ASCN

Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia informática

Aplicações e Serviços de Computação em Nuvem

Ano Letivo de 2024/2025

Moonshot

Augusto Campos, pg57510
Carlos Alberto Silva, pg57518
Eduardo Cunha, pg55939
Tiago Alexandre Silva, pg57614
Tiago Pinheiro da Silva, pg57617

5 de janeiro de 2025

Grupo 13

Índice

1. Introdução

Neste relatório, elaborado no âmbito da unidade curricular de Aplicações e Serviços de Computação em Nuvem, descreve-se o trabalho desenvolvido, tendo como objetivo a implementação da aplicação *Moonshot*, utilizando os serviços da *Google Cloud*. O processo de instalação foi totalmente automatizado através da ferramenta *Ansible*, com base nos conhecimentos adquiridos nas aulas práticas e teóricas.

A aplicação Moonshot foi projetada para gerir a emissão, validação e armazenamento dos Certificados Digitais Verdes (DGCs - Digital Green Certificates), que contêm informações sobre vacinação, testes e recuperação da COVID-19. Destinada a utilizadores autorizados, como profissionais de saúde, a aplicação permite criar e administrar esses certificados de forma segura e eficiente.

A implementação envolveu o uso do Kubernetes, através do serviço *Google Kubernetes Engine* (GKE), que oferece uma plataforma robusta para a gestão automatizada dos containers que compõem a aplicação, garantindo replicação e escalabilidade automática.

Ao longo deste documento, serão detalhadas as decisões tomadas durante o processo de instalação, desde a configuração inicial até à plena operacionalização da aplicação. Serão ainda analisadas as opções escolhidas para a implementação do auto scaling, bem como a configuração do balanceamento de carga, recorrendo ao GKE. Adicionalmente, será abordada a utilização de ferramentas como o JMeter e os serviços de monitorização da *Google Cloud Platform*, os quais foram essenciais para a avaliação contínua do desempenho da aplicação *Moonshot*.

2. Arquitetura e Componentes da aplicação

A aplicação *Moonshot* foi construída utilizando *Python* com o *framework Django REST Framework* (DRF), especializado em desenvolvimento de APIs REST. A aplicação utiliza uma base de dados *PostgreSQL* para armazenar informações cruciais relacionadas aos Certificados Digitais Verdes (DGCs), como registros de vacinação, testes e status de recuperação. O *PostgreSQL* foi escolhido como sistema de gerenciamento de banco de dados, configurado para rodar dentro do ambiente *Kubernetes*.

Para hospedar a aplicação, foi utilizado o Google Kubernetes Engine (GKE) da Google Cloud, que oferece uma solução escalável e resiliente para gerenciamento da infraestrutura. A aplicação Moonshot e a base de dados PostgreSQL são encapsulados em imagens Docker, garantindo a portabilidade e a facilidade de implantação. Essas imagens Docker serão utilizadas para criar os *containers* necessários para a execução da aplicação, sendo orquestrados pelo *Kubernetes*.

A aplicação *Moonshot* fica acessível ao exterior através de um Service do tipo *Load Balancer* na porta 80, utilizando o IP fornecido pelo Google Cloud. Este setup permite que a aplicação seja escalada e gerenciada de forma eficiente em um ambiente de nuvem. Este serviço conta com *sessionAffinity* ao nível do *ClientIP*, de modo a manter a coerência da sessão entre um determinado cliente com uma só instância da camada aplicacional. Foi a decisão mais sensata de um ponto de vista do lançamento da aplicação no mundo real, embora isso dificulte o processo de benchmarking.

No contexto da Tarefa 2, também foi lançado um Horizontal Pod Auto Scaler (HPA) para gerir o número de réplicas da camada aplicacional, isto é, para ordenar a criação ou a destruição de pods moonshot-deployment com base na utilização média de cpu.

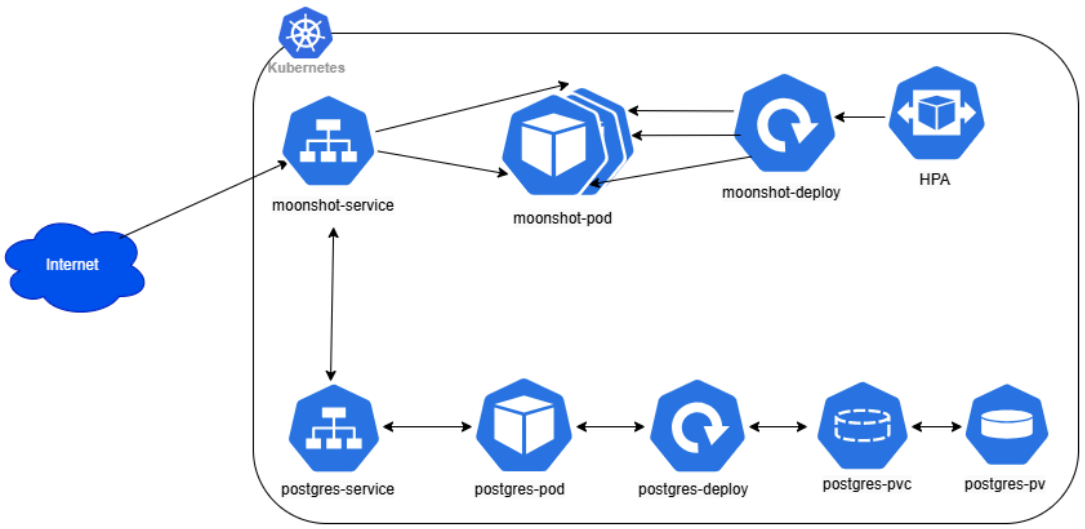

Fig.1 - Arquitetura Kubernetes

3. Instalação e Configuração

A implementação do projeto foi realizada utilizando o *Google Kubernetes Engine* (GKE) da *Google Cloud* em conjunto com o *Ansible*, uma plataforma robusta para automação de tarefas de configuração e gestão de sistemas. O *Docker* desempenhou um papel essencial neste processo, encapsulando a aplicação em contêineres para garantir isolamento, portabilidade e simplicidade na implementação.

Componentes e Ferramentas Utilizadas

Docker

O Docker foi responsável por encapsular a aplicação em contêineres, criando um ambiente isolado e consistente para o backend.

- O ficheiro Dockerfile define o ambiente de execução, instalando todas as dependências necessárias e configurando o servidor para operar a aplicação.

Kubernetes (K8s)

O Kubernetes foi utilizado para orquestrar os contêineres, assegurando a escalabilidade e a gestão eficiente da aplicação.

- Os ficheiros ***".yaml"*** contêm configurações cruciais, incluindo o deployment, a gestão de réplicas do backend, a definição de Persistent Volume Claims (PVCs) e os serviços necessários para expor endpoints e permitir a comunicação entre os pods.

Ansible

O Ansible desempenhou um papel central na gestão integrada do ambiente Kubernetes e no ciclo de vida do cluster GKE, oferecendo:

- Automação de tarefas, como a criação e destruição do cluster GKE, configuração do Kubernetes e deploy/undeploy da aplicação bem como da sua base de dados (*PostgreSQL*)
- Administração eficiente do ambiente Kubernetes através de módulos como o ***k8s***, permitindo a criação e controlo dos componentes necessários à orquestração dos contêineres.

.

4. Exploração e otimização da aplicação

4.1. Considerando a instalação base proposta pelo grupo para a Tarefa 1:

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

R:

No que diz conta à base de dados um aumento no número de clientes pode sobrecarregar o desempenho da base de dados, especialmente em operações de leitura e escrita intensiva, como registros de vacinação e testes.

Já no caso da camada aplicacional para operações que exijam maior poder computacional, esta também é uma camada que se beneficiará da replicação.

Por último no PVC, o desempenho do armazenamento pode limitar operações de escrita/leitura intensivas do *PostgreSQL*.

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

R:

O mesmo será explorado nos capítulos de Avaliação e Resultados. Contudo, prevê-se desde logo que o desempenho se vá deteriorando à medida que o número de clientes e a carga de trabalho aumente. Esse é o principal motivo que nos leva a

querer implementar métodos de replicação para aumentar a disponibilidade e tentar manter o desempenho.

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

R:

Antes de aplicar qualquer tipo de replicação, a base de dados, o serviço que expõe o moonshot e o PVC constituem pontos únicos de falha.

4.2. Com base nas respostas dadas para as respostas anteriores:

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

R:

Implementar replicação da base de dados e camada aplicacional para melhorar o desempenho e garantir alta disponibilidade, adicionar redundância ao moonshot-service (Load Balancer) e implementar um sistema de cache (como *Redis*) para armazenar resultados de consultas frequentes, reduzindo a carga na base de dados.

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

R:

Desempenho:

- A replicação da base de dados reduzirá os tempos de resposta para operações de leitura.
- A replicação da camada aplicacional será importante para pedidos que exijam maior capacidade computacional.

- O uso de um sistema de cache reduzirá a dependência de consultas diretas à base de dados, diminuindo a carga no backend.

Resiliência:

- A replicação da base de dados e da camada aplicacional aumentam a disponibilidade da aplicação em caso de falhas.

Estas otimizações criam um ambiente mais robusto, escalável e preparado para lidar com picos de tráfego e eventuais falhas, garantindo uma melhor experiência ao usuário.

5. Monitorização

Relativamente à monitorização, usamos as ferramentas da Google Cloud para o efeito, eles também proporcionam esse serviço. Procedemos à criação de uma dashboard personalizada, mas também tiramos partido de muitas outras já previamente construídas, nomeadamente ao nível da monitorização dos pods moonshot-deployment. As métricas às quais mais prestamos atenção foram a utilização do CPU e de memória, devido principalmente à afinidade que estas métricas têm em relação à política de escalonamento escolhida. Seguem-se exemplos de dashboards que foram úteis na hora de desenvolvimento de testes.

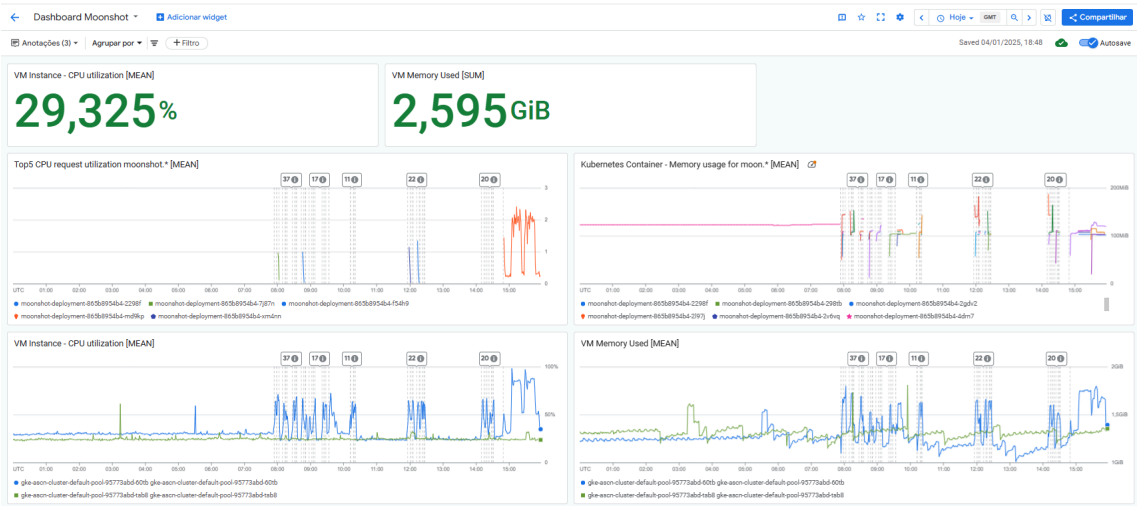

Fig.2 - Dashboard personalizada

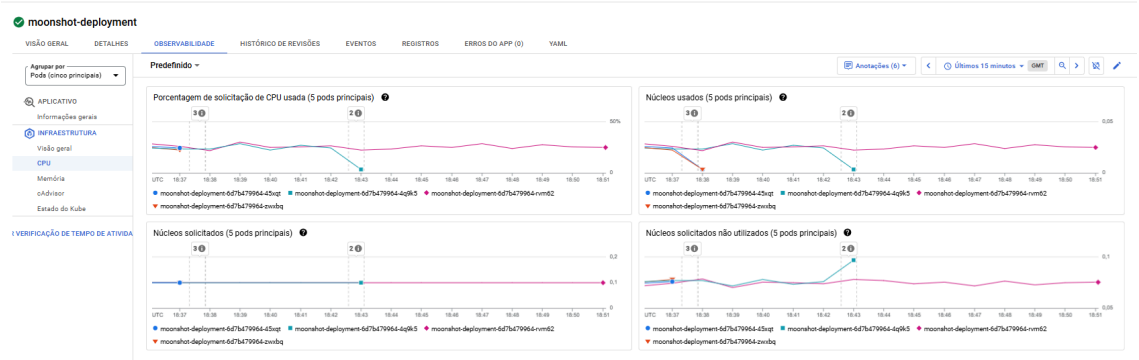

Fig.3 - Dashboard CPU no contexto moonshot-deployment

6. Avaliação

A ferramenta utilizada para proceder aos testes da aplicação foi o Apache JMeter, visto que os elementos do nosso grupo já haviam trabalhado com essa ferramenta nas aulas práticas da disciplina.

Os “Test Plan’s” foram feitos de maneira a poderem no futuro ser automatizados através de User Defined Variables podendo assim adaptar o ip da app e a porta dinamicamente. Como há também o intuito de efetuar testes de login, as variáveis “app_notifier_email” e “app_notifier_password” também podem ser modificadas no momento em que se executa o teste. No role que desenvolvemos para automatizar testes, também alteramos o número de threads e o número de iterações do Thread Group de acordo com as variáveis ansible definidas no contexto desse mesmo role.

Infelizmente não conseguimos tirar partido das flags do JMeter para proceder à mudança de variáveis, tendo-o feito a partir da modificação do próprio ficheiro .jmx com ansible.builtin.replace e expressões regex.

Fig.4 - Exemplo do uso de variáveis na configuração dos testes

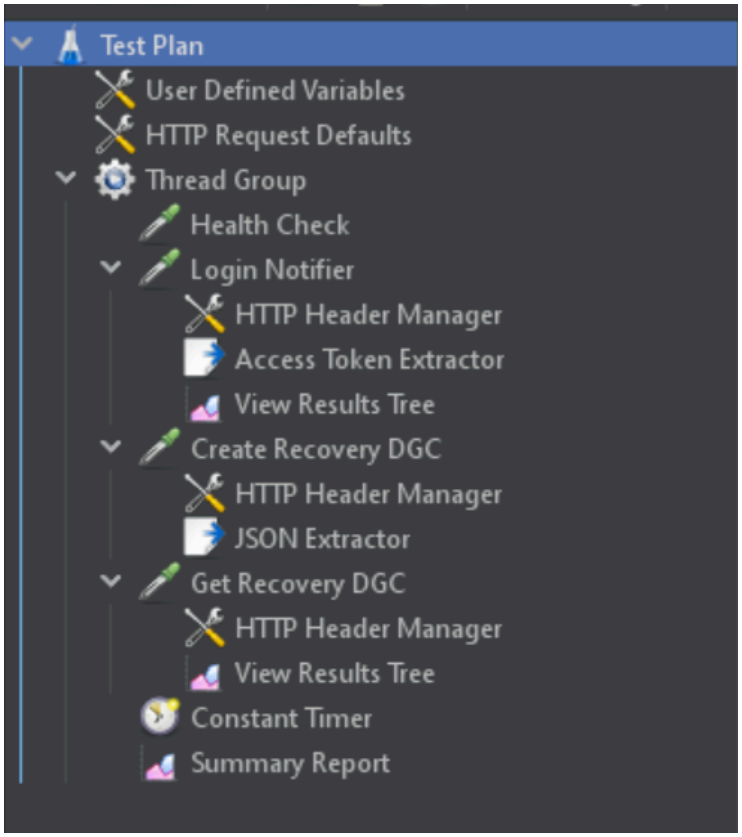

Fig.5 - Configuração teste com requests variados

7. Resultados

O seguinte teste teve como propósito simular algo próximo a uma utilização natural do Moonshot, isto é, consultar a página inicial, efetuar login, criar um “Recovery DGC” e consultar esse DGC. Para isso foram simulados 10 users (threads), com pedidos a cada 300ms ao longo de 15 iterações. A cada iteração, cada user efetua cada um destes requests.

Com a ajuda de monitorização em tempo de execução dos testes, foi possível perceber que a utilização de CPU do lado aplicacional não é tão elevada como a utilização da memória no Postgres-deployment. Esse dado juntamente com o elevado Response Time (RT) do Login e do “Create DGC” sugere que para o provisionamento desta aplicação no mundo real, a replicação também tenha que ocorrer ao nível da camada de dados.

Requests	Average RT (ms)	Min RT (ms)	Max RT (ms)	Throughput (/min)
/health	577	271	2206	14,8
Login	15951	1383	43440	14,7
Create DGC	17851	835	40440	14,8
Get DGC	4303	503	12681	14,9
TOTAL	9670	271	43440	58,6

Tabela 1 - Resultados teste com requests variados

Passaremos a testes mais simples que têm como finalidade demonstrar os benefícios da escalabilidade da camada aplicacional. Vamos submeter o sistema a sucessivos pedidos à página /api/health com um número variado de users (threads).

Nº threads	32	64	128	256	512	1024 (erros)
Average RT (ms)	303	333	334	407	683	1738
Throughput (/s)	51,0	92,4	185,9	249,8	326,0	91,5

Tabela 2 - Pedidos api/health com 4 réplicas moonshot-deployment

Nº threads	32	64	128	256	512	1024 (erros)
Average RT (ms)	312	321	349	420	707	2371
Throughput (/s)	50,5	100,6	182,9	274,1	369,1	138,3

Tabela 3 - Pedidos api/health com 1 réplica moonshot-deployment

Como podemos observar nas tabelas acima não foi possível observar diferenças significativas entre 1 e 4 réplicas. No entanto, algo revelador foi demonstrado pelo gráfico de percentagem de solicitação de CPU. Os pedidos não estavam a ser bem distribuídos pelos vários pods moonshot-deployment. Isto quer dizer que na prática, o escalonamento automático horizontal de pods não estava de facto a ter efeito. Este comportamento é explicado devido à sessionAffinity na configuração do serviço. Embora para aceder à página inicial esta não fosse necessária, num contexto natural envolvendo autenticação e memória de sessão, faz todo o sentido um utilizador estar conectado à mesma instância por razões de coerência de contexto. Uma das alternativas possíveis para testes mais realistas e fidedignos seria correr testes em várias máquinas virtuais de ip's diferentes.

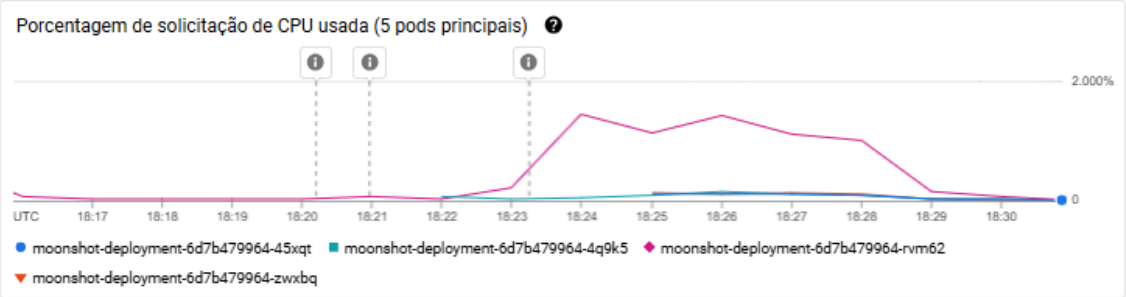

Fig.6 - Carga mal distribuída pelos pods moonshot-deployment

Visto que o objetivo deste teste era demonstrar os benefícios da replicação da camada aplicacional, podemos concluir que foi um teste falhado. No entanto, num contexto mais real, a sessionAffinity não afetaria, já que os utilizadores reais fazem pedidos de ip's e localizações diferentes.

8. Reflexão final

A realização deste projeto prático foi essencial para consolidar o conhecimento adquirido ao longo das aulas teóricas e práticas da disciplina. Foi uma oportunidade de aplicar na prática os conceitos explorados em sala, enfrentando desafios reais de implementação.

O grupo considera que alcançamos o objetivo principal: automatizar de forma significativa o processo de provisionamento e deployment da aplicação Moonshot.

Reconhecemos, no entanto, um erro no processo de instalação inicial: a utilização de um IP estático, que resultou na impossibilidade de ultrapassarmos o segundo checkpoint de avaliação. Contudo, acreditamos ter corrigido essa falha com sucesso após recebermos o respetivo feedback. Ainda assim, consideramos que o trabalho realizado foi consistente, bem estruturado e alinhado com os objetivos do projeto.

Adicionalmente, destacamos a capacidade da aplicação de armazenar dados de forma persistente e de escalar de maneira dinâmica, ajustando-se ao volume de pedidos recebidos, o que reforça o mérito das soluções implementadas.

No entanto, reconhecemos que há aspetos a melhorar nomeadamente mecanismos de replicação da base de dados PostgreSQL, para agilizar processos de Login, entre outras operações que recorram à base de dados.

A automatização dos testes também pode ser melhorada sendo que de momento só tem referência para um ficheiro .jmx. A própria análise do output de onde se pretende extrair resultados também pode ser trabalhada sendo que de momento apenas é apresentado o “Average Response Time” e o Throughput.

Ainda no âmbito de testes, gostaríamos de submeter o sistema a pedidos a partir de variadas máquinas com ip’s diferentes, de modo a perceber melhor qual o número de réplicas de cada componente que mais se adequa à demanda que se espera que a aplicação tenha num contexto real.

Ficou igualmente em tentativa a aplicação de um Ansible Vault para garantir a proteção de dados sensíveis. Esta ferramenta encripta informações críticas e seria de grande importância implementá-la, mas não conseguimos concretizá-la.

Em síntese, este trabalho proporcionou-nos uma experiência prática valiosa, permitindo aprofundar o conhecimento sobre ferramentas como o Kubernetes, uma solução moderna e amplamente adotada para a gestão de aplicações em contêineres, e o Ansible, utilizado para automação de processos. Além disso, tivemos a oportunidade de explorar a Google Cloud Platform num ambiente aplicado. Este projeto deu-nos uma visão clara dos desafios reais associados à instalação e gestão de aplicações no mundo real, bem como das ferramentas disponíveis para os resolver.