Universidade do Minho

Mestrado em Engenharia Informática

Aplicações e Serviços da Computação em Nuvem

Trabalho Prático

Deployment e Management da plataforma Lavarel.io

GRUPO 12

Anabela Pereira (PG49995)

André Vieira (PG50219)

Gonçalo Medeiros (PG50399)

Joana Alves (PG50457)

Vicente Moreira (PG50799)

Dezembro 2023

Conteúdo

1 Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de **Aplicações e Serviços da Computação em Nuvem** do Mestrado em Engenharia Informática da Universidade do Minho. Este tem como objetivo a automatização, instalação, configuração, monitorização e avaliação da aplicação **Laravel.io** através da ferramenta **Ansible**.

Neste relatório será apresentada uma descrição detalhada da aplicação *Lavarel.io*, nomeadamente uma análise da arquitetura geral da implementação do sistema, incluindo os vários *playbooks Ansible* presentes no mesmo.

De seguida, abordamos e explicitamos as ferramentas utilizadas, tendo estas sido referidas e exploradas ao longo do semestre na unidade curricular, como por exemplo, *Ansible*, *Google Kubernetes Engine*, *GCP Dashboards* e, finalmente, *JMeter*. Para além disto, será também apresentada a abordagem seguida no desenvolvimento do trabalho prático, incluindo uma breve descrição de cada *playbook Ansible*.

Por último, serão apresentadas todas as observações e decisões tomadas na monitorização e avaliação da ferramenta, dando resposta às questões colocadas no enunciado do trabalho prático.

2 Arquitetura Geral

Nesta secção começamos por apresentar uma arquitetura geral do projeto desenvolvido, quer seja pela arquitetura de ficheiros presentes no repositório deste, ou pela a apresentação de um esboço da arquitetura presente no momento de aprovisionamento da aplicação *laravel.io* no *Kubernetes Engine*.

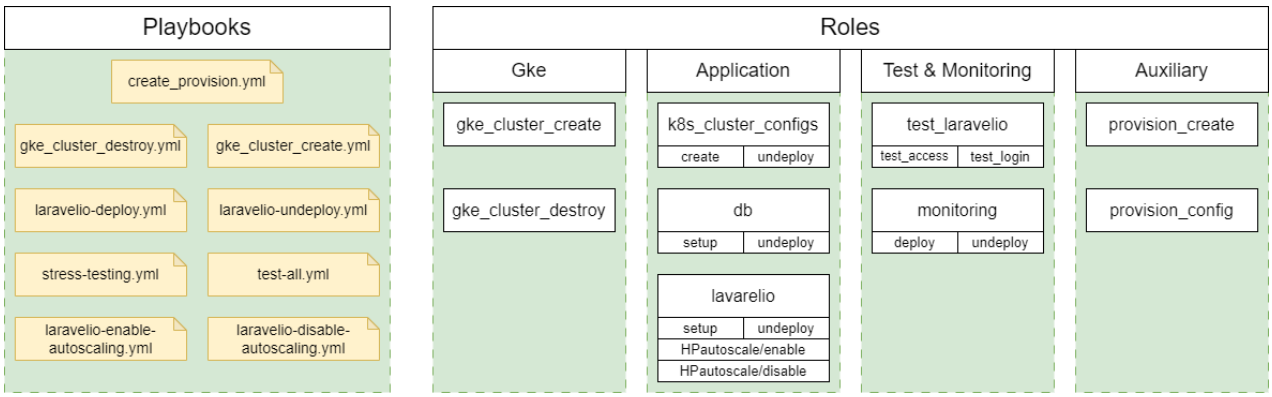

Figura 1: Arquitetura do repositório

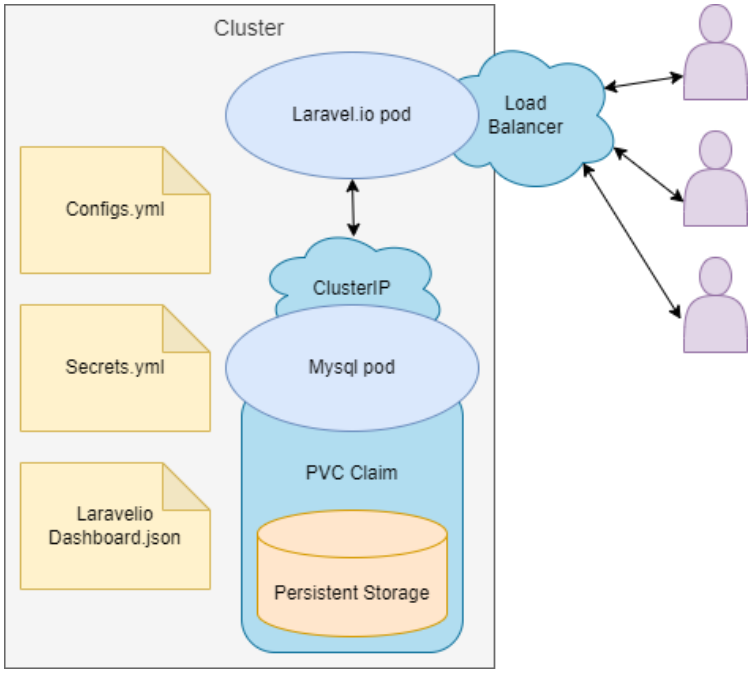

Figura 2: Arquitetura Kubernetes

3 Ferramentas

A primeira ferramenta utilizada neste trabalho prático foi o *Ansible*, abordado e explorado nas aulas práticas. O *Ansible* é uma ferramenta de automação *open-source* utilizada para simplificar tarefas de gestão, configuração e implementação de sistemas e aplicações. O **Ansible** utiliza *YAML* como linguagem para descrever o seu inventário de máquinas no sistema, configurações, *playbooks* e os vários *roles*/tarefas do sistema. Estas tarefas e comandos serão depois executadas nos sistemas alvo através de *SSH*.

Para o desenvolvimento utilizou-se o *Google Cloud Plataform (GCP)*, nomeadamente o serviço *Google Kubernetes Engine (GKE)*, sendo este um sistema *open-source* escalonável e automatizado. Tirando proveito desta ferramenta, apenas nos focamo no desenvolvimento da aplicação *Lavarel.io* em *containers*, enquanto que o *Google Cloud* gere a infraestrutura.

Para a tarefa de monitorização e avaliação utilizou-se a ferramenta do *Dashboard* do *GCP*, apesar da ferramenta explorada na aula ter sido o *Elasticsearch* e o *Kibana*. Por um lado, o *Elasticsearch* é um software de recolha de dados de performance de sistemas, utilizado para armazenar, indexar e permitir a busca em grandes volumes de dados de forma eficiente e rápida. Por outro lado, o *Kibana* é uma interface para a exploração, visualização e interação com os dados armazenados no *Elasticsearch*, permitindo criar *dashboards* interativos, gráficos e tabelas para analisar os dados recolhidos pelo *Elasticsearch*. Porém o grupo optou pelo *Dashboard do GCP*, visto que, oferece um conjunto integrado de recursos de monitorização, facilitando a configuração, permitindo a uma maior facilidade de utilização.

Por último, utilizou-se o *JMeter* para testar e medir o desempenho da nossa implementação, visto que é uma ferramenta de teste de desempenho e carga, desenvolvida pelo projeto *Apache* e, sendo também esta explorada nas aulas práticas da unidade curricular.

4 Abordagem

Para a realização deste projeto, foram desenvolvidos no total 8 *playbooks*:

- **gke-cluster-create**
- **gke-cluster-destroy**
- **laravel-deploy**
- **laravel-undeploy**
- **test-all**
- **stress-testing**
- **laravelio-enable-autoscaling**
- **laravelio-disable-autoscaling**

Destes *playbooks*, mostramos de seguida o funcionamento dos *playbooks* dedicados à criação do *cluster* e ao processo de instalação e configuração da plataforma, assim como os *playbooks* utilizados para o *undeployment* do sistema e destruição do *cluster*.

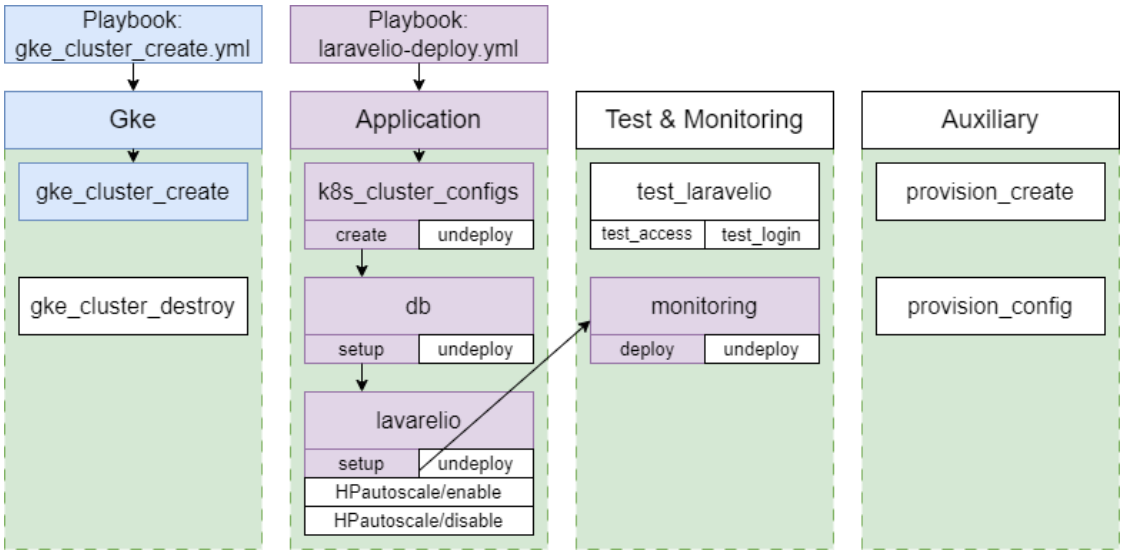

Figura 3: Processo de *Deployment* da plataforma Laravel.io

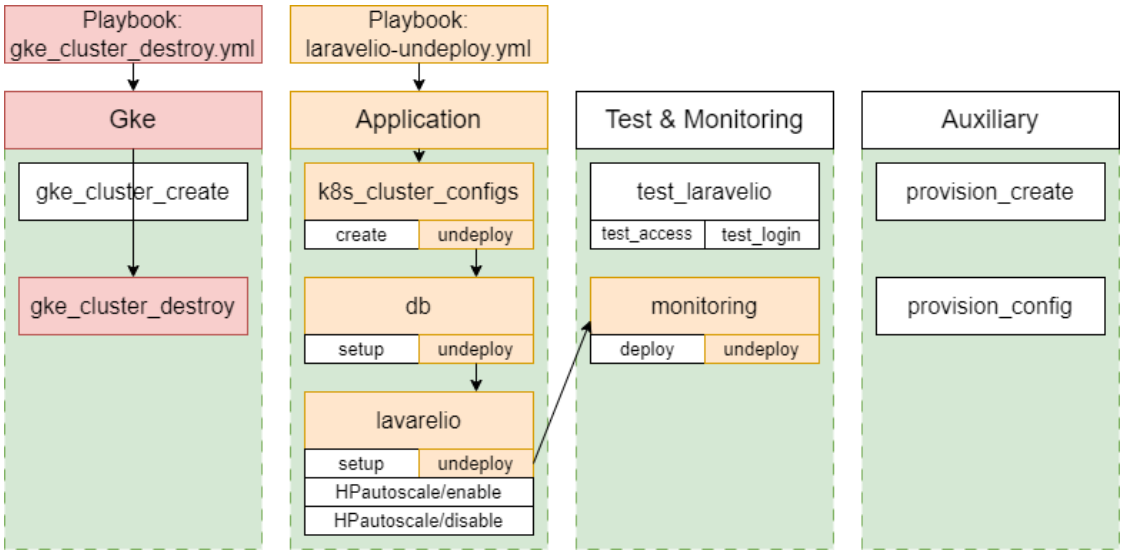

Figura 4: Processo de *Undeployment* da plataforma Laravel.io

De seguida incluímos uma listagem de todos os *roles* no projeto, assim como um pequeno sumário do que é executado:

- **gke_cluster_create:** Criação do *Google Kubernetes Cluster*.
- **gke_cluster_destroy:** Eliminação do *Google Kubernetes Cluster*.
- **k8s_cluster_configs**
  - **create:** Cria o *namespace* utilizado no projeto e define o uso deste por defeito, assim como cria no *cluster* os recursos 'Secret' e 'ConfiMap', responsáveis por, respetivamente, guardar a *password* e as variáveis de configuração da base de dados.
  - **delete:** Neste passo, apenas são eliminados os recursos 'Secret' e 'ConfigMap'.
- **bd**
  - **setup:** Aprovisionamento da base de dados, através da criação de um **PVC** (Persistent Volume Claim) de **30Gi** no modo ***ReadWriteOnce***, um **Service** que expõe a porta 3306 ao *cluster* **(ClusterIP)** e o *deployment* da aplicação da base de dados **mysql**, utilizando os valores configurados no ConfiMap e com a utilização do PVC gerado. Por último, este *pod* é testado para avaliar o sucesso do aprovisionamento.
  - **undeploy:** *Undeployment* da base de dados, assim como o seu serviço e o PVC.
- **laravelio**
  - **setup:** Neste role, começamos por efetuar o *deployment* do *container* da aplicação *laravel.io*, utilizando uma imagem *docker* criada a partir de um *Dockerfile*, fornecendo também os valores corretos da base de dados a utilizar e a definição da porta do *container*. De seguida é criado um serviço para expor esta porta ao público, este serviço é do tipo **LoadBalancer**, para que este faça a gestão automática de carga no sistema, e faz a tradução da porta interior do *container* (8000) para a porta exterior *default* do HTTP (80).
    Também são efetuados vários comandos com o objetivo de obter e gravar o IP atual da aplicação, para que mais tarde este possa ser utilizado pelos *playbooks* de teste. Este processo deve-se à atribuição de IP aleatória no aprovisionamento da aplicação. Por último, é avaliado se a aplicação se encontra disponível.
  - **undeploy:** *Undeployment* do *laravel.io*, assim como o seu serviço.
  - **HPautoscaling/enable:** Permite o escalonamento automático da aplicação *Laravel.io*, com base na utilização do CPU dos *pods*. O escalonamento é do tipo **HorizontalPodAutoscaler** e tem como objetivo gerar novos *pods* réplicas de forma a suportar cargas maiores. Para este foi definido um limite máximo de 5 réplicas.
  - **HPautoscaling/disable:** Desliga o escalonamento automático do *Laravel.io*

- **test_laravelio**
  - **test_access:** *Role* para teste simples do funcionamento da aplicação, testando apenas se o *website* responde corretamente.
  - **test_login:** *Role* para teste da funcionalidade de *login* do *website*, com o objetivo de testar mais profundamente a lógica da aplicação, assim como o funcionamento correto da base de dados através da operações com o *token* de sessão.
- **monitoring**
  - **deploy:** Criação da *Dashboard* a ser utilizada para a monitorização do *cluster*.
  - **undeploy:** Eliminação da *Dashboard* do *cluster*.
- **provision_create:** *Role* para a definição e criação de uma máquina virtual. Esta máquina tem como objetivo poder ser utilizada como máquina de aprovisionamento do projeto.
- **provision-config:** Configuração da máquina virtual, instalando todas as dependências necessárias para o funcionamento da ferramenta *Ansible*, assim como configurações adicionais do *Google Cloud*.

5 Monitorização e Avaliação

5.1 Dashboard

Para a monitorização da nossa aplicação, o grupo decidiu utilizar as funcionalidades de monitorização incluídas pela plataforma do *Google Cloud*, através do uso de *dashboards* pré-definidas e personalizadas. O grupo definiu uma *dashboard* personalizada que contém várias métricas como a latência de resposta, a percentagem de utilização do CPU nos nodos, o *throughput* de dados dentro do sistema, a quantidade de memória ocupada e o tráfego de pacotes no *cluster*. Com a *dashboard* criada decidimos gravar esta num ficheiro **.json** e de seguida automatizar o *deployment* da *dashboard* quando se efetua o aprovisionamento do *Laravel.io*. Apresentamos de seguida uma parte da dashboard desenvolvida.

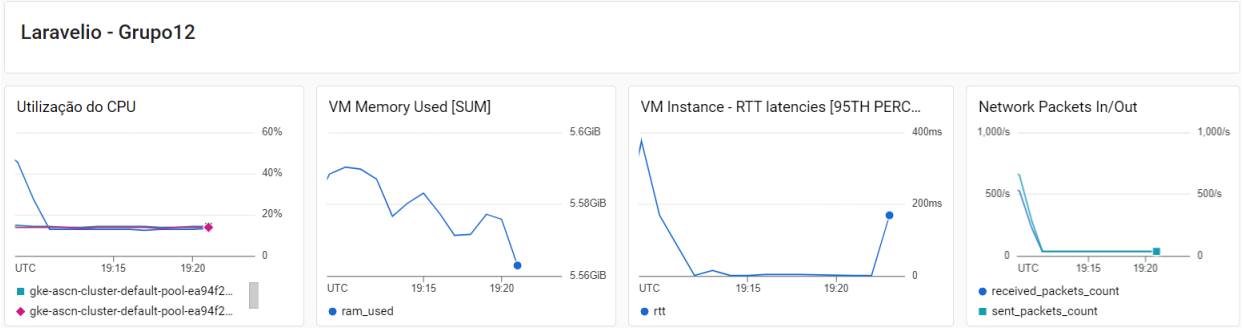

Figura 5: *Dashboard* utilizada

5.2 Avaliação

Com a utilização das *dashboards*, podemos avaliar a performance do sistema ao longo da sua utilização, assim como recolher informação sobre o estado dos vários recursos do *cluster*, de forma a perceber potenciais problemas existentes no sistema.

Para avaliar o sistema durante períodos de stress e alta carga, a equipa utilizou a ferramenta **jmeter** para simular uma grande quantidade de clientes a efetuar pedidos na aplicação. Começamos por testar o sistema base simples, simulando, respetivamente, 10 e 100 clientes.

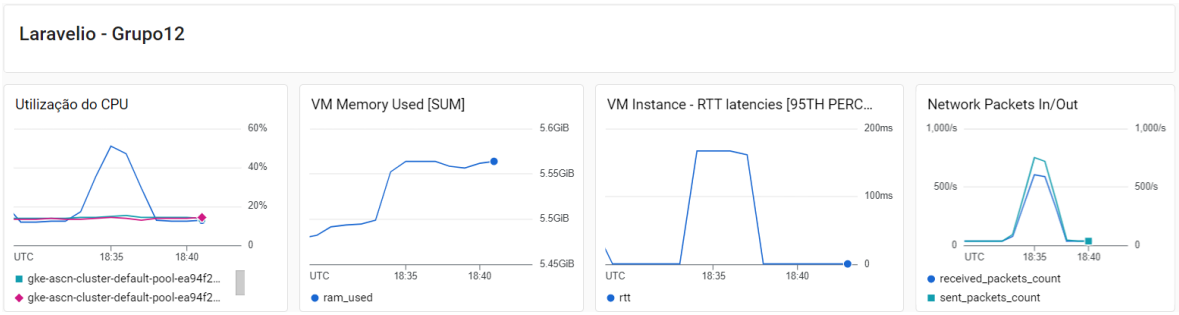

Figura 6: Teste 1 - 10 clientes

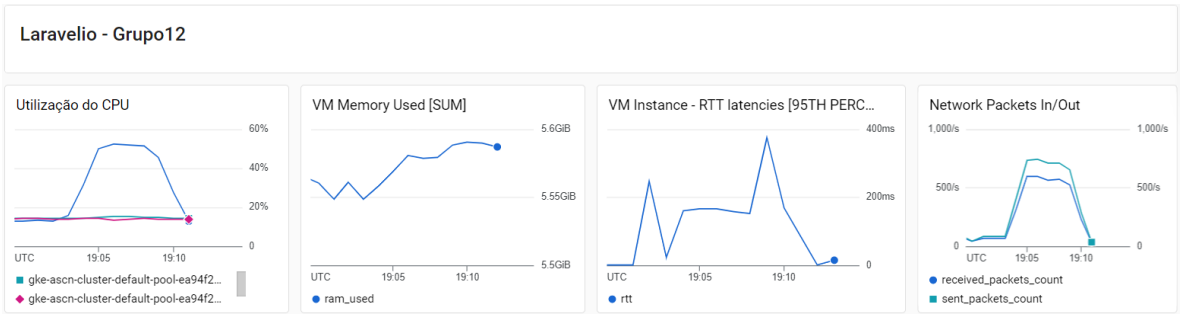

Figura 7: Teste 2 - 100 clientes

Pela observação dos gráficos acima, podemos concluir que apesar das restantes métricas se manterem moderadamente estáveis, a latência de resposta no teste de 10 clientes não subiu para além dos 150ms, no entanto, para 100 clientes, este já alcançou cerca de 400ms, um diferença notável.

Para os próximos testes, decidimos ligar e testar o mecanismo de *autoscaling* desenvolvido para o *pod* do *Laravel.io*. Este deverá ser capaz de, através da percentagem de utilização do CPU, detetar um alto nível de carga e responder através da criação de novos *pods*, de forma a reduzir a carga do sistema. Para estes testes, decidimos simular 100 e 1000 clientes.

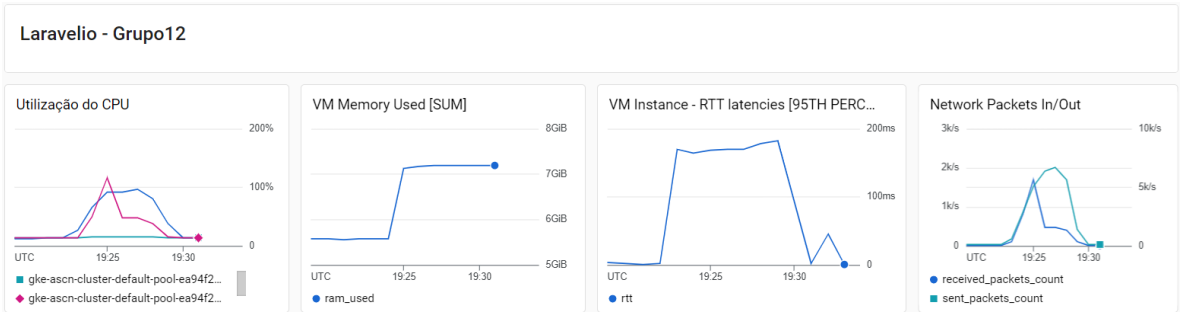

Figura 8: Teste 3 - 100 clientes - AutoScaling

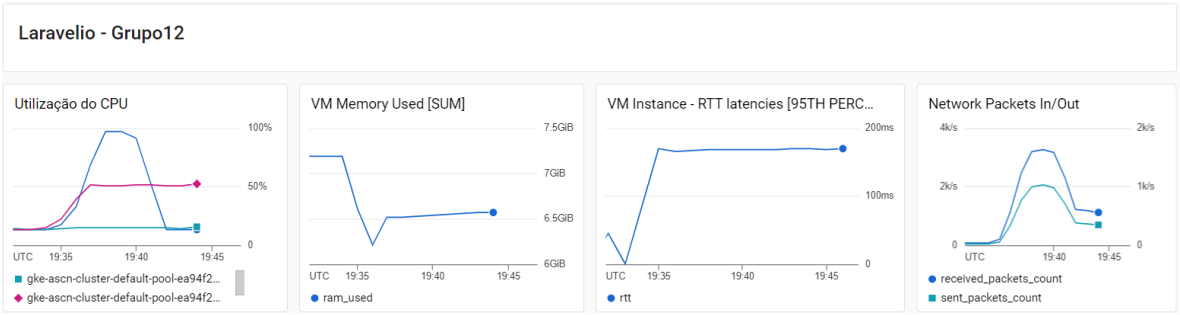

Figura 9: Teste 4 - 1000 clientes - AutoScaling

Estes testes demonstram que, para o mesmo número de clientes ou até um número muito mais elevado, a resposta do mecanismo de *autoscaling* ajudou na estabilização do sistema, permitindo que este conseguisse manter a sua performance geral, como se pode observar através da métrica de latência e tráfego de pacotes.

6 Tarefa 2

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

Para um número crescente de clientes, tanto a base de dados, como o *laravel.io* são dois pontos de gargalo de desempenho, visto que, estão *deployed* apenas num *pod*. No caso do *pod* da base de dados, todas as consultas realizadas serão feitas ao mesmo *pod*, levando a tempos de execução mais longos, impactando o desempenho global do sistema. No caso do *pod* do *laravel.io*, irá partilhar os mesmos recursos e pode ser sobrecarregado com muitas solicitações simultâneas.

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

Como se pode observar pelos testes anteriores, com um número elevado de clientes o tempo de resposta do sistema aumenta, levando a problemas de desempenho do sistema, falhas na comunicação entre clientes e, possivelmente, falhas no serviço como um todo.

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

Componentes como o *pod* responsável pelo *Laravel.io* poderão constituir um ponto único de falha, pois caso este tenha algum problema, o sistema falha por todo. No caso de existirem réplicas deste *pod*, um outro ponto de falha poderá ser a componente de serviço *LoadBalancer* usada, visto que todo o tráfego terá de navegar por essa componente para alcançar a aplicação.

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

As otimizações que podem ser aplicadas às componentes da instalação base é o escalonamento horizontal e/ou vertical.

O escalonamento horizontal permite criar vários *pods*, distribuindo assim a carga tanto para o *laravel.io* (replicação de computação) quanto para a base de dados (replicação de dados) por meio desses *pods*. Para além disto, permite lidar com um aumento de pedidos sem que o sistema se torna um gargalo de desempenho.

O escalonamento vertical permite aumentar os recursos de cada *pod*, levando a uma maior capacidade de resposta, no entanto, o problema de ponto único de falha e/ou gargalo de desempenho mantém-se.

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

Como foi observado nos testes anteriores, com o auto-escalamento horizontal o sistema foi capaz de responder a um maior número de pedidos sem haver qualquer impacto de performance. Este método por norma consome mais recursos mas estes apenas são consumidos temporariamente, e apenas quando necessário. Outro benefício vem na forma de resiliência do sistema por redundância das suas componentes.

7 Conclusão

Com a conclusão do trabalho prático, o grupo encontra-se satisfeito com o trabalho desenvolvido, tendo sido alcançados todos os objetivos estabelecidos quer pelos docentes, como pelo próprio grupo.

Para além disto, este trabalho permitiu também aprofundar e desenvolver os conhecimentos lecionados nas aulas da unidade curricular, como automatização de tarefas (p.ex. *deploy* de aplicações) através da ferramenta *Ansible*, e monitorização e avaliação de sistemas através de ferramentas como *Dashboard* do *GCP* e *JMeter*, respetivamente.