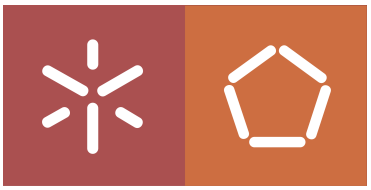

Universidade do Minho
Escola de Engenharia

Aplicações e Serviços de Computação em Nuvem

Relatório Trabalho Prático

Grupo **19**

MEI - 1º Ano - 1º Semestre

Trabalho realizado por:

PG55945 - Gonçalo Marinho
PG55947 - Henrique Vaz
PG57886 - Lucas Oliveira
PG55987 - Mike Pinto
PG56000 - Rafael Gomes

Braga, 5 de janeiro de 2025

Índice

Lista de Figuras

1. Introdução

A aplicação Moonshot desempenha um papel crucial na gestão de Certificados Verdes Digitais, servindo como um servidor aplicacional que interage com uma aplicação mobile através de uma API. Este trabalho prático tem como objetivo explorar e otimizar os desafios associados à instalação, configuração e monitorização desta aplicação no Google Kubernetes Engine (GKE). Com base numa configuração inicial básica, avançamos para uma abordagem mais robusta, capaz de lidar com os desafios de desempenho, escalabilidade e resiliência, garantindo a continuidade do serviço mesmo sob condições adversas.

Neste contexto, destacamos a utilização de ferramentas como o Ansible para automação de processos, o JMeter para testes de carga que simulam atividades reais de utilizadores, e as ferramentas nativas do Google Cloud para monitorização, como dashboards personalizados baseados em queries de CPU e memória. Esta abordagem permite não apenas avaliar o comportamento da aplicação sob carga, mas também identificar e mitigar pontos de falha críticos.

2. Questões

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

1. Servidor Aplicacional:

   À medida que o número de clientes aumenta, o servidor aplicacional pode enfrentar dificuldades em processar múltiplos pedidos simultaneamente. O excesso de carga pode resultar em tempos de resposta mais longos, maior latência e, eventualmente, falhas no atendimento de novos pedidos.

2. Base de Dados:

   A base de dados pode tornar-se outro ponto crítico, especialmente em operações de escrita, que frequentemente requerem garantias adicionais de consistência. Um número elevado de queries simultâneas pode sobrecarregar o sistema, reduzindo o desempenho e aumentando os tempos de resposta.

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

Na Tabela 1 mostramos o desempenho da aplicação perante diferentes números de clientes simultâneos a aceder à página inicial da aplicação, num intervalo de 300ms em 300ms. (Como o nosso objetivo seria replicar a aplicação web, os clientes apenas fazem um pedido de GET da página de login de administrador).

Tabela 1: Desempenho da aplicação com diferentes threads(users)

Num. Users	5	50	100	200	500
Mean Response Time	0.137s	0.177s	0.5s	3.24s	8.67s
Min Response Time	0.128s	0.126s	0.127s	0.128s	0.128s
Max Response Time	0.366s	0.487s	0.508s	9s	21.747s

Os gráficos abaixo demonstram a carga no sistema com 200 e 300 clientes em simultâneo. Para 200 clientes o sistema está a usar cerca de 70% do CPU disponível. Para 300 clientes já existem picos no consumo de CPU que indicam que o sistema não têm capacidade suficiente para lidar com tantos pedidos na configuração atual (sem replicação).

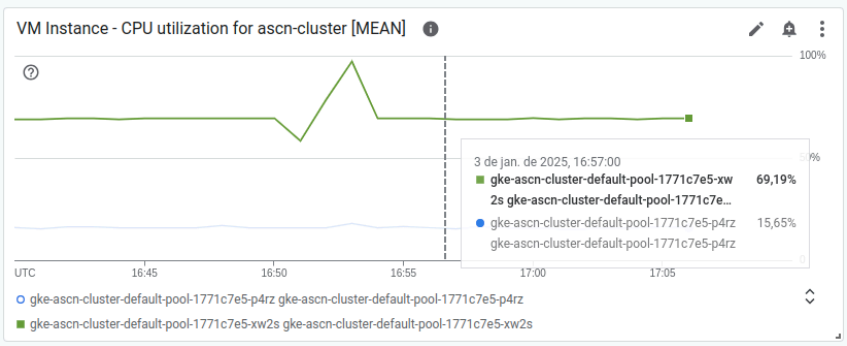

Figura 1: Utilização de CPU para 200 clientes (threads)

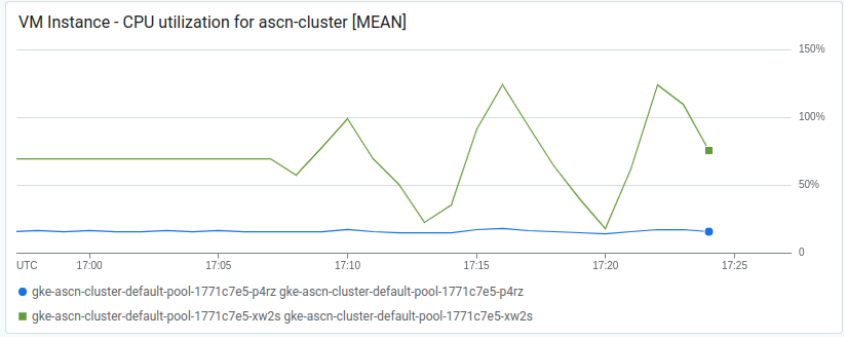

Figura 2: Utilização de CPU para 300 clientes (threads)

Foi também testado o desempenho da aplicação perante um cenário mais perto do real em que são feitos pedidos de login e obtidos tokens de acesso. O intervalo dos pedidos é o mesmo que no cenário anterior.

Tabela 2: Desempenho da aplicação com diferentes threads(users)

Num. Users	**5**	**50**	**100**	**200**	**500**
Mean Response Time	6.52s	65.439s	-	-	-
Min Response Time	2.179s	3.236s	-	-	-
Max Response Time	9.734s	124.73s	-	-	-

A partir dos 50 utilizadores em simultâneo não foram efetuadas as medições pois os valores já estavam demasiado elevados.

Foram também feitos testes com cargas de trabalho mais complexas (inserção e consulta de certificados) mas os seus resultados não serão apresentados por uma questão de simplificação.

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

1. Servidor Aplicacional:

   Se houver apenas uma instância do servidor aplicacional, a sua falha resultará na incapacidade da aplicação processar pedidos dos clientes. Sem redundância ou um sistema de balanceamento de carga, o servidor torna-se um ponto crítico de falha.

2. Base de Dados:

   A base de dados é outro componente essencial. Uma falha na base de dados, seja por sobrecarga, corrupção de dados ou indisponibilidade, impede o servidor aplicacional de aceder aos dados necessários para atender aos pedidos dos utilizadores. Este problema é agravado em configurações sem replicação ou sistemas de failover.

Questão 4
Que otimizações de distribuição/replicação de carga podem ser aplicadas à instalação base?

Uma das principais estratégias para otimizar a instalação base é o escalonamento horizontal do servidor aplicacional. Esta abordagem permite aumentar dinamicamente o número de pods em execução,

ajustando-se automaticamente à carga crescente da aplicação. Com a implementação de um Horizontal Pod Autoscaler (HPA), o sistema monitoriza métricas e aumenta ou reduz o número de pods conforme necessário, distribuindo os pedidos uniformemente entre os pods ativos, evitando sobrecargas em instâncias específicas e melhorando o tempo de resposta para os utilizadores.

Outra área crucial de otimização é a replicação da base de dados, uma vez que este componente frequentemente se torna um *bottleneck*. Configurar réplicas da base de dados permite distribuir as consultas de leitura por várias instâncias, aliviando a carga na base de dados primária e melhorando o desempenho global. Para lidar com operações de escrita em grande escala, pode ser considerado o uso de sharding, que divide os dados em múltiplas instâncias, permitindo que cada uma lide com uma parte específica do conjunto total de dados. Estas estratégias de replicação e distribuição são fundamentais para suportar o crescimento da aplicação e melhorar a sua resiliência a falhas.

Questão 5
Qual o impacto das otimizações propostas no desempenho e/ou resiliência da aplicação?

No caso do escalonamento horizontal do servidor aplicacional, o desempenho é melhorado devido à capacidade de aumentar dinamicamente o número de pods, o que permite que a aplicação responda de forma eficiente a picos de tráfego sem degradação perceptível da qualidade do serviço. A distribuição uniforme de pedidos por meio de um balanceador de carga evita que instâncias específicas sejam sobrecarregadas, garantindo tempos de resposta mais rápidos e consistentes.

A replicação da base de dados também contribui significativamente para o desempenho ao distribuir as consultas de leitura por várias instâncias, o que reduz a latência e aumenta a capacidade de lidar com volumes elevados de pedidos. O uso de sharding melhora a eficiência das operações de escrita, uma vez que cada instância da base de dados processa apenas uma fração do total de dados.

3. Arquitetura

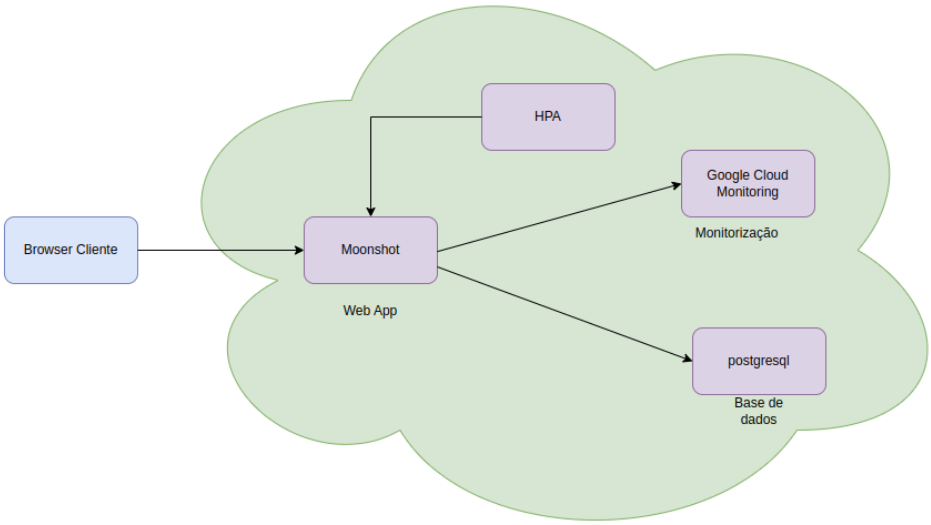

Figura 3: Arquitetura da aplicação

- **Camada de Interface:** A aplicação está acessível através de um navegador, utilizando o endereço IP disponibilizado pelo GKE.
- **Camada de Aplicação:** A aplicação está hospedada no GKE, com seu container configurado para escalar horizontalmente até um máximo de 10 réplicas sempre que o uso de CPU ultrapassar 75%.
- **Camada de Dados:** A base de dados está contida num único container hospedado no GKE.
- **Monitorização e Avaliação:** A monitorização foi implementada utilizando a API do Google Cloud Monitoring, enquanto o benchmarking foi realizado com a ferramenta JMeter.

4. Instalação e Configuração Automática da Aplicação

Para a implementação do projeto, utilizámos o Google Kubernetes Engine (GKE) da Google Cloud, em conjunto com o Ansible, uma plataforma de automação para configuração e gestão de sistemas.

Além de automatizar a configuração do sistema, o Ansible também oferece uma camada adicional de segurança por meio do Ansible Vault. Esta ferramenta permite encriptar e proteger informações sensíveis, como credenciais de utilizadores e passwords. No nosso projeto, utilizámos o Ansible Vault para garantir a segurança dos dados da conta de teste usada nas tarefas de configuração do sistema, restringindo o acesso às informações apenas a quem possui a password correta (ascn2425).

4.1. Criação do Cluster Kubernetes

Para efetuar a criação do cluster basta correr o comando abaixo, será criado um cluster com 2 nodos.

```
$ ansible-playbook -i inventory/gcp.yml gke-cluster-create.yml
```

4.2. Executar Aplicafção

Correndo o comando abaixo é arrancada tanto a aplicação Moonshot como a respetiva base de dados. É também criado o HPA para permitir a replicação do Moonshot.

```
$ ansible-playbook -i inventory/gcp.yml moonshot-deploy.yml --ask-vault-pass
```

4.3. Executar Aplicação sem HPA (Horizontal Pod Autoscaler)

Caso seja necessário correr o sistema sem que exista replicação de pods é possivel usando o comando abaixo.

```
$ ansible-playbook -i inventory/gcp.yml moonshot-deploy.yml -e skip_hpa=true --ask-vault-pass
```

4.4. Criação do Dashboard no Google Cloud

É possível a criação automática do *dashboard* usando o comando abaixo,

```
$ ansible-playbook -i inventory/gcp.yml createDashboard.yml --ask-vault-pass
```

4.5. Instalação do Jmeter

Para instalar o Jmeter e todas as suas dependências usa-se o comando abaixo.

```
$ *ansible-playbook -i inventory/gcp.yml installJmeter.yml --ask-vault-pass
```

4.6. Executar os Testes Automaticamente

Para correr os testes de forma automática foi criado um *script bash* cuja forma de uso está descrita abaixo.

```
$ *./runJmeter.sh -i \<LoadBalancerIP> -t \<NrThreads>
```

5. Análise da Escalabilidade

Sendo a arquitetura composta por a uma única instância de servidor aplicacional e uma base de dados, surgirão desafios consideráveis à medida que a utilização da aplicação aumenta. Embora este modelo básico seja suficiente para cenários de baixa carga, ele torna-se rapidamente insuficiente para suportar um crescimento sustentado no número de utilizadores ou na complexidade das operações realizadas.

Um dos problemas mais evidentes é a incapacidade de escalabilidade desta configuração. O aumento do número de clientes leva inevitavelmente a uma sobrecarga do servidor aplicacional, que, por sua vez, compromete a capacidade de processar múltiplos pedidos de forma eficiente. Este congestionamento traduz-se em tempos de resposta mais longos e, em casos extremos, na falha no processamento de alguns pedidos, o que é notório em momentos de pico de utilização, onde a carga simultânea excede a capacidade do servidor, resultando numa degradação generalizada da experiência do utilizador.

Por outro lado, a base de dados também enfrenta limitações críticas. À medida que o número de queries, especialmente as de escrita, aumenta, a performance global é impactada. Estas operações, muitas vezes mais exigentes devido à necessidade de garantir a consistência dos dados, tornam-se um *bottleneck*. Esta limitação não apenas afeta diretamente os tempos de resposta, mas também a estabilidade da aplicação como um todo, uma vez que uma base de dados sobrecarregada reduz a capacidade do sistema de suportar transações simultâneas de forma confiável.

Outro ponto preocupante desta arquitetura inicial é a falta de redundância. Sem instâncias adicionais que possam assumir as responsabilidades do servidor aplicacional ou da base de dados em caso de falha, a aplicação torna-se extremamente vulnerável. Um simples erro no servidor aplicacional pode interromper completamente o processamento de pedidos. Da mesma forma, qualquer problema na base de dados, seja por sobrecarga ou falha técnica, inviabiliza o acesso a dados essenciais, comprometendo a aplicação na sua totalidade.

Estas limitações sublinham a importância de evoluir para uma solução mais robusta e escalável. A adoção de estratégias como o balanceamento de carga para distribuir pedidos entre múltiplos servidores, a replicação da base de dados para garantir disponibilidade e consistência, e o uso de mecanismos de escalonamento horizontal para ajustar dinamicamente a capacidade da infraestrutura são passos fundamentais para superar os gargalos identificados. Este tipo de evolução arquitetural não apenas resolve os problemas de desempenho, mas também aumenta a resiliência do sistema, reduzindo os riscos de interrupção e garantindo uma experiência mais estável e confiável para os utilizadores.

6. Avaliação do Desempenho

6.1. Ferramentas Utilizadas

Para a validação e monitorização da aplicação, utilizámos uma combinação de ferramentas que nos permitiram testar a sua performance e garantir uma gestão eficiente dos recursos durante o funcionamento.

Recorremos ao JMeter para simular a atividade de utilizadores reais na aplicação, com foco em operações comuns, como o processo de login e a inserção de dados. Estas simulações permitiram-nos reproduzir cenários de utilização realista, avaliar o comportamento da aplicação sob diferentes níveis de carga e identificar possíveis pontos de estrangulamento. Com o JMeter, configurámos múltiplos threads para simular utilizadores simultâneos, medindo tempos de resposta e taxa de sucesso para as operações simuladas.

Para a monitorização, utilizámos as ferramentas integradas do Google Cloud para observar e analisar o desempenho da aplicação em tempo real. Criámos dashboards personalizados no painel de monitorização, onde configurámos queries específicas para monitorizar métricas críticas, como o uso de CPU e memória dos pods. Estes dashboards facilitaram a visualização do impacto das cargas simuladas e ajudaram-nos a validar a configuração do Horizontal Pod Autoscaler (HPA), verificando como o sistema respondia ao aumento de tráfego.

Adicionalmente, utilizámos logs e métricas fornecidos pela infraestrutura do Kubernetes no Google Cloud para aprofundar a análise de desempenho. Com estas ferramentas, conseguimos identificar padrões de utilização de recursos e confirmar a eficácia das estratégias de escalonamento configuradas, ajustando-as quando necessário para alcançar um desempenho otimizado. Este conjunto de ferramentas foi essencial para garantir a estabilidade e escalabilidade da aplicação ao longo do desenvolvimento e testes.

6.2. Monitorização

Relativamente à monitorização, desenvolvemos uma dashboard que apresenta os dados mais relevantes sobre a infraestrutura da aplicação. Esta dashboard inclui gráficos que mostram, por exemplo, o número de réplicas do servidor aplicacional e a percentagem de CPU utilizada pelos pods desse servidor, como é possível observar na Figura 4.

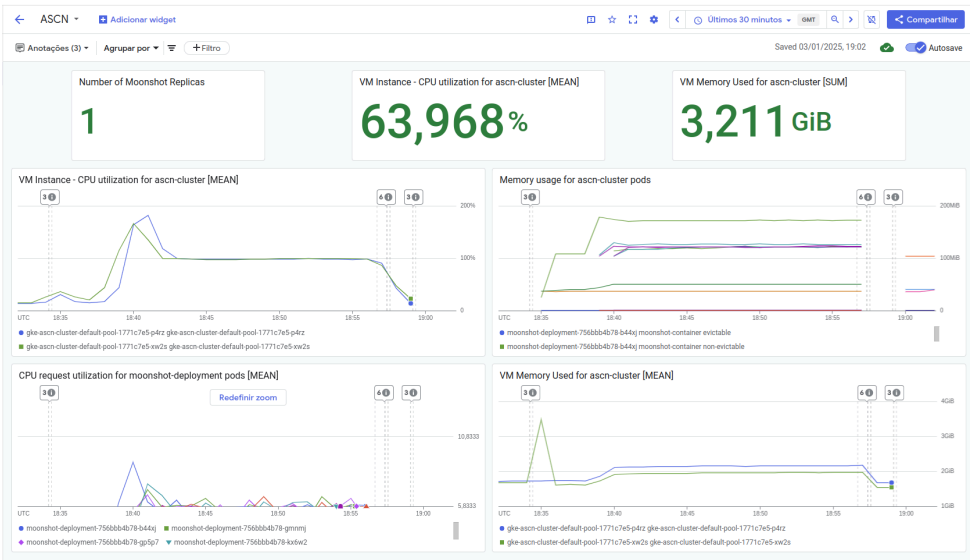

Figura 4: Dashboard de monitorização

Como referido anteriormente este dashboard pode ser criado automaticamente através de um playbook Ansible.

6.3. Escalonamento Automático

Quanto ao escalonamento, optámos por implementar um Horizontal Pod Autoscaler (HPA) para a aplicação web devido à sua flexibilidade e capacidade de ajustar o número de pods com base nas variações de carga. A escolha de escalar a web app, em vez da base de dados, foi motivada pela maior complexidade envolvida na escalabilidade horizontal de uma base de dados, incluindo a necessidade de sincronização entre réplicas e possíveis problemas de consistência. Por outro lado, a web app apresenta um modelo mais simples de escalonamento, focado em lidar com aumento de tráfego sem alterações significativas na arquitetura.

Quanto à configuração do HPA, inicialmente considerámos a memória e a CPU como métricas de escalonamento. Definimos os limiares de utilização de CPU em 75%, não porque representa o ponto máximo de desempenho dos pods, mas sim porque foram considerados valores seguros. Este limiar permite que o sistema tenha tempo suficiente para inicializar novos pods antes que os existentes atinjam um estado crítico de saturação. Desta forma, conseguimos prevenir atrasos ou falhas no processamento de pedidos enquanto mantemos um equilíbrio entre desempenho e custos operacionais, evitando escalonamentos desnecessários ou tardios.

Durante a implementação, enfrentámos desafios relacionados com a integração das métricas no ambiente Google Cloud. Apesar de seguir a documentação oficial, constatámos que algumas métricas descritas não eram reconhecidas pelo Metrics Server. Esta limitação restringiu a exploração das métricas que inicialmente planeávamos utilizar, obrigando-nos a adaptar as configurações para trabalhar com métricas suportadas, como a utilização de CPU, e a ajustar as nossas expectativas em relação à monitorização.

6.4. Análise de Desempenho

Na Tabela 3 são apresentados os resultados do desempenho da aplicação, medido com um número crescente de réplicas. Os valores foram obtidos simulando 300 utilizadores (threads) simultâneos a enviar pedidos de GET da página de login de administrador, com um intervalo de 300 ms, ao longo de 50 iterações.

Tabela 3: Desempenho com aumento do número de réplicas do servidor aplicacional

Num. Replicas	**1**	**2**	**3**	**4**	**5**
Mean Response Time	4.18s	1.579s	0.727s	0.540s	0.510s
Min Response Time	0.130s	0.126s	0.126s	0.127s	0.127s
Max Response Time	13.288s	18.048s	4.096s	4.923	4.951
Throughput(req/s)	62	182	238	258	258

Na Figura 5 a criação de 4 replicas da aplicação, quando a carga sobre os pods começa a subir são criadas novas réplicas para que a carga de trabalho possa ser distribuída e seja mantido um tempo de resposta aceitável.

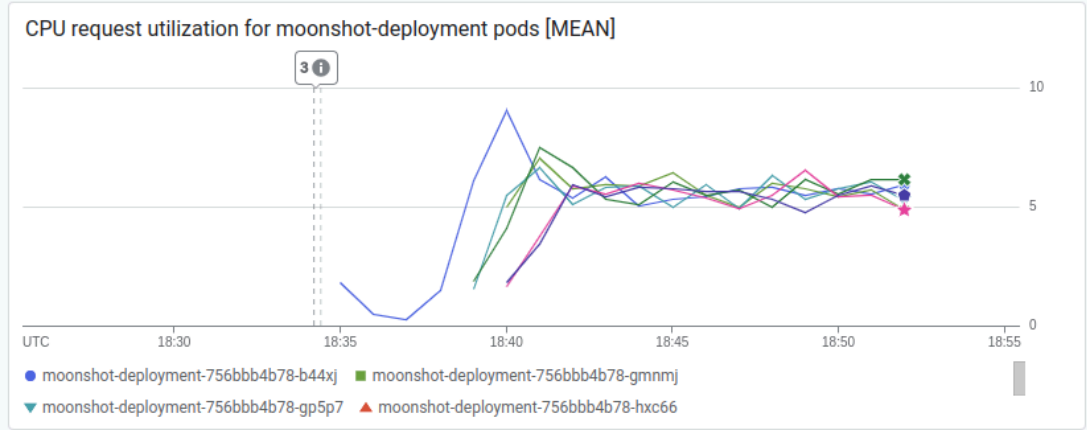

Figura 5: Pedidos de CPU por pod

Na nossa dashboard é também possível visualizar o número de réplicas existentes tal como é visível na Figura 6.

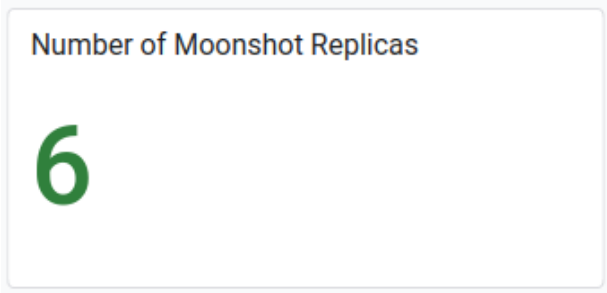

Figura 6: Número de réplicas

7. Conclusão

Ao longo deste trabalho, tivemos a oportunidade de aprofundar os nossos conhecimentos sobre conceitos essenciais de computação em nuvem, como escalabilidade, resiliência e automação. A implementação e otimização da aplicação Moonshot permitiram-nos compreender melhor os desafios práticos de gerir uma aplicação distribuída num ambiente como o Google Kubernetes Engine (GKE), desde a instalação automatizada até à monitorização e avaliação do desempenho.

Uma das principais aprendizagens foi a importância de escolher cuidadosamente as métricas de monitorização e os limiares de escalonamento. Ajustar o Horizontal Pod Autoscaler (HPA) para considerar apenas a CPU foi uma decisão fundamentada por testes que revelaram a irrelevância da memória como indicador de carga na nossa aplicação. Além disso, o uso do JMeter para simular interações reais de utilizadores ensinou-nos como avaliar o impacto da carga no comportamento da aplicação e identificar potenciais gargalos.

Entre as dificuldades enfrentadas, destacamos problemas relacionados com a integração das métricas de monitorização no ambiente do Google Cloud. Algumas métricas descritas na documentação não eram reconhecidas pelo Metrics Server, limitando as possibilidades de análise que inicialmente ambicionávamos. Esta limitação obrigou-nos a adaptar as configurações e explorar alternativas dentro do escopo permitido pelas ferramentas utilizadas.

No entanto, reconhecemos que há espaço para melhorias e exploração futura. Poderíamos aprofundar a análise de métricas adicionais, como I/O de disco, para obter uma visão mais completa do comportamento da aplicação. A introdução de estratégias de escalonamento baseadas em eventos, como o uso do Kubernetes Event-driven Autoscaler (KEDA), também seria uma área interessante a explorar para otimizar ainda mais o desempenho.

Por fim, este trabalho destacou a importância de uma arquitetura bem projetada e da automação para simplificar tarefas complexas e minimizar erros. Apesar dos desafios, conseguimos implementar uma solução funcional e escalável, e estamos confiantes de que as aprendizagens adquiridas servirão como uma base sólida para projetos futuros.