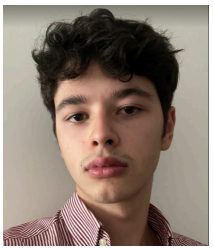
Universidade do Minho
Mestrado em Engenharia Informática

Trabalho Prático

Aplicações e Serviços de Computação em Nuvem

Ano Letivo de 2024/2025

José Correia pg55967



Pedro Lopes pg55992

Gonçalo Costa pg55944

Rodrigo Novo pg56006

Marta Rodrigues pg55982

ASCN

Índice

1. Introdução

Este trabalho foi desenvolvido no âmbito da unidade curricular de Aplicações e Serviços de Computação em Nuvem com o objetivo de automatizar a instalação, configuração, monitorização e avaliação da aplicação **Moonshot**, com base no conhecimento adquirido nas aulas práticas e teóricas.

Os objetivos foram atingidos com recurso a ferramentas como **Ansible**, de forma a automatizar totalmente o processo de instalação, **Kubernetes** através do serviço Google Kubernetes Engine, para uma gestão autónoma dos *containers* da aplicação e escalabilidade automática da mesma, **Google Cloud Platform** para monitorização e, por fim, **JMeter** com o objetivo de testar a escalabilidade do sistema (*benchmarking*).

Durante este relatório serão explicadas e justificadas todas as decisões tomadas e os motivos das mesmas, assim como as metodologias de *auto scaling* e balanceamento de carga.

2. Instalação e configuração automática

A aplicação foi implementada utilizando o Google Kubernetes Engine (GKE) do Google Cloud e recorrendo à utilização do **Ansible**, que desempenha um papel essencial para automatizar o aprovisionamento, a administração do ambiente Kubernetes, as ferramentas de monitorização e *benchmarking*, entre outros componentes adicionais.

Além disso, o **Ansible** possui uma ferramenta que adiciona uma camada de segurança, o **Ansible Vault** que permite encriptar e proteger informação sensível que o desenvolvedor ache pertinente, como credenciais de uma base de dados, por exemplo.

Com recurso ao **GKE**, foi possível gerir o ciclo de vida dos *containers* e dos seus componentes.

Após a inicialização do **Cluster GKE** é, portanto, possível executar a aplicação **Moonshot** e respetivos componentes em qualquer ambiente a partir do comando `ansible-playbook -i inventory/gcp.yml moonshot-deploy.yml`. Para encerrar a aplicação é possível utilizar o comando `ansible-playbook -i inventory/gcp.yml moonshot-deploy.yml`.

3. Arquitetura da aplicação

O **Moonshot** é uma aplicação responsável por gerir a emissão, validação e armazenamento de **DGCs** (Digital Green Certificates), usados para registar vacinas, testes e estado de recuperação, permitindo que utilizadores autorizados, como profissionais de saúde, criem certificados e possam gerir os mesmos.

A aplicação foi desenvolvida em **Python** e utiliza uma base de dados para armazenar informações relevantes, sendo essa **PostgreSQL**.

Uma vez que a aplicação será hospedada utilizando GKE, foi necessária a construção de imagens **Docker** tanto da aplicação como da sua base de dados. Estas imagens encontram-se disponíveis no **Docker Hub**, permitindo a criação dos containers que executam a aplicação e que serão geridos pelo ambiente Kubernetes. Esse mesmo GKE será responsável por disponibilizar um IP, através de um *Service* do tipo ***Load Balancer***, possibilitando o acesso externo à aplicação.

3.1. Análise da arquitetura inicial

Para a primeira fase do projeto (Tarefa 1), existe apenas uma única instância da aplicação **Moonshot** e da base de dados **PostgreSQL**, cada uma com um *pod* dedicado, proporcionando assim uma base sólida para futuras expansões.

Porém, esta configuração possui diversas fragilidades, tanto a nível de robustez como a nível de escalabilidade, onde o aumento de clientes leva ao declínio da qualidade de serviço.

Quando possuímos um número crescente de clientes e diferentes cargas de trabalho, esta configuração possui um gargalo de desempenho sem capacidade de escalabilidade.

Inicialmente, neste cenário, a qualidade de serviço do **servidor aplicacional** vai diminuindo quase exponencialmente, porque o servidor não consegue resistir à pressão imposta por vários pedidos em simultâneo, uma vez que não tem forma de lidar com este problema.

Além disso, a base de dados também pode afetar a eficiência do sistema quando esta fica sobrecarregada com demasiadas *queries* acumuladas, pois cada uma delas necessita da devida atenção para que os dados permaneçam coerentes.

Para além da escalabilidade, a falta de redundância para cada componente é uma vulnerabilidade que requer atenção, pois cria dois pontos únicos de falha. Se ocorrer uma falha tanto no **servidor aplicacional** como na **base de dados**, toda a aplicação fica comprometida. No caso de servidor falhar, a aplicação não consegue responder a pedidos. No caso da base de dados falhar, o servidor não será capaz de aceder aos recursos necessários para fornecer uma resposta.

Na seguinte tabela mostramos o desempenho da aplicação perante diferentes números de clientes simultâneos a aceder à pagina inicial da aplicação, num intervalo de 300ms em 300ms:

Users	10	50	100
Média tempo resposta (s)	0.434	8.346	21.262
Min (s)	0.417	1.021	1.134
Max (s)	0.450	10.874	23.347

Tabela 1: Desempenho da aplicação para diferentes quantidades de utilizadores

Podemos observar o tempo de respostas aos pedidos dos utilizadores vai piorando e aumentando gradualmente com número de clientes, e se considerarmos que mais de **5 segundos** é um tempo ineficiente, logo aos **50 utilizadores** a aplicação demonstra-se incapaz de lidar com a carga de trabalho.

De forma a resolver estes problemas distribuição de carga, uma possível solução é replicar o **servidor aplicacional**. Desta forma a aplicação irá possuir capacidade computacional suficiente para responder aos pedidos mais rápida e eficientemente, resolvendo e acabando com os gargalos de desempenho.

3.2. Análise da Arquitetura final

Após a anterior análise, temos agora como objetivo otimizar o desempenho, escalabilidade e resiliência da aplicação. Como referido anteriormente, a nossa aplicação está hospedada num ambiente ***Kubernetes***, que exige a especificação dos vários componentes que cooperam entre si para alcançar o funcionamento desejado para a aplicação.

Começando pelo **Moonshot**, sendo este um servidor aplicacional sem dados persistentes, a replicação é simples e melhora significativamente o desempenho da aplicação. Assim sendo, decidimos recorrer a um **HorizontalPodAutoscaler** (HPA) que aumenta ou diminui automaticamente o número de réplicas do *Pod* consoante a utilização de CPU. Este componente permite que a aplicação lide eficientemente com picos de carga e também ajuda para um uso mais eficiente de recursos computacionais.

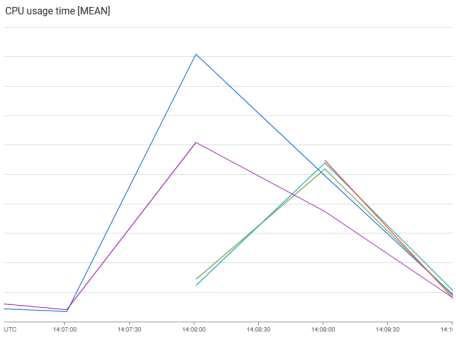


Figura 6: Exemplo de comportamento do HPA

Além disso, optamos por utilizar um *LoadBalancer* em vez de um *NodePort*, devido ao facto do *NodePort* expor o serviço diretamente numa porta fixa de cada nó, o que exigiria configuração manual para redirecionar o tráfego para os nodos. Isso pode resultar numa distribuição ineficiente de carga e numa limitação na escalabilidade. O *LoadBalancer*, por outro lado, distribui automaticamente os pedidos entre as réplicas, garantindo maior eficiência e resiliência. Isto foi possível devido à Google Cloud oferecer integração direta deste serviço.

Já na base de dados, para garantir a **persistência de dados da aplicação** independentemente do ciclo de vida do *Pod*, utilizamos um *PVC* (Persistent Volume Claim) que nos permite requisitar armazenamento persistente no *cluster* automaticamente, isto é, sem precisarmos de criar e gerir instâncias de *Persistent Volumes*.

Uma vez que aplicar técnicas de replicação à base de dados é muito complexo (comparativamente a replicar o servidor aplicacional, que não possui armazenamento persistente) e estas não possuem melhorias significativas de desempenho, optamos por não replicar a base de dados. Porém, sabemos que isto seria possível utilizando mecanismos como um *StatefulSet* que garante a coerência da informação entre as diversas réplicas, mas exigindo um maior rigor e coordenação na replicação do componente.

4. Avaliação de desempenho

Assegurar que uma aplicação funciona de forma estável e eficiente é fundamental para garantir uma boa experiência aos utilizadores, tornando indispensável o uso de ferramentas apropriadas.

4.1. Ferramentas de monitorização

De forma a monitorizar o estado da aplicação e o desempenho da máquina onde a nossa aplicação está hospedada optamos por utilizar DashBoards personalizáveis do Google Cloud Monitoring. Esta monitorização foca-se mais em observar métricas como o CPU e memória a ser utilizados, que nos permite ter uma melhor noção do estado atual da aplicação e possíveis gargalos ou pontos de falha.

Para além de observar os recursos gerais consumidos pela VM, também dividimos a atenção para observar individualmente os containers do moonshot e da base de dados. E prestamos atenção ao sistema de *logging* ("*Log Bytes Ingested*" e "*Logging Throughput*") e ao número de réplicas do Moonshot ativas.

O "*Log Bytes Ingested*" mede o volume total de dados ingeridos, sendo útil para monitorizar o impacto no armazenamento e na rede. Por outro lado, o "*Logging Throughput*" reflete a frequência com que os eventos são registados, ajudando a identificar picos na geração de logs.

Ao analisar ambos em conjunto, é possível obter *insights* valiosos sobre o comportamento do sistema. Se o *throughput* aumentar sem um crescimento proporcional no volume de bytes ingeridos, isso pode indicar a presença de logs pequenos e repetitivos. Em contrapartida, um *throughput* baixo associado a um elevado volume de bytes ingeridos sugere a existência de poucos eventos de grande dimensão, o que pode resultar num consumo excessivo de armazenamento e recursos de rede.

Esta análise permite detetar padrões anómalos e otimizar a geração e gestão de logs, prevenindo desperdícios e garantindo um desempenho mais eficiente do sistema.

Podemos observar o números de réplicas do Moonshot ao somar as instâncias ativas de containers, pois cada Pod possui um único container.

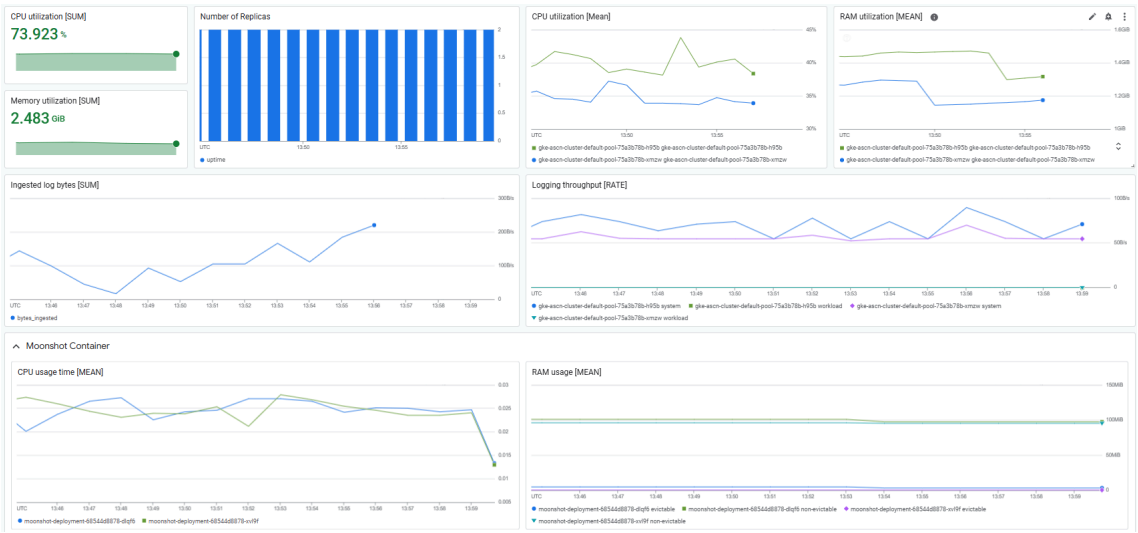


Figura 7: Dashboard

Vale realçar que foi desenvolvido um playbook `monitoring.yml` para conseguirmos importar os ficheiros JSON com as nossas Dashboards para um projeto no GCP, e o projeto é definido pela variável `gcp_project`.

4.2. Ferramentas de avaliação

Neste contexto, é importante destacar a utilização do **Apache JMeter**. Esta ferramenta permite simular diferentes níveis de carga sobre uma aplicação, replicando o comportamento de múltiplos utilizadores ou processos em simultâneo. Através desta simulação, é possível recolher e analisar diversas métricas que oferecem uma visão detalhada do desempenho da aplicação sob diferentes condições.

Entre as funcionalidades do **JMeter**, destaca-se a capacidade de medir o tempo de resposta, a taxa de erro e o *throughput*, permitindo identificar potenciais gargalos e pontos de falha que possam comprometer a experiência do utilizador ou a estabilidade do sistema. Esta análise é essencial para otimizar a aplicação, garantindo que esta responde de forma eficiente mesmo em cenários de elevada utilização.

Com recurso ao **JMeter** foi possível realizar dois tipos de *benchmark* essenciais. O primeiro *benchmark* foca-se em aplicar cargas mais pesadas, mas com um número reduzido de utilizadores. Este tipo de teste é útil para analisar o desempenho da aplicação em operações que exigem mais recursos por sessão, como transações complexas ou processamento intensivo. O segundo *benchmark* simula uma carga mais leve, mas envolve um grande número de utilizadores a interagir com a aplicação ao mesmo tempo. Esse teste é essencial para medir a capacidade de escalabilidade e verificar como o sistema responde quando há múltiplas solicitações simultâneas, replicando um ambiente de grande afluência.

Ao utilizar ambos os *benchmarks*, é possível obter uma análise mais completa do desempenho da aplicação, identificando pontos críticos e garantindo que o sistema oferece uma experiência consistente e estável, mesmo sob diferentes níveis de carga.

4.3. Testes

De forma a simular diferentes cenários com diferentes cargas, optou-se por usar *threads* que representam múltiplos utilizadores simultâneos, ao invés de múltiplas máquinas distintas. Isto quer dizer que os testes foram realizados localmente em uma única máquina, levando que a latência observada seja menor do que seria caso fossem usados sistemas distribuídos ou pedidos *Cloud*.

Para avaliar a escalabilidade da aplicação tanto verticalmente como horizontalmente foram configurados dois planos de teste distintos:

1. **Teste de Escalabilidade Vertical**: Este plano de teste foi concebido para simular operações intensivas realizadas por um pequeno número de utilizadores simultâneos. O teste consiste em realizar uma sequência de ações distintas sequencialmente, como verificar se a aplicação está operacional, efetuar login, criar um DGC e aceder ao DGC.
   - **Justificação:** Este teste tem como objetivo avaliar a capacidade da aplicação em lidar com operações complexas e de alto custo computacional, identificando possíveis gargalos no processamento e assim testar a estabilidade sob diferentes condições de carga intensa.
2. **Teste de Escalabilidade Horizontal**: Neste plano de teste, foi simulada uma utilização ampla e moderada da aplicação, com um grande número de utilizadores que realizam apenas um pedido, ou seja verificar se a aplicação está operacional, todos simultaneamente.
   - **Justificação:** Este teste é focado em avaliar a escalabilidade horizontal da aplicação, isto é, o comportamento do sistema quando submetido a um elevado número de pedidos concorrentes. Este cenário é essencial para validar a robustez e a eficiência na gestão de múltiplas conexões simultâneas.

4.4. Resultados

4.4.1. Teste de Escalabilidade Vertical

Aqui apresentamos uma tabela com o desempenho medido com um número crescente de utilizadores para um *deploy* com **5 réplicas**. Estes valores são resultantes de **N utilizadores** que, ao longo de **5 iterações**, executam o procedimento do segundo teste (verificar conexão à página, efetuar *login*, criar DGC e adquirir DGC criado) com intervalos mínimos entre iterações de **500ms**.

Users	10	30	50
Média tempo resposta (s)	5.724	21.824	72.336
Min (s)	2.142	2.144	2.144
Max (s)	12.156	129.701	300.584

Tabela 2: Desempenho da aplicação para diferentes números de users

Com os resultados anteriores, podemos observamos que o processo é significativamente pesado, envolvendo múltiplos passos como ligar à página, efetuar *login*, criação do certificado (DGC) e acesso ao mesmo. Este conjunto de operações gera uma elevada quantidade de pedidos HTTP em circulação.

Os resultados indicaram que o desempenho mínimo com 50 *threads* é similar ao observado com 10 *threads*, no entanto, o tempo médio de resposta ultrapassou 1 minuto com 50 *threads*, sendo inviável para operações em tempo real. Este comportamento sugere que a aplicação pode atingir o limite de eficiência sob estas condições.

4.4.2. Teste de Escalabilidade horizontal

Aqui apresentamos uma tabela com o desempenho medido com um **número crescente de réplicas**. Estes valores são resultantes de **1000 utilizadores** simultâneos a aceder à página `api/health/` da aplicação num intervalo de **300ms** ao longo de **15 iterações**.

Réplicas	1	3	5
Média tempo resposta (s)	1.863	1.040	0.898
Min (s)	0.417	0.417	0.417
Max (s)	57.333	27.823	21.771

Tabela 3: Desempenho da aplicação para diferentes números de réplicas

A análise dos dados revela que a replicação do servidor aplicacional traz benefícios, refletindo-se numa redução do tempo de execução à medida que o número de réplicas aumenta. No entanto, essa redução não ocorre de forma linear, o que sugere que existe um ponto de saturação, a partir do qual o aumento do número de réplicas deixará de contribuir para melhorias significativas no desempenho.

Por último, apresentamos uma tabela com o desempenho medido com um número crescente de utilizadores para um *deploy* com **5 réplicas**. Estes valores são resultantes de **N utilizadores** simultâneos a aceder à página `api/health/` da aplicação, num intervalo de **300ms** ao longo de **15 iterações**.

Users, N	100	500	1000
Média tempo resposta (s)	0.588	0.515	0.898
Min (s)	0.417	0.417	0.417
Max (s)	2.967	3.502	21.771

Tabela 4: Desempenho da aplicação para diferentes números de users

Os resultados mostram que a aplicação responde bem a um aumento expressivo de utilizadores simultâneos. A análise da tabela revela que o tempo médio de resposta aumenta de forma pouco significativa ao passar de 100 para 1000 utilizadores. Este desempenho sugere que o sistema consegue alocar recursos de forma eficiente, permitindo-lhe suportar elevados níveis de concorrência e confirmando a sua capacidade de escalabilidade horizontal.

5. Conclusão

Com a realização deste trabalho foi possível consolidar o conhecimento adquirido ao longo das aulas, tanto praticas e teóricas do semestre.

Consideramos que os objetivos essenciais e pretendidos foram atingidos, sendo possível providenciar e fazer deploy da aplicação Moonshot de forma autónoma em qualquer ambiente desejado, sendo capaz de armazenar os dados de forma persistente e de escalar de acordo com o número de pedidos recebidos pela aplicação.

Porém, reconhecemos que há espaço para possíveis melhorias e complementos. Uma melhoria seria replicarmos a base de dados utilizando *StatefulSet*, que permitiria descentralizar a base de dados e manter os dados sincronizados entre diferentes bases de dados. Poderíamos também ter ido mais a fundo no benchmarking da aplicação, para obter uma melhor perceção de outras possíveis falhas. A nível de código acabamos por não conseguir implementar a tempo ConfigMaps, que possibilitariam uma definição mais clara e flexível das variáveis do sistema.

Concluindo, explorámos os fundamentos de **Kubernetes**, uma tecnologia amplamente utilizada atualmente para orquestrar aplicações em nuvem. Também nos familiarizámos com o **Ansible**, uma ferramenta de automatização, e tivemos a oportunidade de trabalhar diretamente com a **Google Cloud**. Este trabalho prático permitiu-nos compreender melhor os desafios inerentes à implementação de software em ambientes reais e a sua enorme importância no contexto do mundo real, bem como a familiarização com algumas novas ferramentas que acreditamos serem indispensáveis, tanto para o mercado de trabalho como para os nossos projetos pessoais.