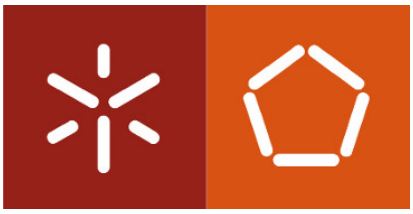

Universidade do Minho

Mestrado em Engenharia Informática

Aplicações e Serviços de Computação em Nuvem

Trabalho Prático

Grupo 11

Duarte Parente (PG53791) Gonçalo Pereira (PG53834)

José Moreira (PG53963) Santiago Domingues (PG54225)

Ano Letivo 2023/2024

Índice

Capítulo 1

Introdução

O presente relatório visa documentar as diversas etapas e soluções adotadas durante a execução das várias tarefas do trabalho prático desenvolvido no âmbito da Unidade Curricular de **Aplicações e Serviços de Computação em Nuvem**, que pretendia consolidar e aplicar os conhecimentos adquiridos ao longo do semestre de forma a automatizar a instalação, configuração, monitorização e avaliação da aplicação **Laravel.io**.

Na fase inicial, relativa ao primeiro *checkpoint*, o objetivo primordial passaria pela especificação de uma imagem **Docker** para a camada aplicacional do Laravel.io. A correta execução desta etapa surgiu como um requisito essencial para a realização do segundo *checkpoint*, garantindo uma base sólida para a conclusão da primeira tarefa definida para este trabalho prático, visando a automatização da instalação e configuração da aplicação. Para o efeito foi usada a ferramenta **Ansible** em conjunto com o serviço **Google Kubernets Engine (GKE)** da *Google Cloud*.

Relativamente à segunda tarefa, é esperado que seja realizada uma análise aprofundada ao desempenho, escalabilidade e resiliência da aplicação desenvolvida. A automatização de um serviço de **monitorização** revelou-se uma primeira etapa fundamental para a realização da tarefa, assim como a correta configuração e definição de testes de carga utilizando o **Apache JMeter**. Só com uma base sólida de *benchmarking* foi possível retirar ilações acerca de possíveis gargalos de desempenho e pontos únicos de falha na solução implementada. Por último, o componente relativo ao **servidor aplicacional** do `Laravel.io` foi o escolhido para a implementação da estratégia de replicação, com o objetivo de melhorar a escalabilidade e resiliência da aplicação.

Capítulo 2

Instalação e Configuração Automática da Aplicação

2.1 Camada Aplicacional do Laravel.io

De forma a cumprir com sucesso o *checkpoint* do projeto foi necessário recorrer às funcionalidades disponibilizadas pelo **Docker**, nomeadamente através da criação de uma imagem para a correta especificação da camada aplicacional da aplicação.

Primeiramente foi necessário estudar a documentação da própria aplicação de forma a detetar os passos necessários para a sua instalação e respetiva configuração, assim como as ferramentas e pacotes necessários para o seu correto funcionamento, nomeadamente:

- PHP 8.2
- Composer
- Npm

Foram também introduzidas variáveis de ambiente para controlo dos processos de migração (`migrate`) e povoamento(`seed_database`) da base de dados, assim como o processo de remoção dos dados persistentes da aplicação(`delete_data`).

2.2 Implementação da Solução

Uma vez que a primeira tarefa do trabalho prático, assim como o segundo *checkpoint* definiam como objetivo a correta automatização da instalação e configuração do Laravel.io, foi necessário recorrer às ferramentas **Ansible** e **Google Kubernetes Engine (GKE)**. A utilização dos módulos disponibilizados pelo Ansible revelou-se essencial para a redução da complexidade da interação com a plataforma da *Google Cloud*, mais concretamente com o serviço de **Kubernets**.

Atendendo aos requisitos definidos pela equipa docente a arquitetura da solução foi pensada de forma a que os seus componentes (aplicação e base de dados) executassem em *pods* distintos.

Para o efeito foram adicionados novos *roles* à estrutura definida pelo corpo docente nos critérios de desenvolvimento, responsáveis pelo *deploy* e *undeploy* dos pods relativos ao `Laravel.io` e ao `MySql`.

Foi ainda utilizado um **inventário**, peça fundamental em ambientes de automatização que recorrem à ferramenta Ansible. Este componente apresenta-se como um local centralizado para armazenar todas as configurações necessárias, simplificando a gestão e manutenção de variáveis específicas, conferindo uma maior flexibilidade e facilidade de adaptação dos *playbooks* e suas respetivas tarefas a diferentes contextos e cenários. No contexto da solução apresentada, o inventário é utilizado para armazenar variáveis relativas ao **cluster GKE**, nomeadamente a sua identificação, configuração e controlo de autenticação.

2.2.1 Laravel.io

O *pod* relativo à aplicação Laravel.io é configurado através de um *deployment* em Kubernetes, por intermédio da criação de um **container** que utiliza a imagem `Docker` criada para o primeiro *checkpoint*. Realça-se que uma vez que esta aplicação necessita de comunicar com a sua base de dados, são passadas diversas variáveis de ambiente aquando da criação do *container*, tais como: `DB_HOST`, `DB_DATABASE`, `DB_USERNAME` e `DB_PASSWORD`. Estas variáveis são fundamentais para possibilitar a conexão ao **serviço** do `MySql` que se encontra em execução no outro *pod*.

Para além da criação do *container*, e dada a necessidade de exposição da aplicação para efeitos de acesso e utilização externa, foi implementado um **Service** no pod em análise, possibilitando o acesso à aplicação por um qualquer cliente mediante a utilização de um navegador web. Na configuração deste serviço optou-se pela utilização do **LoadBalancer**, de forma a garantir um melhor controlo de tráfego entre os *pods*.

O endereço IP externo utilizado pelo `LoadBalancer` é calculado automaticamente através do módulo `gcp_compute_address` disponibilizado pelo Ansible. Este endereço é posteriormente atualizado automaticamente no inventário, tal como o valor em memória da respetiva variável (`app_ip`). A porta escolhida para a exposição da aplicação (**8000**) permanece inalterada durante todo o processo.

2.2.2 MySql

O *pod* relativo à base de dados da aplicação é também configurado através da criação de um **container** que utiliza a imagem `Docker` oficial do `MySql`. No entanto, de modo a que os dados armazenados sejam persistentes, ou seja, que se mantenham em armazenamento mesmo depois do tempo de vida do *pod*, foi utilizado um **Persistent Volume Claim (PVC)** associado a uma **StorageClass**, criada de forma a permitir que o **GKE** crie esta estrutura de armazenamento num cenário de necessidade.

Tal como no processo de exposição da aplicação Laravel.io, e uma vez que esta necessita de aceder e consumir a sua base de dados foi criado também um **Service** para este pod. Contudo, uma vez que o acesso a este serviço nunca poderá acontecer externamente ao **cluster**, é utilizado o tipo **ClusterIP** para a criação de um IP interno específico para este serviço.

Capítulo 3

Exploração e otimização da aplicação

3.1 Monitorização

A segunda etapa deste trabalho prático visa aprofundar a compreensão e otimização de desempenho, escalabilidade e resiliência da aplicação. Para alcançar esse objetivo, foi necessário implementar um processo autónomo de monitorização suficientemente abrangente e robusto de forma a permitir a análise de diferentes métricas de desempenho.

Sendo assim, e uma vez que a plataforma da *Google Cloud* oferece um sistema integrado de monitorização composto por uma ampla variedade de coleções de métricas e eventos, a decisão estratégica recaiu sobre a utilização desse mesmo sistema. No entanto, foi imediatamente evidente que, para além da falta de clareza na visualização e compreensão dos *dashboards* fornecidos, havia uma sobrecarga de informações desnecessárias à análise pretendida, acrescentando complexidade ao processo de observação mediante necessidade de filtragem de resultados.

No sentido de combater o cenário descrito anteriormente, e através da consulta da documentação da plataforma, optou-se pela implementação de uma **dashboard personalizada** povoada com as métricas de desempenho relevantes ao estudo pretendido e organizadas num formato natural ao acesso e análise da informação desejada.

A figura apresentada em baixo, procura mostrar precisamente um excerto da *dashboard* implementada. Denota-se que esta procura avaliar o desempenho da aplicação a partir de métricas como a utilização de recursos, nomeadamente **CPU** e **RAM**, assim como avaliar o **throughput**. Para a concretização da automação da criação desta janela foi acrescentado um *role* de monitorização à solução apresentada no capítulo anterior (`monitor`). Este *dashboard* é criado imediatamente após a inicialização da aplicação podendo ser prontamente consultado na categoria `Custom` inserida na secção de monitorização da consola da plataforma *Google Cloud*.

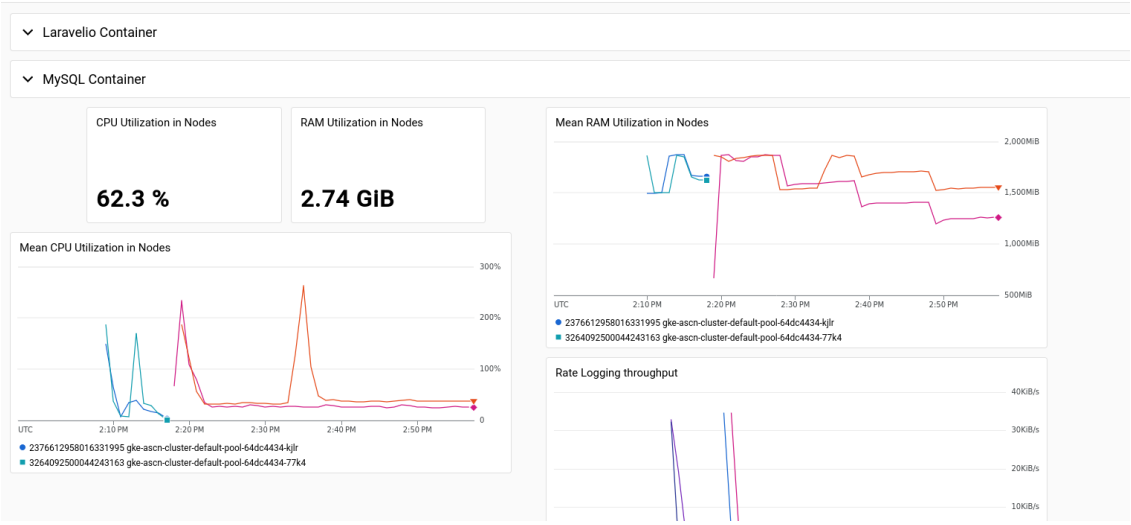

Figura 1: Dashboard de monitorização personalizada

3.2 Benchmarking

Com a configuração adequada do serviço de monitorização, estão estabelecidas as condições necessárias para prosseguir para a fase de avaliação experimental. Nesta etapa, serão conduzidos diversos testes de carga para avaliar a resposta do sistema perante os cenários propostos. Para o efeito, utilizou-se a ferramenta **Apache JMeter**.

Para a realização da avaliação foram definidos dois cenários, com variações na complexidade das ações de forma a estudar diferentes cargas associadas a dois tipos de ações diferentes.

- **Cenário 1** - Representa uma simples e breve interação. Um utilizador faz *login* na aplicação, consulta a sua página de perfil e de seguida efetua o *logout*.
- **Cenário 2** - Este cenário procura explorar diferentes vertentes e funcionalidades da aplicação, realizando ações com maior carga computacional. Um utilizador acede à sua conta, consulta a página de fóruns e publica uma nova **thread**.

Para além dos dois cenários de interação definidos, efetuaram-se as medições com um valor aleatório de *delay* entre cada pedido, num intervalo de 0 a 5s. Esta abordagem surge como tentativa de aproximação a um **cenário real**, onde não é possível prever com exatidão a quantidade e cadência de acessos à aplicação num determinado momento. A introdução desta imprevisibilidade e variabilidade contribui para a relevância, e inclusive validação, dos resultados e respetivas conclusões obtidas a partir dos testes.

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

Na avaliação experimental, a definição de expectativas e objetivos claros é considerada uma boa prática e um passo fundamental para a correta avaliação do sistema. Em particular, o re-

conhecimento de gargalos de desempenho é também uma etapa essencial para a otimização do desempenho aplicação e da garantia da sua escalabilidade.

Em relação à solução implementada e descrita no capítulo anterior, há diversos componentes que poderão pôr em risco o desempenho do sistema em situações de sobrecarga. Primeiramente, o **servidor aplicacional** como componente essencial desta aplicação poderá tornar-se um ponto crítico num cenário de confronto com um grande número de solicitações simultâneas para as quais não está devidamente dimensionado. Para além disso, e uma vez que a aplicação se encontra em execução num ambiente **Kubernetes**, os **recursos alocados**, nomeadamente `CPU` e `RAM`, num cenário de má configuração ou alocação nos *pods* poderão também levar a gargalos de desempenho devido a limitações dos mesmos.

1. b) Qual o desempenho da aplicação perante diferentes números de clientes e cargas de trabalho?

Segue-se a apresentação em formato tabular dos resultados obtidos nas diversas medições efetuadas.

	Cenário 1			Cenário 2		
Nº Requests	20	100	500	20	100	500
% CPU (Nodes)	93.1	124	163	97.9	138	91
RAM (Nodes)	2.73 GB	2.73 GB	2.98 GB	2.68	2.7 GB	2.83 GB
Throughput (pedidos / sec)	2.3	1.3	0.797	2.3	1.2	0.625
CPU Usage Time (Laravelio)	0.186	0.914	0.988	0.185	0.907	1.016
CPU Usage Time (MySql)	0.0174	0.1082	0.1561	0.02	0.1223	0.342
RAM Usage (Laravelio)	136.64 MiB	136.64 MiB	132.63 MiB	136.26 MiB	136.26 MiB	136.26 MiB
RAM Usage (MySql)	402.16 MiB	402.16 MiB	398.56 MiB	397.4 MiB	397.4 MiB	397.4 MiB

Figura 2: Resultados obtidos na avaliação experimental

Os resultados acima representados, à exceção do valor do **throughput**, foram todos obtidos a partir da *dashboard* apresentada na secção de monitorização, através da extração do valor máximo de cada parâmetro em cada medição. O valor associado ao *throughput* foi obtido através do relatório do teste apresentado pelo `JMeter`.

Analisando os resultados obtidos é desde logo percetível o valor constante de `RAM` a ser utilizada, o que poderá indicar que a gestão de memória está a ser efetuada de forma eficiente. O aumento da utilização do CPU conforme o aumento do número de solicitações vai de encontro ao resultado esperado, uma vez que a carga computacional associada também aumenta. Já em relação ao valor do *throughput*, verifica-se uma diminuição acentuada do número de pedidos atendidos por segundo conforme o aumento do número de clientes. Esta quebra no desempenho associada à crescente utilização do CPU poderá indicar que a aplicação está a atingir o seu limite de capacidade, com o aparecimento de um gargalo de desempenho.

1. c) Que componentes da aplicação poderão constituir um ponto único de falha?

A identificação de pontos únicos de falha é crucial para a garantia de resiliência e alta disponibilidade da aplicação. A **base de dados**, como componente essencial da aplicação, poderá impactar severamente o funcionamento do sistema num cenário de falha. Para além desse componente, o **servidor aplicacional** aparece também com um papel nuclear na disponibilidade da aplicação, uma vez que o acesso à mesma está dependente dele. Em relação ao ambiente de *deployment*, os **recursos utilizados** como a `Rede`, `CPU` e `RAM`, assim como o **LoadBalancer** terão de estar bem configurados e em bom funcionamento para a garantia de disponibilidade da aplicação.

Todos estes componentes poderão consituir um ponto único de falha na aplicação. De forma a mitigar esta dependência e falha na resiliência da aplicação é fundamental implementar estratégias de distribuição e replicação.

3.3 Otimização da aplicação

2. a) Que otimizações de distribuição/replicação de carga podem ser aplicadas à instalação base?

Após a análise realizada ao desempenho da instalação base, tornou-se claro que para melhorar a escalabilidade e resiliência de um dos componentes da aplicação seria necessário replicá-lo. O componente escolhido para a realização da tarefa foi o **servidor aplicacional**.

Sendo assim, optou-se por adicionar à solução implementada um mecanismo de criação de réplicas do *pod* do servidor aplicacional do `Laravel.io` no sentido de conferir um maior poder computacional à solução, para além de uma maior resiliência, uma vez que num cenário de indisponibilidade de uma das réplicas continuavam a existir outros pontos de acesso.

Para o efeito utilizou-se o **HorizontalPodAutoscaler**, configurado de forma a poder ser aplicado na arquitetura implementada. Este foi programado para ser ativado num cenário de sobrecarga da aplicação, ou seja com um valor limite de 75% de taxa de utilização do `CPU`. O componente é iniciado com 1 única réplica, e uma vez que foi definido que o número de nodos a serem criados é 2, este é o número máximo de réplicas que poderão ser criadas.

Este número poderia ter sido modificado e aumentado para outro valor, conferindo a possibilidade de criação de um maior número de réplicas. Contudo, optou-se por manter o valor de base uma vez que a esta modificação está associado um aumento do custo operacional e *overhead* computacional relacionado com o controlo e manutenção das interações entre as diferentes réplicas.

Por último, salienta-se que a escolha da utilização do **LoadBalancer** possui um papel importante nesta implementação, uma vez que este possui a responsabilidade de distribuir a carga pelas diferentes réplicas criadas, evitando sobrecargas num único ponto e garantindo a utilização efeciente dos recursos.

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

A implementação da estratégia de escalabilidade horizontal, com o uso do **HorizontalPodAutoscaler**, permite que a aplicação se ajuste dinamicamente à carga de trabalho. Esta característica resulta numa otimização do desempenho da aplicação, uma vez que o aumento da sobrecarga de solicitações poderá ser atendida por novas réplicas do servidor aplicacional. Para além disso, a existência de réplicas do servidor aplicacional atenua o risco de falha do sistema provocado diretamente pela falha deste componente essencial para o funcionamento do mesmo.

Capítulo 4

Conclusão

Realizando uma análise crítica e aprofundada de todo o trabalho realizado relativamente às duas principais tarefas definidas para este trabalho prático, podemos afirmar com confiança que uma grande parte dos requisitos estabelecidos foi integralmente cumprida. A solução final apresentada permite constatar um *deployment* totalmente funcional e automatizado da aplicação `Laravel.io`. Para além disso, e respeitando os critérios definidos pela equipa docente, as soluções de otimização, assim como os processos de monitorização e avaliação encontram-se na sua integridade em *playbooks* `Ansible` de forma a poderem ser executados e reproduzidos de forma automática.

A execução bem-sucedida do trabalho prático permitiu a exploração e consolidação de boas práticas de **DevOps**, e à utilização eficiente de tecnologias para um *deployment* na *cloud* com uma base robusta para futuras melhorias e expansões.

Bibliografia

[1] https://www.codingful.com/how-to-use-jmeter-to-test-a-login-page-with-a-csrf-token/

[2] https://github.com/GoogleCloudPlatform/monitoring-dashboard-samples/blob/master/dashboards/google-kubernetes-engine/gke-active-idle-clusters.json

[3] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/