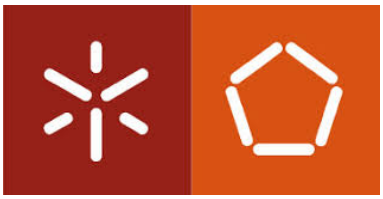

Universidade do Minho
Mestrado em Engenharia Informática

Aplicações e Serviços de Computação em Nuvem

Trabalho Prático

14 de janeiro de 2023

António Santos (PG47031)
Carlos Ferreira (PG47087)
Joel Martins (PG47347)
Manuel Moreira (PG47439)
Sara Dias (PG47667)

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O trabalho prático proposto tem como objetivo principal automatizar o processo de instalação, configuração, monitorização e avaliação da aplicação Ghost (https://ghost.org). Para tal, foi necessário recorrer aos conhecimentos obtidos durante as aulas desta Unidade Curricular.

O trabalho prático é constituído por quatro tarefas, das quais a tarefa base, tem como principal objetivo a instalação e configuração automatizada da aplicação Ghost no serviço Google Kubernetes Engine (GKE) da Google Cloud. Para tal, foram efetuadas as seguintes tarefas:

- Compreensão da arquitetura e componentes da aplicação
- Identificação dos componentes cujo desempenho e resiliência são cruciais.
- Compreensão do Google Kubernetes Engine.

A primeira tarefa objetiva na aplicação de ferramentas de monitorização para observar a instalação da aplicação. Para tal, as seguintes tarefas foram planeadas:

- Exploração das ferramentas de monitorização fornecidas pelo Google Cloud.
- Seleção e configuração automatizada da ferramenta de monitorização.
- Visualização e avaliação dos dados resultantes da ferramenta.

A segunda tarefa objetiva na aplicação de métodos de validação para avaliar experimentalmente a instalação e configuração da aplicação. Para tal, as seguintes tarefas foram planeadas:

- Avaliar as diversas funcionalidades da aplicação, assim como os seus componentes, tendo em consideração traces reais.
- Repetição do processo anterior, agora com *workloads* sintéticos.
- Automatização do processo execução de testes.

Na última tarefa pretende-se incorporar na aplicação mecanismos de replicação, que permitam melhorar o desempenho e a resiliência da instalação da aplicação. Para tal, as seguintes tarefas foram planeadas:

- Avaliar e selecionar o componente a ser utilizado no processo.
- Automatização da gestão do número de réplicas.
- Avaliação da eficácia do mecanismo de replicação.

Assim, aproveitando os conceitos aprendidos nas aulas da UC, juntamente com alguma pesquisa efetuada, tentamos cumprir com o objetivo do trabalho prático, apresentado detalhadamente ao longo dos próximos capítulos.

Capítulo 2

Tarefa base

2.1 Descrição de Arquitetura e Componentes

A aplicação Ghost trata-se de um sistema de gerenciamento de conteúdo (Content Management System) criado para criar e gerenciar *blogs*. O Ghost permite que os utilizadores escrevam publicações e possam gerir as suas próprias páginas, incluindo opções para adicionar imagens, vídeos e outros conteúdos multimédia. Esta aplicação é construída com Node.js, o que a torna escalável e personalizável.

Tal como planeado, foi inicialmente efetuado o estudo aprofundado da arquitetura da aplicação de forma a perceber os componentes mais críticos. Assim, foi possível identificar que a plataforma apresenta a arquitetura de uma aplicação Web, baseada em micro-serviços, tal como consta na seguinte imagem 2.1, extraída da própria documentação da respetiva aplicação.

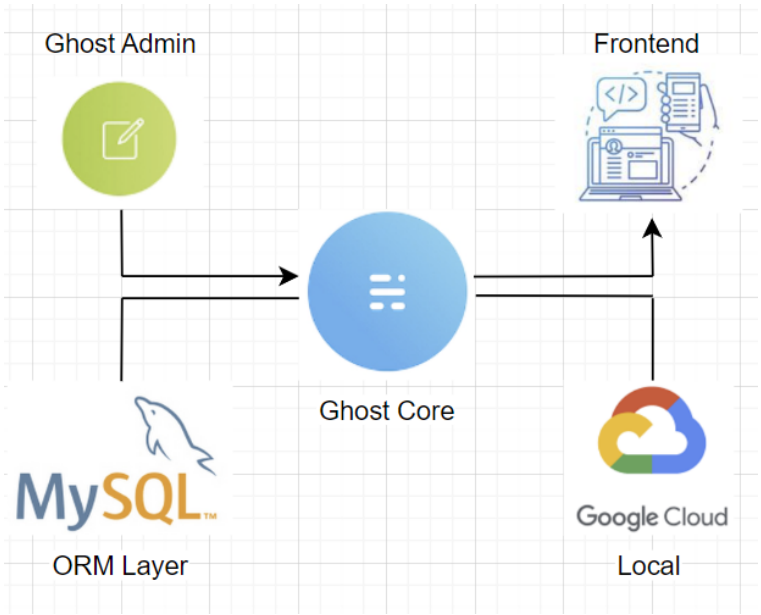

Figura 2.1: Arquitetura do Ghost

Deste esquema podemos retirar as principais componentes da aplicação: o **Frontend**, o **Core**, a camada **ORM** e finalmente o adaptador de **armazenamento**.

2.1.1 Frontend

Sendo um sistema de gestão de conteúdo *headless*, esta aplicação não contém uma componente de Frontend definida, sendo completamente agnóstica para qualquer *framework* utilizável. Nesta arquitetura será utilizada a componente *frontend* que vem na instalação base do Ghost.

2.1.2 Ghost Core

Tendo em conta a arquitetura do Ghost, podemos ver que o Core se trata de uma RESTful JSON API, que tem a função de criar, gerir e obter facilmente o conteúdo de uma publicação. Esta API encontra-se dividida em duas partes: *Content API* e *Admin API*, cada uma com métodos de autenticação e estrutura específicos, tendo por isso as ferramentas necessárias para criar publicações comuns com o mínimo esforço.

Em relação ao *Content API*, sabemos que este trata de disponibilizar o conteúdo publicado e pode ser acedido por qualquer cliente, podendo este aceder aos dados quantas vezes quiser sem limites.

No caso do *Admin API*, esta componente tem permissões de leitura e escrita, e trata de criar, gerir e atualizar o conteúdo das publicações. Esta API fornece autenticação segura para que se possa publicar de qualquer lado sem problema. Quando se está autenticado como administrador, esta API fornece controlo total para criar, editar e apagar toda a informação da publicação do utilizador.

Para além disso, o Ghost Core oferece ainda um cliente API e SDK do JavaScript, feito para facilitar a autenticação e o acesso aos dados, assim como *webhooks* que irão notificar um serviço externo da alteração do conteúdo.

2.1.3 Camada ORM

Esta aplicação utiliza a biblioteca *Bookshelf.js* de forma a possibilitar a comunicação entre a aplicação com a camada de base de dados. Esta biblioteca permite que aplicações feitas em *javascript* comuniquem com bases de dados relacionais, tais como SQLite, MySQL, PostgresSQL, etc. Desta forma, existe flexibilidade na escolha da base de dados a utilizar na arquitetura do sistema.

Na realização deste trabalho, foi usado o MySQL, devido ao suporte fornecido pela equipa da aplicação, tal como a familiaridade com o sistema da base de dados proveniente do desenvolvimento dos guiões das aulas práticas.

2.1.4 Armazenamento

A aplicação apresenta também flexibilidade nos métodos de Armazenamento, oferecendo diversas opções, tais como Amazon S3, Azure, ou localmente. No desenvolvimento deste trabalho, adotou-se uma abordagem de armazenamento local, tendo sido criado um *persistent volume* para o *container* do MySQL, garantindo a existência de mais espaço quando necessário.

Desta forma, podemos dizer que a base de dados está hospedada na própria Google Cloud Platform, plataforma onde se encontra implementada a solução, sendo a sua disponibilidade fornecida na respetiva porta, graças ao *MySQLService*.

2.2 Instalação e configuração automática da plataforma

2.2.1 Análise dos pontos críticos da plataforma

Como a aplicação utiliza serviços de armazenamento, rapidamente se percebe a existência de um ponto crítico ao seu funcionamento. O armazenamento é crucial, tanto para o desempenho como para a integridade dos dados guardados, no entanto, é uma componente complexa de se escalar horizontalmente, devido à necessidade de sincronização entre os servidores de suporte à componente em questão.

Relativamente ao *frontend*, não consideramos que este seja um ponto crítico, uma vez que não requer sincronização entre os servidores de suporte. Cada servidor consegue receber e responder a pedidos dos clientes, pelo que a complexidade da sua escalabilidade horizontal é reduzida.

2.2.2 Ferramentas e abordagem utilizadas

Vagrant

De modo a criar uma Virtual Machine, é utilizada a ferramenta **Vagrant**. Esta é uma ferramenta para construir e gerir ambientes de desenvolvimento virtuais. Permite que os desenvolvedores criem e configurem ambientes de desenvolvimento leves, reproduzíveis e portáteis, usando um único arquivo de configuração.

Com o Vagrant, é possível criar e gerir facilmente máquinas virtuais que são configuradas para um projeto em específico e podem ser facilmente partilhadas com outros membros da equipa. Isso permite que os programadores desenvolvam e testem o código de forma consistente no mesmo ambiente, independentemente do sistema operativo ou da configuração de *hardware* da máquina *host*. Além disso, o Vagrant pode ser integrado com várias ferramentas de provisionamento, como *shell scripts* ou Ansible, para instalar e configurar automaticamente o *software* nas máquinas virtuais.

Ansible e Kubernetes

Para garantir a disponibilidade da plataforma de forma automática e flexível, foi escolhido o **Ansible** como ferramenta de provisionamento, e assim desenvolvido um *playbook* de Ansible. Este serve para a gestão de configurações e é capaz de realizar o *provisioning* e *deployment* dos diferentes componentes facilmente, em conjunto com a *framework* **Kubernetes**, responsável pela gestão e comunicação das componentes instaladas.

Em relação ao funcionamento do deploy da aplicação desenvolvido segue-se aproximadamente a seguinte estrutura:

Primeiro é instalado e configurado num container uma base de dados do *mysql* junto com a têcnica de *persistent volume claim*, seguido da criação de um *service* para garantir a conexão futura a esta base de dados.

Antes de partir para o *deployment* da aplicação Ghost, é realizado o *provisioning* de um *service* disponibilizando um *ip externo* para utilização do cliente, onde é também implementado um *loadbalancer* de forma a máximizar a escalabilidade horizontal, e é definida uma porta para a qual serão redirecionados os pedidos feitos para determinado *container*, que no caso será a aplicação Ghost.

Após realizado o *provisioning*, é extraído o respetivo IP, e efetuado o *deployment* da própria aplicação Ghost, nesse respetivo IP e porta definida. Também é nesta fase realizada a conexão à base de dados através do *mysql-Service*. É importante notar que, tal como dito anteriormente, a base de dados recorre a *provision volumes* para garantir espaço de armazenamento.

Posteriormente, é extraído o id do *pod* responsável pela hospedagem da base de dados e alterado o respetivo utilizador/administrador e, por útlimo, ativada a escalabilidade horizontal, proporcionada pelo próprio *kubernetes*, garantindo assim a gestão da replicação de *containers* automaticamente, quando necessário. É importante notar que todo o processo de escalabilidade horizontal é realizado automatizadamente.

Para realizar o *undeploy* da aplicação Ghost, são simplesmente apagados todos os *pods*, serviços e *deployments* efetuados, recorrendo à respetiva funcionalidade oferecida pelo próprio kubernetes.

Arquitetura da solução

De modo sintetizado, uma das ferramentas utilizadas foi o **Vagrant** – para realizar o *provisioning* e *deployment* de uma máquina virtual local –, possibilitando assim a utilização da ferramenta **Ansible**, na realização do *provisioning* e *deployment* de todos os componentes da aplicação, recorrendo à ferramenta **Kubernetes**.

A arquitetura desenvolvida para fazer o *deployment* da plataforma Ghost inclui diversas componentes, e apresenta-se resumida na figura 2.2.

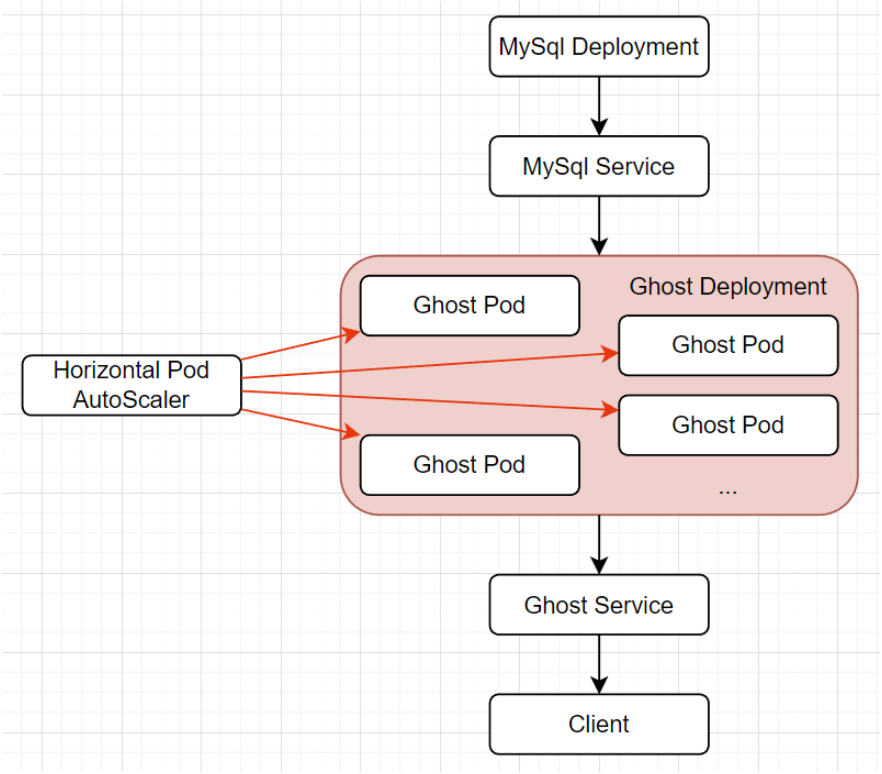

Figura 2.2: Arquitetura da solução.

O MySQL Deployment faz a implementação da base de dados de suporte à aplicação, sendo criado um *persistent volume* de 20Gb. De acordo com as variáveis definidas no Ansible, o MySQL é criado com um utilizador *default*, *password default* e base de dados também *default*, exposta por defeito na porta 3306.

O Ghost Deployment inclui *pods* com as instâncias da aplicação, tantas quanto as necessárias no momento, conforme os recursos requeridos pela necessidade dos utilizadores. O *Horizontal Pod AutoScaler* é utilizado para ajustar o número de *pods* de forma automática, para garantir a escalabilidade e possibilidade de existirem vários *pods* replicados.

Tanto o MySQL Service como o Ghost Service são processos necessários à correta implementação deste *software*.

De notar que entre as várias abordagens realizadas, encontra-se o uso de expressões regulares em comandos executados internamente aos *pods*, recorrendo ao *Kubectl exec*. Ainda, depois de se fazer o Ghost deployment, é utilizado código que garante que o *deployment* está em execução antes de criar o utilizador ou utilizar HTTP GET, existindo sincronização nesta fase.

Capítulo 3

Primeira tarefa

3.1 Monitorização

A monitorização, tal como planeado anteriormente, será realizada recorrendo às ferramentas disponibilizadas pela Google Cloud Platform, para observar as seguintes métricas:

- **Taxa de utilização do CPU** - Permite perceber se as máquinas têm capacidade de processamento suficiente para lidar com as operações necessárias, ou se necessitam de ser melhoradas. Também podemos avaliar a carga que a aplicação exerce no CPU.
- **Quantidade de memória necessária** - Permite aos utilizadores analisar a memória consumida pela aplicação, permitindo ao mesmo avaliar a necessidade de melhorias na infraestrutura.
- **Quantidade de operações de leitura/escrita do disco** - Permite inferir se a maioria dos pedidos depende de escritas ou leituras de disco.
- **Quantidade de *bytes* lidos/escritos do disco** - Através da quantidade de *bytes* lidos, juntamente com a métrica anterior, é possível calcular o tamanho médio das operações de escrita e leitura, possibilitando avaliar se o disco consegue lidar com os pedidos de escrita e leitura simultâneos.
- **Quantidade de *bytes* enviados/recebidos pela rede** - Recorrendo a esta métrica, rapidamente descobrimos se existe largura de banda suficiente para o processamento dos pedidos dos utilizadores.
- **Quantidade de pacotes enviados/recebidos pela rede** - Permite, juntamente da métrica anterior, analisar o tamanho médio de cada pacote que é transportado na rede.

Após realizar o *provisioning*, monitorizamos as taxas de utilização de recursos, com o sistema em *idle* (figuras 3.1 e 3.2).

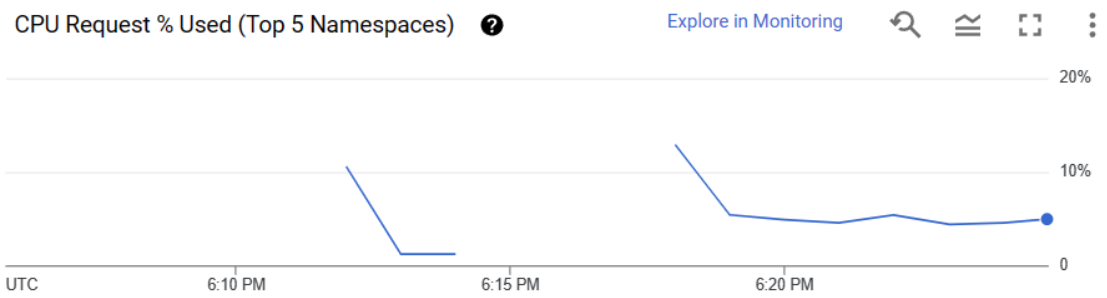

Figura 3.1: Utilização de CPU após o *provisioning*.

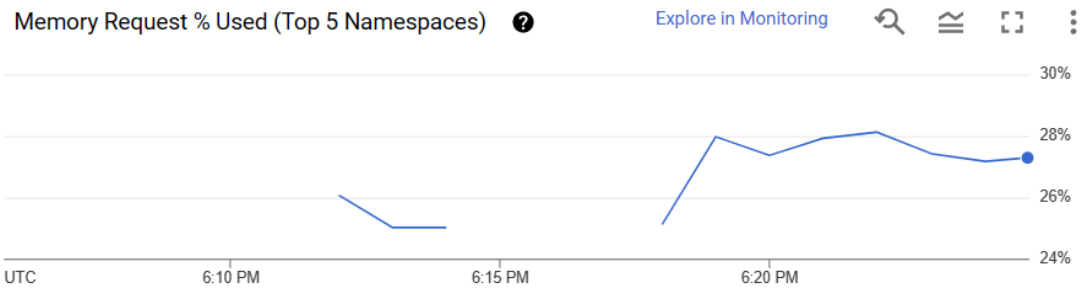

Figura 3.2: Utilização de memória após o *provisioning*.

Por fim, após realizar o *deployment*, monitorizamos de novo as taxas de utilização de recursos (figuras 3.1 e 3.2).

Figura 3.3: Utilização de CPU após o *deployment*.

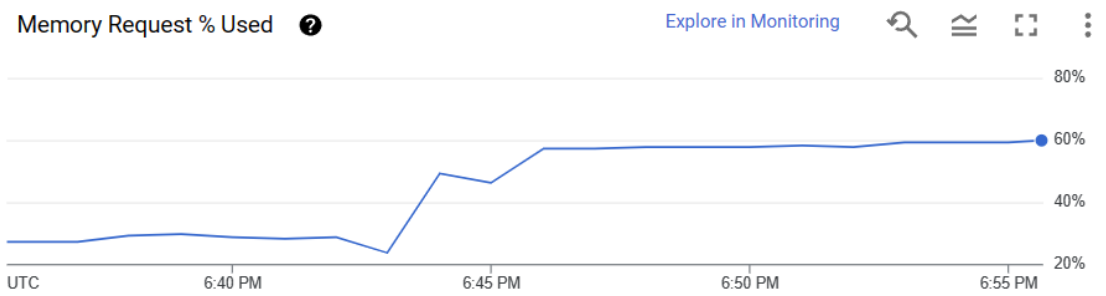

Figura 3.4: Utilização de memória após o *deployment*.

Podemos verificar que a taxa de utilização do CPU permaneceu inalterada após o *deployment*, enquanto a percentagem de utilização de memória aumentou cerca de 25%.

3.1.1 Dashboard

Foi também desenvolvida uma *dashboard* personalizada de forma a facilitar a análise do desempenho das diferentes componentes do *cluster*.

De notar que, de forma a todas as métricas estarem disponíveis, é necessária a instalação de *Agents* em cada nodo presente.

CPU e RAM

Nesta secção estão presentes as seguintes informações relativas à utilização dos CPU e da memória RAM:

- ***Cluster CPU Utilization***: Percentagem da média da utilização de todos os processadores do *cluster* no momento.
- ***Cluster Memory Utilization***: Percentagem de memória RAM a ser utilizada pelo *cluster* no momento.
- ***Cluster CPU Usage***: Percentagem da média da utilização de todos os processadores do *cluster* ao longo do tempo.
- ***Cluster Memory Usage***: Percentagem de memória RAM a ser utilizada pelo *cluster* ao longo do tempo.

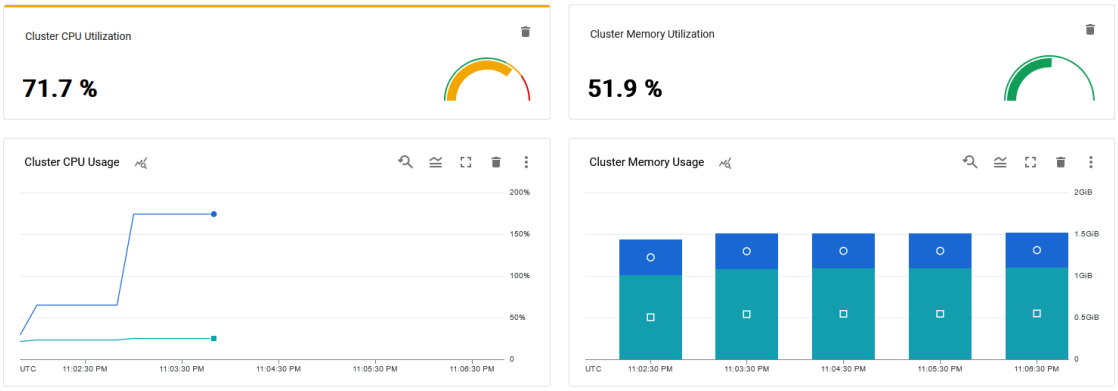

Figura 3.5: *Dashboard* de *CPU* e *RAM*.

Network

Nesta secção estão presentes as seguintes informações relativas à carga a nível de rede no *cluster* onde a aplicação está hospedada:

- ***Received Packets***: Número de pacotes por segundo que cada nodo recebeu ao longo do tempo.
- ***Sent Packets***: Número de pacotes por segundo que cada nodo enviou ao longo do tempo.
- ***Network Traffic***: Média do tráfego de rede do *cluster* ao longo do tempo
- ***RTT Latency per Node***: Latência do *Round-Trip-Time* de cada nodo do *cluster*
- ***RTT Latency per Pod***: Latência do *Round-Trip-Time* de cada *pod* do *cluster*

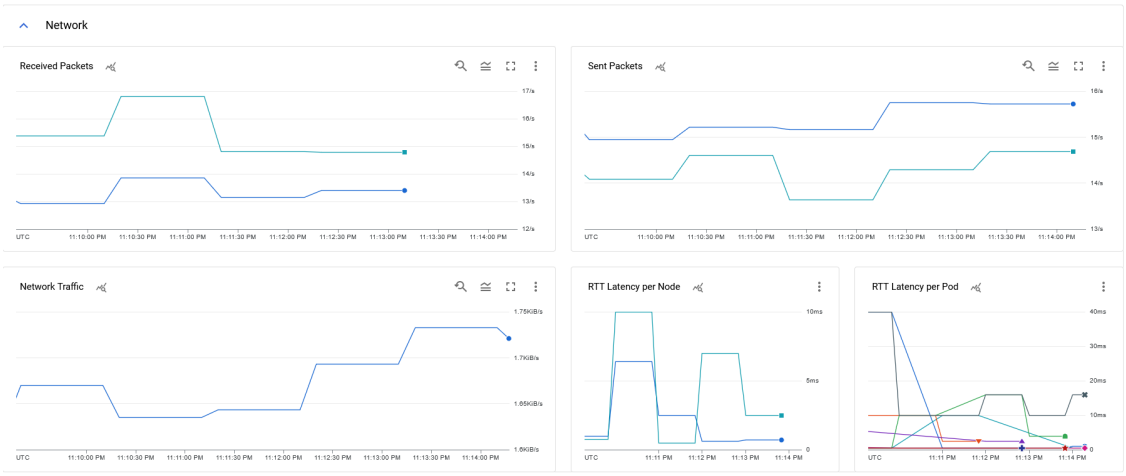

Figura 3.6: *Dashboard* de rede.

Storage

Nesta última secção encontra-se presente a informação referente ao armazenamento do *cluster*:

- ***Cluster Storage Usage***: Percentagem da utilização do armazenamento do *cluster*
- ***Disk Bytes Used***: Quantidade de armazenamento utilizado no *cluster*
- ***Disk Read Operations***: Número de operações de leitura no disco do *cluster*
- ***Disk Write Operations***: Número de operações de escrita no disco do *cluster*

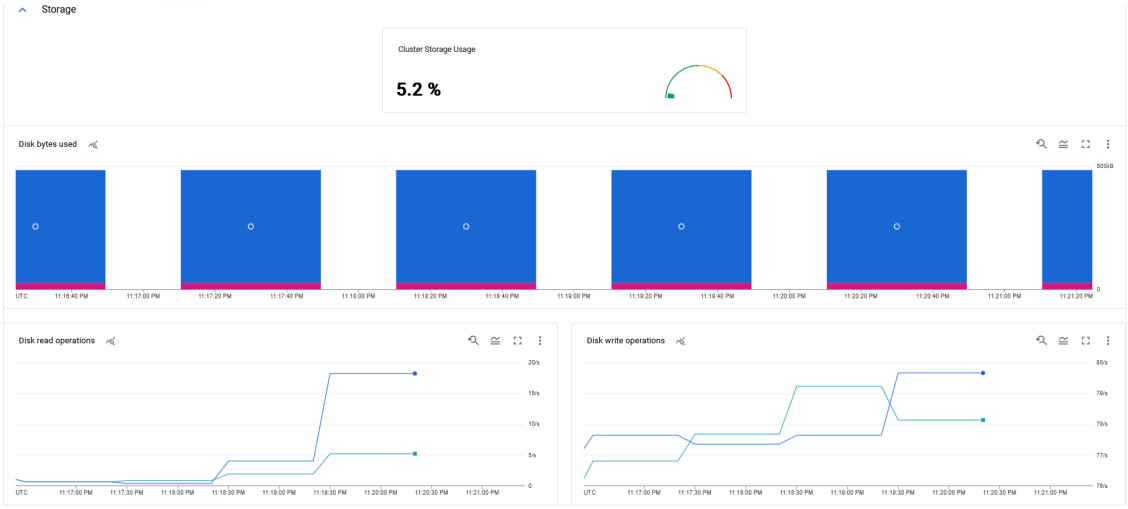

Figura 3.7: *Dashboard* de armazenamento.

Capítulo 4

Segunda tarefa

4.1 Avaliação Experimental

A realização da avaliação experimental baseou-se na utilização de testes automatizados realizados com recurso ao *software* JMeter. Para isso, foram efetuados os seguintes testes:

4.1.1 Teste 1

Neste teste é criado um caso sintético de acessos à página principal do ghost executada por 100 *threads* com 300ms de intervalo.

É executado um GET request para a página principal.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	200	1312	302	2346	757,97	11,50%	28,8/sec	302,51	2,93	10764,4
TOTAL	200	1312	302	2346	757,97	11,50%	28,8/sec	302,51	2,93	10764,4

Figura 4.1: Sumário do teste 1.

4.1.2 Teste 2

Neste teste é criado um caso sintético de login ao ghost-admin do Ghost executada por 10 *threads* com 1000ms de intervalo

É executado um POST request para a plataforma com as credenciais de um utilizador.

4.1.3 Monitorização sob carga

As taxas de utilização de recursos foram monitorizadas e encontram-se de seguida (figuras 4.2 e 4.3).

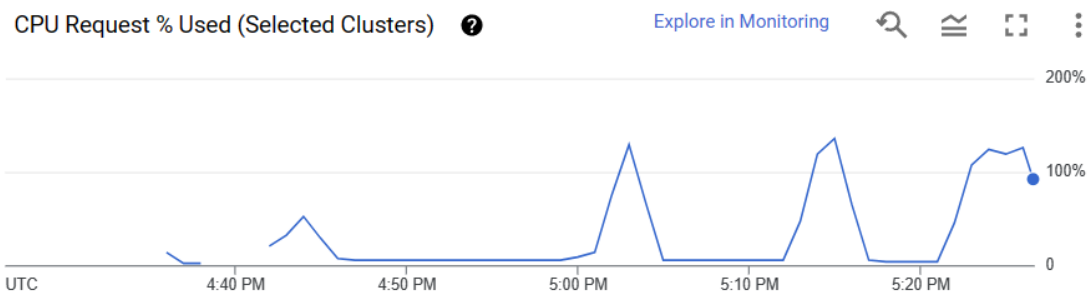

Figura 4.2: Utilização de CPU em testes de carga.

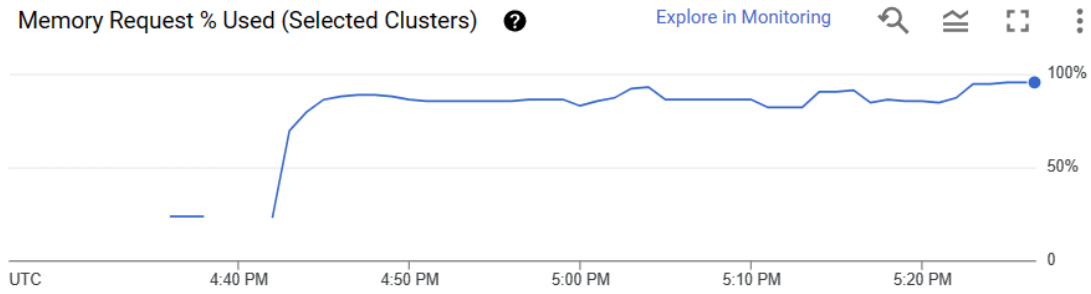

Figura 4.3: Utilização de memória em testes de carga.

Capítulo 5

Terceira tarefa

5.1 Escalabilidade e Resiliência

As Kubernetes são um sistema de orquestração de *containers* que consegue automatizar o *deployment*, dimensionamento e gestão de aplicações com possível grande número *containers*, automatizando também a gestão da escalabilidade dos mesmos. Tratam do armazenamento persistente, criando volumes de dados distintos para o MySQL, o que faz com que os dados não sejam perdidos na interrupção da execução da aplicação.

Considerando estas características, gerimos a escalabilidade dos *containers* utilizando o Google Kubernetes Engine (GKE).

Com a utilização do comando

kubectl autoscale deployment ghost-deployment –min 1 –max 10 –cpu-percent 65

é criado um *Horizontal Pod Autoscale* que automaticamente escala a carga de trabalho conforme as exigências no momento. Neste caso, quando o CPU atingir 65% da sua capacidade, é acrescentado 1 *pod*, até serem utilizados no máximo 10 *pods* em simultâneo.

5.1.1 Análise critica da abordagem

A utilização do *autoscale* de Kubernetes, para abordar a escalabilidade dos *containers* e a sua replicação, apresenta diversas vantagens na escalabilidade e resiliência de qualquer aplicação que o utilize. Mas quando se fala no Ghost, uma plataforma *open-source* de *blogs*, a escalabilidade é um fator importante a ser considerado. Pela sua natureza, esta aplicação pode experienciar períodos de alta demanda.

Considerando os recursos que são utilizados por toda a aplicação, o *autoscale* ajusta automaticamente o número de réplicas de um *deploy* baseando-se na utilização de recursos pelos *pods*. Isto garante que os recursos são utilizados de forma eficiente e que os *pods* não ficam sobrecarregados de informação, melhorando a utilização de recursos, o que pode garantir que o Ghost consiga lidar com altas cargas de tráfego.

Da mesma forma, isto permite que o custo de executar *clusters* com a plataforma Ghost seja reduzido, já que é evitado o uso desnecessário de recursos. Disto resulta a facilidade de gestão do número de réplicas de um *deploy* e a facilidade de resposta às mudanças nos requisitos dos recursos, o que pode ser benéfico para o *software* Ghost, que pode ter configurações e dependências complexas.

Para além disto, o *autoscale* garante que existe sempre capacidade suficiente para lidar com o tráfego de entrada, mesmo durante períodos em que há vários pedidos em simultâneo, o que é importante para uma plataforma de *blogs* como o Ghost, pois garante que os utilizadores possam aceder ao *site* e publicar conteúdo mesmo durante os períodos de pico de utilização. Numa situação como esta, garante ainda que o desempenho do *deploy* permanece consistente.

O *autoscale* no Kubernetes pode ser uma ferramenta útil para ajustar automaticamente o número de réplicas de um *deployment* com base no uso de recursos, mas pode vir com complexidade adicional e exigir alguma configuração específica para funcionar eficazmente com o Ghost. Para garantir que funciona corretamente e atende às necessidades da aplicação, é necessário proceder a testes e à monitorização do seu desempenho.

No entanto, o Google Kubernetes Engine encontra-se de momento com um erro temporário [1], impossibilitando a utilização do *autoscale* e, consequentemente, impossibilitando colocar no presente relatório os testes e monitorização efetuada ao desempenho do mesmo.

[1] https://status.cloud.google.com/incidents/cRkfNr6MLK8aP2hZsusU

Capítulo 6

Conclusões

O objetivo deste projeto passa pela caracterização, análise, instalação, monitorização e avaliação da aplicação Ghost utilizando os conceitos abordados ao longo do semestre nas aulas da Unidade Curricular, possibilitando a consolidação de conhecimentos relativamente às tarefas de gestão e automatização do processo de instalação, monitorização e avaliação da aplicação.

Este trabalho representa uma fase fulcral no desenvolvimento de uma aplicação, dada a necessidade desta, em ser capaz de se encarregar por todas as exigências suscetíveis de surgirem, sendo ainda mais importante quando o objetivo é planear uma tipologia de serviço/aplicação que necessite de alta escalabilidade e resiliência.

Concluímos que com a realização deste trabalho foi possível obter conhecimentos valiosos, uma vez que foram utilizadas ferramentas muito úteis, relativamente simples e intuitivas de se usarem, assim como obtida experiência na utilização prática do Google Kubernetes Engine fornecido pela Google Cloud Platform.