Tarefa 1:

1. Para um número crescente de clientes, que componentes da aplicação poderão constituir um gargalo de desempenho?

R: Em ambientes distribuídos como o proposto, alguns componentes da aplicação Laravel.io podem-se tornar gargalos de desempenho à medida que o número de clientes aumenta.

Um dos pontos críticos é a base de dados. À medida que mais clientes interagem com a aplicação, a carga sobre a base de dados cresce consideravelmente. Isto pode resultar em lentidão nas consultas e operações. É essencial realizar monitorização e otimização constantes das consultas, índices e esquemas para lidar com esse aumento no serviço.

Além disso, o balanceamento de carga é outro aspecto crucial. Se não estiver configurado adequadamente, pode tornar-se um gargalo ao direcionar excesso de tráfego para determinados pods ou componentes da aplicação.

2. Qual o desempenho da aplicação perante diferentes números de clientes e cargas de trabalho?

R: O desempenho da aplicação varia consideravelmente conforme o número de clientes e a carga de trabalho aplicada. Com poucos clientes e uma carga leve, a aplicação mantém um desempenho estável e responsivo. À medida que o número de clientes aumenta gradualmente, assim como a carga de trabalho, podem surgir sinais de desaceleração, embora ainda geríveis.

Entretanto, em picos de carga ou com um grande influxo de clientes, a aplicação pode sofrer problemas mais significativos de desempenho. Isto inclui tempos de resposta mais lentos, alguns períodos de inatividade e erros devido à sobrecarga dos recursos, especialmente tendo em conta que a aplicação não está preparada para lidar com esse volume de tráfego.

3. Que componentes da aplicação poderão constituir um ponto único de falha?

R: De momento, a nossa aplicação dispõe apenas de um único *Load Balancer*. Trata-se de um componente crucial cuja função principal é distribuir o tráfego de entrada entre vários servidores, garantindo que nenhum servidor individual fique sobrecarregado, melhorando assim a confiabilidade e a disponibilidade do sistema. Assim, se este falha, todo o tráfego pode ser interrompido ou dirigido incorretamente, interrompendo serviços. Similarmente, neste momento, apenas temos uma única base de dados representando um ponto único de falha em que, caso esta falhe, todos os serviços ficam suspensos. O *cluster* também poderá falhar caso os seus *pods* falhem, quer sejam ambos ou apenas o pod do *MySQL* devido à dependência que a aplicação *Laravel* apresenta deste serviço.

6.2 Com base nas respostas dadas para as questões anteriores:

1. Que otimizações de distribuição/replicação de carga podem ser aplicadas à instalação base?

R: Para otimizar a distribuição/replicação da carga, podemos efetuar uma distribuição de carga na base de dados, onde, através do particionamento dos dados, equilibramos a carga de trabalho pelos nós da base de dados, ou através da replicação das leituras efetuadas à base de dados, aliviando o peso da consulta à mesma. Também pode ser implementado escalabilidade horizontal entre os pods, como fazemos com o uso de um *load balancer* de forma a evitar que um pod fique sobrecarregado, ou com escalabilidade automática, onde configuramos os pods para que se adaptem a picos de tráfego, dando mais carga a outros pods quando necessário e possível. Alternativamente, o uso de microsserviços também pode ajudar neste problema, tratando cada pod como a sua própria componente independente, tendo sempre em atenção que será preciso uma forma de resumir o progresso do sistema na eventual falha de uma destas componentes, normalmente, tendo outros pods a assumir a função que deu em erro.

2. Qual o impacto das otimizações propostas no desempenho e/ou resiliência da aplicação?

R: As melhorias propostas na questão anterior visam elevar o desempenho e a robustez da nossa aplicação na instalação base sugerida, oferecendo várias otimizações de distribuição e replicação de carga. Para a base de dados, estratégias como replicação de leitura e particionamento dos dados têm o propósito de distribuir consultas e aliviar a carga da base principal. Isso pode melhorar significativamente o desempenho ao reduzir os tempos de resposta, especialmente em momentos de tráfego intenso. Além disso, a escalabilidade horizontal dos pods, através do *autoscaling* e balanceamento de carga, permite a expansão automática da capacidade para lidar com picos de tráfego, distribuindo eficientemente a carga entre os pods disponíveis.

A introdução de microsserviços independentes e mecanismos de failover aumenta a resiliência da aplicação. A redundância oferecida por réplicas de base de dados, pods escaláveis e serviços independentes reduz a vulnerabilidade a falhas individuais, garantindo maior estabilidade da aplicação em caso de problemas. No entanto, é importante estar ciente de que essas otimizações podem acarretar desafios adicionais. A complexidade na configuração e gestão da infraestrutura pode aumentar, e custos extras podem surgir devido à necessidade de infraestrutura adicional para suportar essa distribuição e redundância. O equilíbrio entre os benefícios de desempenho e resiliência e os desafios de complexidade e custos adicionais é essencial. É crucial garantir que a aplicação continue sendo escalável, confiável e economicamente viável ao implementar essas otimizações. Um planeamento cuidadoso é necessário para garantir que os benefícios superem os desafios adicionais introduzidos por essas melhorias.

7 Conclusão

Após a conclusão do trabalho prático, apresentamos uma análise crítica, ponderada e refletida sobre o projeto.

É relevante salientar que a implementação do Laravel está funcional e operacional, destacando-se o *deployment* eficiente através das funcionalidades providenciadas pelo *Google Cloud*, nomeadamente o *Google Kubernetes Engine (GKE)*. Sublinhamos também a utilização de recursos como o balanceamento de carga através de um *LoadBalancer*, que contribui significativamente para a estabilidade e desempenho do sistema.

No que concerne a melhorias e atualizações no projeto, consideramos benéfico implementar uma hierarquia na base de dados. Propomos a criação de uma base de dados primária responsável por suportar o programa, com bases de dados secundárias a entrar em funcionamento em caso de falha da base primária. Esta abordagem proporcionaria maior robustez e resiliência ao sistema, assegurando uma resposta contínua face a eventuais falhas.

O aspeto que consideramos mais fraco no projeto centra-se nos testes de carga que, originalmente, não foram muito conclusivos. Infelizmente, não conseguimos chegar ao cerne da questão, o que dificultou o processo de discussão e resposta às questões do enunciado.

Resumindo, acreditamos que o trabalho tem um balanço positivo. As dificuldades encontradas foram superadas com sucesso, e todos os requisitos propostos foram integralmente cumpridos.