Aplicações e Serviços de Computação em Nuvem
Trabalho Prático

Universidade do Minho
2023/2024

Grupo 16
Inês Daniela Alves Ferreira pg53870
Inês Nogueira Ferreira pg53879
Jéssica Fernandes a93318
Marta da Silva e Sá pg54084
Rafael Ferreira pg53985

29 de dezembro de 2023

Conteúdo

1 Introdução

O presente documento serve o propósito de documentar o trabalho prático desenvolvido no âmbito da Unidade Curricular de Aplicações e Serviços de Computação em Nuvem. O projeto tem como objetivo aplicar os conhecimentos adquiridos ao longo do semestre na automatização de instalação, configuração, monitorização e avaliação de uma aplicação. Para este trabalho em específico, a aplicação em questão é o **Laravel.io**.

A primeira tarefa foca-se na utilização da ferramenta *Ansible* para automatizar a instalação e configuração da aplicação Laravel.io no Google Kubernetes Engine (GKE) da Google Cloud.
A segunda visa explorar e otimizar o desempenho, escalabilidade e resiliência da aplicação, respondendo a questões cruciais relacionadas com o seu funcionamento em condições diversas.

Ao longo do relatório, serão detalhadas escolhas arquiteturais, ferramentas utilizadas e soluções implementadas para atingir os objetivos propostos. Adicionalmente, serão apresentadas reflexões críticas sobre as decisões tomadas e desafios enfrentados ao longo do desenvolvimento do trabalho.
Com este relatório pretendemos não só cumprir os requisitos do projeto mas também demonstrar compreensão dos conceitos lecionados.

2 Aplicação Laravel.io

O Laravel.io é uma aplicação web open source construída com base na framework PHP Laravel. A sua arquitetura é estruturada para proporcionar uma plataforma onde os utilizadores podem colaborar no desenvolvimento de aplicações Laravel.

2.1 Arquitetura

A arquitetura da aplicação segue o modelo Cliente-Servidor, onde o servidor é representado pelo cluster do Kubernetes. Dentro desse cluster, a aplicação Laravel.io e a base de dados estão separadas. Essa abordagem proporciona uma maior capacidade de escalabilidade, uma vez que cada componente pode ser escalado independentemente conforme suas necessidades específicas. Além disso, essa separação oferece uma flexibilidade adicional, permitindo o desenvolvimento e a alteração de funcionalidades de forma independente, pois trabalhamos com *pods* individuais e não com um todo, isto é uma aplicação.

Os elementos centrais são a aplicação *Laravel.io* e a sua base de dados *MySQL*, os quais são implantados e configurados automaticamente pela ferramenta *Ansible* no *Google Kubernetes Engine (GKE)*.

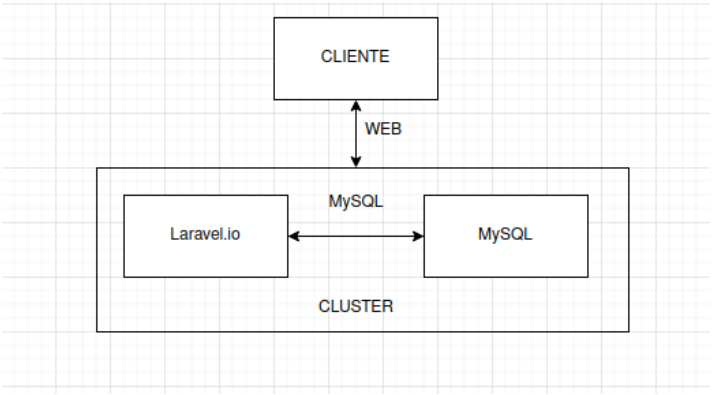

Figura 1: Arquitetura do projeto.

2.2 Componentes

O Laravel.io é composto por vários componentes que se relacionam desempenhando um papel específico na experiência global do utilizador.

- Fórum: o núcleo da app, onde os utilizadores criam tópicos e participam em discussões relacio-

nadas com o desenvolvimento em Laravel;

- Sistema de Respostas: permite responder a perguntas e partilhar conhecimento que contribui para a comunidade;
- Gestão de Utilizadores: inclui funcionalidades como perfis de utilizador, autenticação e autorização para garantir segurança e personalização da experiência;
- Integração com Github: facilita a partilha de código entre utilizadores;
- Pesquisa: para que os utilizadores possam encontrar rapidamente informações relevantes.

3 Instalação e Configuração

3.1 Ferramentas

Utilizamos um *Dockerfile* que foi usado para a construção da imagem *Docker* da aplicação *Laravel.io*. As dependências necessárias são instaladas e o repositório do *Laravel.io* é clonado.

Para automatizar a instalação dos componentes da aplicação, utilizamos a ferramenta *Ansible*. Esta ferramenta proporciona uma abordagem eficiente e simplificada para automatizar processos como o *deployment* e a escalabilidade de aplicações.

3.2 Abordagem

De forma a permitir o balanceamento de carga entre as várias replicações dos serviços críticos do sistema, recorremos ao serviço **Load Balancer** fornecido pela GKE. Este serviço é aplicado durante o deployment do Laravel.io e é responsável por reencaminhar os pedidos para o pod com menos carga.

3.2.1 Criação do *Cluster Kubernetes*

Começamos por criar um *cluster kubernetes* com dois nodos. Sendo que cada um destes é iniciado com uma imagem *Ubuntu*.

Status	Nome	Local	Número de nós	Total de vCPUs	Memória total
✓	ascn-cluster	us-central1-a	2	4	4 GB

Figura 2: Cluster.

Nós

Filtro Filtrar nós

Nome	Status	CPU solicitada	CPU alocável	Memória solicitada	Memória alocável
gke-ascn-cluster-default-pool-cc1de970-1lr5	Ready	287 mCPU	940 mCPU	576.02 MB	1.41 GB
gke-ascn-cluster-default-pool-cc1de970-h6f5	Ready	907 mCPU	940 mCPU	898.92 MB	1.41 GB

Figura 3: Nodos.

3.2.2 Instalação do *Laravel.io* e do *MySQL*

Foi criado um *Deployment* chamado *mysql-deployment*, especificando a imagem do *MySQL* a ser usada e configurando as variáveis de ambiente relacionadas à base de dados.

Para a instalação e *deployment* do *Laravel.io*, criamos um serviço do tipo *LoadBalancer* para o *Laravel.io*, permitindo a exposição externa através da porta 80. Utilizamos a imagem Docker gerada a partir do Dockerfile para criar e executar instâncias da aplicação no ambiente Kubernetes.

No *deployment* do *laravel.io* são instanciados dois pods e criados dois serviços distintos: um para a aplicação *Laravel.io* e outro para sua base de dados. Essa abordagem descentralizada da carga da aplicação permite a distribuição entre vários *containers*, resultando em um balanceamento mais eficiente e otimizado.

Figura 4: Cargas de trabalho (*pods*)

4 Exploração e Otimizações

Para o desenvolvimento deste projeto é necessário fazer a exploração de resultados, com o objetivo de executar as otimizações mais convenientes e atingir a solução ótima. Nomeadamente, devemos ter em conta:

- a existência de pontos de falha, ou seja, componentes que põem em causa a viabilidade do sistema;
- os gargalos de desempenho (*Bottlenecks*), isto é, partes do sistema que limitam capacidade de processamento ou comunicação;
- a variação do número de clientes e da carga de trabalho.

A existência de todos estes aspetos resulta num pior desempenho do sistema.

4.1 Pontos Únicos de Falha

Tendo em consideração os componentes do sistema, é importante avaliar os possíveis pontos de falha, de forma a torná-lo mais resiliente. Desta forma, identificamos os seguintes pontos de falha:

- **Servidor Aplicacional:** caso exista apenas um servidor ativo e ele falhe, o sistema perderá a capacidade de responder a pedidos.
- **Base de Dados:** assim como no servidor aplicacional, depender apenas de uma instância da base de dados pode, em caso de falha, pôr em causa o correto funcionamento do sistema.

Assim, de forma a mitigar este problema, podemos optar pela criação de réplicas do servidor e de um serviço de balanceamento de carga no primeiro caso e pela replicação da base de dados por servidores diferentes no segundo caso.

4.2 *Bottlenecks*

Relativamente à existência de gargalos e analisando mais uma vez os componentes do sistema podemos verificar que, com o aumento do número de utilizadores, tanto o Servidor Aplicacional como a Base de Dados podem tornar-se gargalos de desempenho no sistema.

De forma a reduzir a complexidade desta tarefa optamos por implementar mecanismo que melhorem a resiliência e escalabilidade do Servidor Aplicacional.

4.3 Escalonamento

Esta abordagem foi aplicada através do escalonamento automático de *pod* horizontal, onde há um escalonador automático horizontal de *pods* para cada fluxo de trabalho. Cada escalonador verifica periodicamente as métricas de uma determinada carga de trabalho em relação aos limites configurados e altera a forma da carga de trabalho automaticamente. Neste caso, o escalonador foi configurado para ser ativado quando a utilização do CPU está nos 80%, criando entre 1 a 3 *pods*.

Desta forma, garantimos a escalabilidade e o desempenho adequado do sistema, adicionando ou removendo *pods* conforme a carga de trabalho.

5 Monitorização, métricas e visualização

No que toca ao processo de monitorização, definimos a *role* **monitoring** que é executada aquando a criação do *cluster*. Nesse ficheiro, especificamos a criação de uma *policy* e de uma *dashboard* que permitem, respetivamente, a instalação de agentes responsáveis por recolher métricas das VMs existentes no *cluster* e a visualização das mesmas.

A monitorização é uma parte fundamental do processo pois permite verificar os resultados da instalação e o uso dos recursos. O processo de monitorização de uma aplicação pode ser dividida em quatro etapas: observação, recolha, análise e apresentação.

5.1 Ferramentas Utilizadas

Existem várias ferramentas para monitorizar o comportamento e os recursos da aplicação. No nosso caso, aproveitamos os serviços da *Google Cloud*, utilizando ferramentas como a *Kubernetes Metrics Dashboard* e respetivos agentes. Como foi referido, tanto a *dashboard* como os agentes são instalados durante a criação do *cluster* e acoplados a cada nodo do mesmo.

Para além disso, a *Google Cloud* oferece um serviço de *Logging* que permite rastrear problemas que possam surgir durante a execução da aplicação.

Assim, o sistema está preparado para analisar, recolher e apresentar os dados das componentes correspondentes ao Laravel.io e ao MySQL.

5.2 Métricas Utilizadas

A definição de métricas é relevante para o contexto do problema, uma vez que auxiliam na identificação de possíveis pontos de falha, de modo a permitir uma melhor gestão dos recursos.
Assim, resolvemos escolher métricas que avaliam:

- utilização de CPU
- utilização de RAM
- utilização de disco
- tempo de resposta

O tempo de resposta é uma métrica muito importante para avaliar a disponibilidade da aplicação. A utilização do CPU e da memória permitem verificar se os recursos utilizados foram adequados. Se a utilização for baixa, é possível que se estejam a desperdiçar recursos, se for muito alta, significa que será necessário aumentar os recursos.

Na figura abaixo podemos verificar estas métricas aplicadas ao *cluster*.

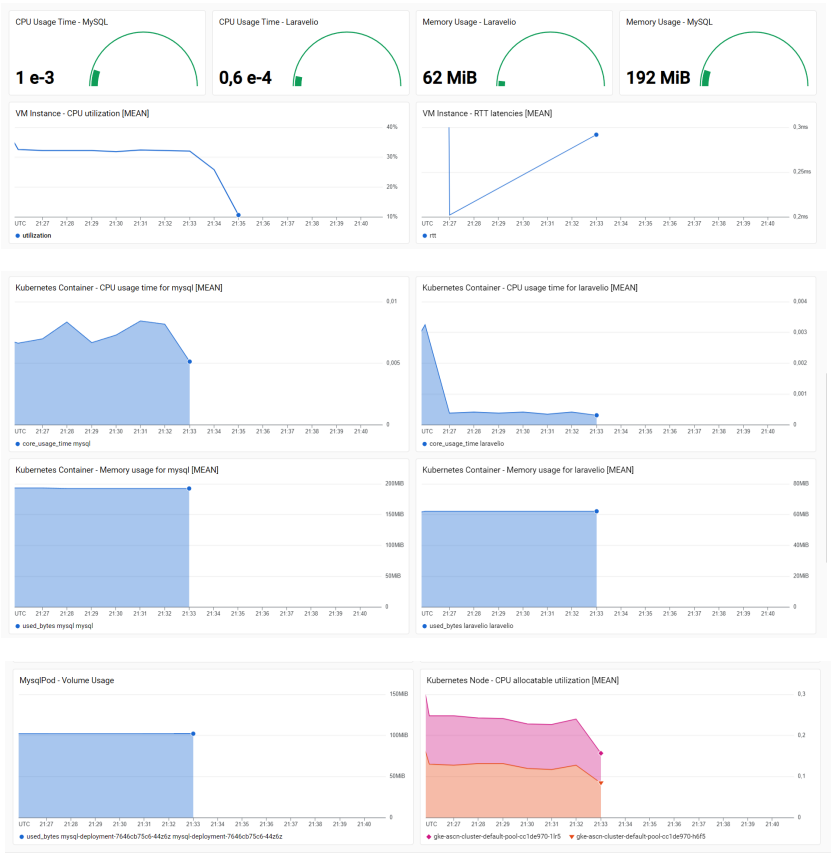

Figura 5: Métricas utilizadas.

6 Avaliação e Testes

Com o objetivo de avaliar o desempenho da aplicação com as otimizações aplicadas, utilizamos o **JMeter**. Esta ferramenta permite a realização de testes que simulam operações sobre o sistema por parte de uma grande afluência de utilizadores na aplicação. Desta forma, decidimos testar o acesso à página inicial da plataforma.

6.1 Análise dos Resultados

Antes de inicializar qualquer teste, avaliamos os recursos utilizados quando o *cluster* não está sob a qualquer tipo de carga, o que possibilita a comparação com os resultados obtidos durante os testes.

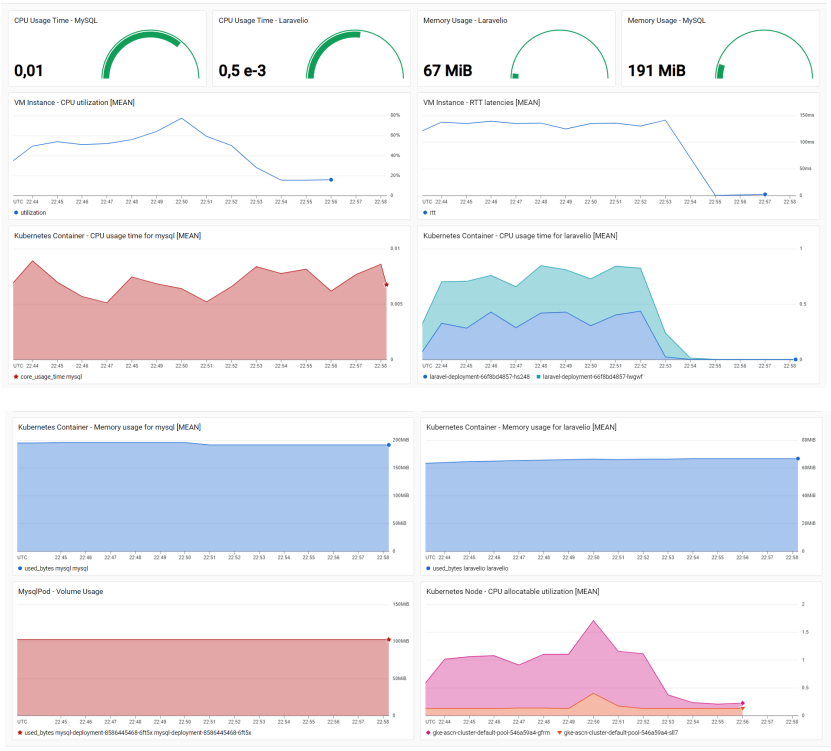

Figura 6: Valores de Referência.

6.1.1 Teste com 100 Threads

Como primeiro teste decidimos simular o acesso de 100 utilizadores há página principal da aplicação, pelo que podemos verificar os resultados obtidos na figura abaixo:

Figura 7: Teste realizado com 100 threads.

De acordo com as métricas obtidas, podemos verificou-se um aumento em cerca de 8 vezes no *CPU Usage Time*. Já a memória utilizada não sofreu mudanças significativas no desempenho durante o aumento do número de pedidos.

Para além disso, podemos verificar que inicialmente temos um *pod* responsável por atender os pedidos dos utilizadores do Laravel.io. No entanto, quando a utilização do CPU atinge o limite estipulado, o sistema trata de replicar o *pod* de modo a balancear a carga através do serviço *Load Balancer*. Esta abordagem afetou também o tempo de resposta aos pedidos dos clientes, uma vez que estabiliza ao fim de algum tempo não ultrapassando os 200 ms.

É importante realçar que para um número mais elevado de pedidos, o sistema não é capaz de replicar um terceiro *pod*, uma vez que a máquina utilizada não disponibiliza CPU suficiente para todos os serviços como é possível ver na imagem abaixo.

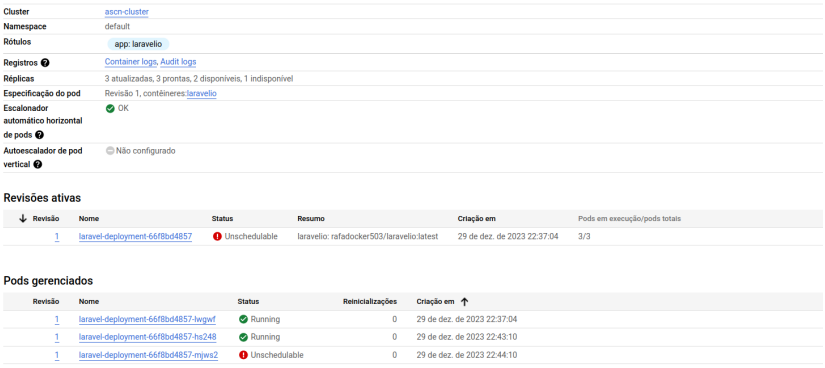

Figura 8: Falha na replicação do terceiro *pod*.

Assim, podemos concluir que as otimizações propostas foram benéficas para o desempenho e resiliência da aplicação, dado que permitiram por um lado, continuar a responder a pedidos em caso de falha no Servidor Aplicacional e, por outro, escalar o número de *pods* de acordo com a variação do número de utilizadores.

7 Considerações Finais

Após a conclusão deste trabalho prático, atribui-se um balanço positivo ao trabalho desenvolvido, já que permitiu praticar e consolidar os conceitos abordados tanto a nível teórico como prático.

Uma das principais dificuldades sentidas foi na implementação da *Horizontal Pod Autoscaling*, resultante de desafios relacionados com erros de configuração de CPU. Em um esforço para contornar essas dificuldades, implementamos um limite de CPU, uma vez que achamos que o problema estava relacionado com um elevado consumo de CPU por parte de algum *pod*. Para além disso, tentamos também modificar o tipo de máquina utilizada de *e2-small* para *e2-medium*.

Em termos de melhorias ou *upgrades* no nosso projeto, consideramos que o escalonamento vertical seria benéfico de implementar pois aumentaria a disponibilidade e o desempenho da aplicação e também o *node affinity* que é uma técnica permite uma maior otimização de *cache* e consistência de dados, aumentando assim a eficiência da aplicação.

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