Aplicações e Serviços de Computação em Nuvem (MEI)

Trabalho Prático

https://github.com/edumagalhaes10/ASCN-Grupo16.git

Cláudia Peixoto Silva
A93177

Eduardo Costa de Magalhães
PG50352

Laura Nunes Rodrigues
PG50542

Mariana Filipa da Silva Rodrigues
PG50628

Rui Pedro Chaves Silva Lousada Alves
PG50745

12 de abril de 2023

Universidade do Minho
Escola de Engenharia

Capítulo 1

Introdução

No âmbito da unidade curricular de Aplicações e Serviços de Computação em Nuvem foi-nos proposto pelos docentes um projeto que visa por em prática os conhecimentos adquiridos de automatização do processo de instalação, configuração, monitorização e avaliação da aplicação Ghost.

Na execução deste trabalho foi usada a ferramenta Ansible para a realização da primeira tarefa de modo a automatizar a instalação e configurar a aplicação Ghost no serviço da Google Kubernets Engine (GKE) da Google Cloud. Para a realização da monitorização foram usadas ferramentas de monitorização disponibilizadas pela Google Cloud Platform e relativamente à avaliação experimental, esta tem como intuito demonstrar o método que foi usado para validar experimentalmente a instalação e configuração da aplicação Ghost.

Por fim, de modo a garantir escalabilidade e resiliência dotámos a aplicação de mecanismos de replicação que permitiram melhorar o desempenho e resiliência da instalação da aplicação.

Capítulo 2

Implementação do Ghost

A primeira etapa deste projeto foi a implementação da aplicação Ghost. Como foi explicado previamente, para esta parte do projeto, recorremos às ferramentas Ansible e Google Kubernetes Engine (GKE).
A ferramenta Ansible permitiu-nos automatizar o processo de comunicação com a ferramenta GKE. Esta facilitou o processo de deployment das diferentes partes da aplicação Ghost. Conseguimos reconhecer que, sem o uso da mesma, este processo de instalação seria muito mais difícil e menos automatizado. Para este efeito, esta ferramenta disponibiliza módulos que permitem trabalhar com a plataforma Google Cloud e com o serviço de kubernetes desta.

Adicionalmente, a ferramenta Ansible disponibiliza a ferramenta Ansible Vault que nos permitiu adicionar um pouco de segurança à nossa solução através da encriptação e proteção de *passwords*.
O serviço GKE da Google Cloud Platform (GCP) permitiu-nos criar *containers* para hospedar a aplicação Ghost e os diferentes componentes que foram necessários lançar para o funcionamento da mesma. A GCP forneceu-nos também mecanismos de monitorização que foram usados em etapas posteriores.
Para a instalação e configuração da aplicação Ghost, recorremos à criação de diferentes pods de deployment e serviço. É nos pods de Deployment que temos os containers necessários para o funcionamento do nosso programa. Para obtermos estes containers, utilizámos as Docker Image do MySql e do próprio Ghost.

Quando analisámos a Docker Image do Ghost disponibilizada, concluímos que existia um conjunto de variáveis de environment que deveriam ser introduzidas. Para que a aplicação pudesse estar funcional, deveríamos introduzir as variáveis de *Environment*: *URL, Database e Email* (uma vez que estas eram as variáveis requisitadas para produção).
Assim, antes da implementação do Ghost, foi necessário criar um *container* com a Image MySQL, ficando esta disponível num dado pod. Associado a este pod, definimos também um serviço que permite que a app comunique com a base de dados. Adicionalmente, de modo a que os dados armazenados na base de dados sejam persistentes e não se encontrem armazenados apenas durante o tempo de vida do pod, foi necessário disponibilizar um *Persistent Volume* ao pod da Base de Dados. Para a criação deste, definimos um Storage Class. Por sua vez, este permite que o GKE crie um persistent volume conforme a necessidade deste para tal.

Para a variável de *Environment* **Email**, criámos uma conta *mailtrap* que nos permitiu definir o serviço de email necessário para a funcionalidade de Subscrição disponibilizada no site do Ghost.

Posteriormente, criámos também, de forma automática, uma conta de Administrador para que pudéssemos utilizar e testar as funcionalidades deste no site.

Juntamente com o ghost deployment, criámos um *Service*. Este permitiu que a aplicação pudesse ser acedida através de um browser por clientes a partir do exterior. Para tal, utilizámos um LoadBalancer que foi pertinente para a posterior implementação da replicação.

Para fornecer a variável de *Environment* **URL**, determinámos o IP do Load Balancer definido anteriormente, e definimos que a variável URL seria composta por esse IP associado a uma porta escolhida(neste caso a porta 80), no formato *IP:PORTA*. Este URL pode variar porque como o mesmo está associado a um LoadBalancer, quando este é reiniciado, o seu ip altera-se.

No final, conseguimos fornecer uma solução de implementação automática da aplicação Ghost. A nossa implementação segue a estrutura Multitier. Associamos a nossa solução a esta arquitetura uma vez que cada *server* (base de dados e servidor aplicação), funciona como *client* do servidor vizinho.

Ao seguirmos uma arquitetura distribuída, podemos garantir que o serviço disponibilizado é escalável, mais resistente e modular.

Capítulo 3

Monitorização

Na segunda tarefa, era pretendido que utilizássemos a ferramenta de monitorização da Google Cloud Platform. A GCP possui um sistema integrado de monitorização que nos permite observar a variação de dadas métricas durante a execução da aplicação Ghost.

A plataforma Google Cloud fornece métricas, eventos e metadados apresentados através da colheita de informação dos componentes envolvidos. Identifica problemas e descobre padrões, ajudando a avaliar a experiência que é possível fornecer a um utilizador e a facilitar a análise da qualidade da implementação.

Para a nossa solução, optámos por implementar um dashboard, com gráficos lineares e alguns valores numéricos. Estes permitem visualizar como varia o CPU e a memória utilizada pelos nós criados, o número de pods criados, o throughput. Podemos também visualizar como uso do CPU e da memória para cada Container (ex: Ghost e MySQL). Optámos por criar este dashboard, uma vez que consideramos que os restantes dashboards disponibilizados pelo GCP eram relativamente confusos e de difícil visualização, que geravam a necessidade de utilizar filtros. Assim, ao automatizarmos a criação do dashboard escolhido, podemos mais facilmente observar as métricas que consideramos pertinentes, e evitar o processo de aplicar filtros aos dashboards já existentes.

Assim, a monitorização permitiu-nos compreender como o programa iria utilizar os recursos que lhe estavam disponíveis e visualizar as anomalias que poderiam ocorrer durante a execução.

Com estas métricas escolhidas, desenvolvemos um Ansible playbook para o *role* de **monitorização**. Este playbook permitiu-nos criar os dashboards com as métricas escolhidas de forma automática, recorrendo ao módulo *command*.

Os dashboards que definimos serão criados após a inicialização da aplicação Ghost e poderão ser catalogados na categoria "Custom"e consultados na janela de monitorização na página browser da GCP.

Capítulo 4

Avaliação Experimental

Numa próxima etapa, pretendemos verificar como é que o nosso sistema reagiria a um elevado número de pedidos. Para tal, decidimos implementar um mecanismo de benchmarking utilizando a ferramenta *Jmeter* e o *monitoring* disponibilizado pelo GCP.

Na nossa solução, utilizamos a ferramenta *Jmeter* para desenvolver um conjunto sintético de *requests* HTTP para a página *https//{ghost_ip}:{ghost_port}/ghost*.
Começamos por definir uma *Thread Group* que representa os *benchmark clients* que efetuam *requests* HTTP infinitamente. Após a definição de *benchmark clients*, implementamos um *timer* aos *requests* a ser feitos à página do ghost pretendida. Com isto, e variando o número de *threads* pertencentes à *Thread Group* e/ou o número de *ms* em que o *timer* efetua *requests* à página, pretendemos visualizar através das ferramentas de monitorização definidas anteriormente o comportamento e a variação das métricas monitorizadas no *cluster*.

Para avaliar a nossa implementação, definimos um dado *benchmark client* irá executar 100 pedidos ao URL mencionado anteriormente, a cada 300ms. Com este testes recolhemos alguns dados.
Durante o período observado, podemos verificar que houve um aumento da percentagem de CPU usado na totalidade de todos os nós e que como seria de esperar houve um aumento no loading throughput.

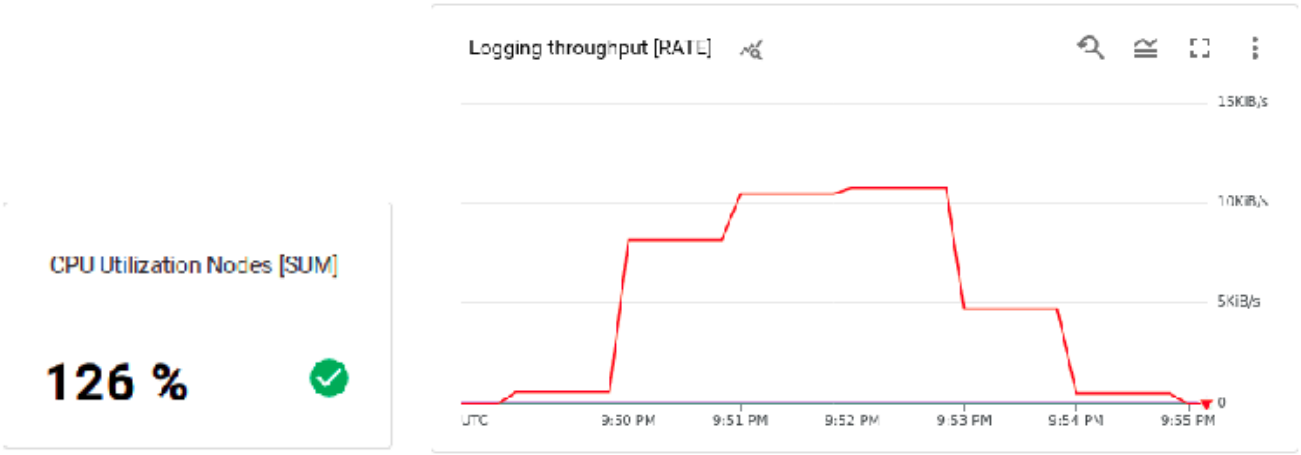

Figura 4.1: CPU usado e percentagem de throughput

Recorrendo às métricas obtidas pelo JMeter, podemos determinar que a percentagem de throughput durante

este período foi *58.3/sec*. Deste modo, concluímos que o nosso servidor aplicacional consegue responde a 58.3 pedidos por segundo.

Ao observar a quantidade de CPU utilizado pelo Container Ghost, reparamos que estes pedidos obrigavam à utilização da totalidade do CPU *requested* pelo container. Podemos ainda constatar que este Container viu também um aumento na quantidade de memória utilizada.

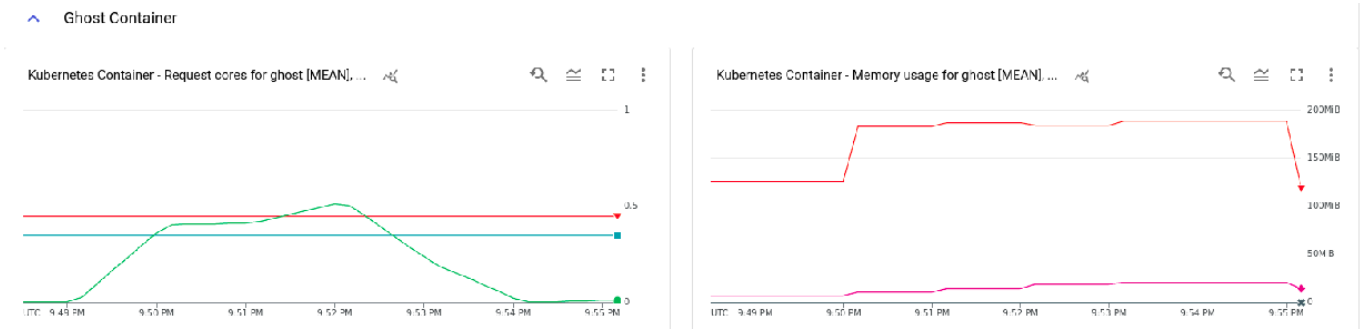

Figura 4.2: Request cores e memory usage do Ghost

Analisando a imagem em maior detalhe, podemos observar que num espaço de 5 min, o teste com a especificação: 100 pedidos a cada 300ms, levou ao uso da totalidade do CPU disponibilizado para a aplicação Ghost. Dado que o throughput é 58.3/s e que a cada 0.3s mais 100 pedidos são gerados, é possível concluir que durante as iterações haverá uma acumulação de pedidos para serem respondidos, o que consequentemente leva ao aumento do CPU necessário.

Assim, se o nosso programa receber 100 pedidos a cada 0.3s, será necessário fazer alterações à aplicação.

Dado que a percentagem de CPU usada na totalidade pelos nós poderá alcançar um máximo de 300%, dado que nós estamos a utilizar 3 nodes e a percentagem corresponde à soma do cpu utilizado. Assim, podemos determinar que poderíamos aumentar o valor requisitado pelo container para cada Pod.

Adicionalmente, tentamos observar o que aconteceria se a aplicação recebesse mais pedidos, mas menos frequentes. Para tal, verificamos o que ocorreria se a aplicação recebesse 500 *HTTP requests* a cada 30s. Desse teste obtivemos os gráficos de throughput e uso de CPU:

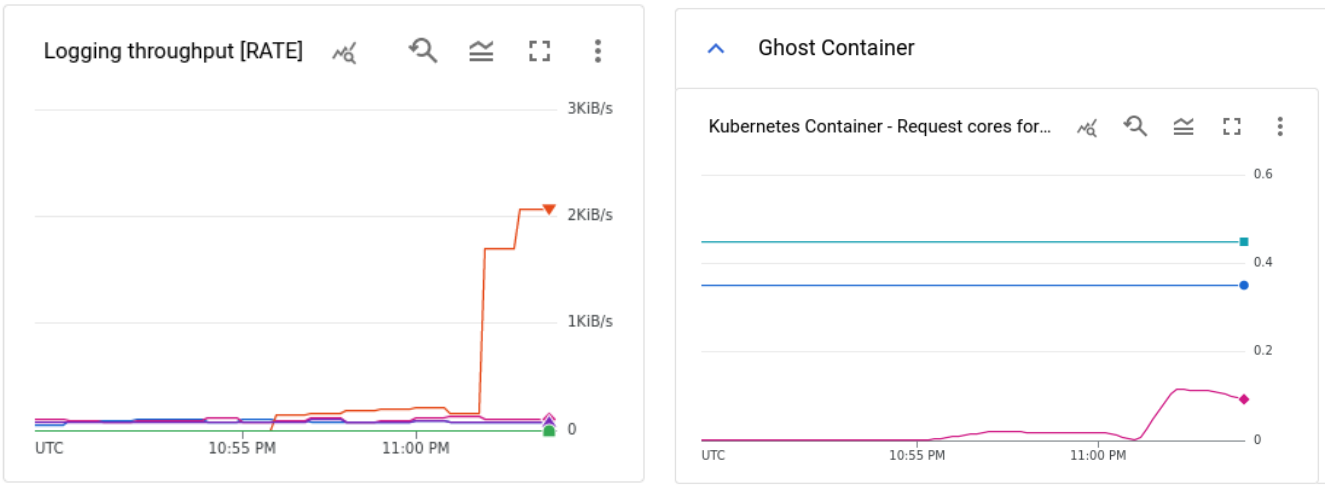

Figura 4.3: Throughput e request cores do Ghost

Podemos verificar que quando os pedidos são mais espalhados temporalmente, a implementação é capaz de responder a um número superior de pedidos simultâneos dado que o intervalo de tempo que possui para responder aos pedidos à espera antes de receber novos pedidos é maior.

Contudo, no mundo real, não poderemos contar com a existência de um intervalo entre os pedidos HTTP, e também não poderemos assumir que os pedidos chegarão de forma previsível.

Para concluir este capítulo, consideramos que, apesar deste conjunto de *requests* permitir a observação da capacidade da aplicação de responder a um elevado conjunto de pedidos, estes pedidos não correspondem a uma representação completa do conjunto de ações que cada tipo de utilizadores (Admin e Utilizador) poderão executar na realidade. No entanto, para uma correta avaliação do sistema, consideramos *benchmarking* uma abordagem de testes sistemática e que permitem uma correta avaliação do nosso sistema.

Capítulo 5

Escalabilidade e Resiliência

Esta tarefa consistia em conseguir replicar e melhorar o desempenho de um dos componentes da aplicação Ghost. Assim, decidimos focar-nos apenas no **servidor aplicacional**.

Como vimos anteriormente, o nosso sistema foi implementado seguindo uma arquitetura Multitier. Assim, a nossa solução deverá ser capaz de apresentar *escalabilidade* e *resiliência*.

Como passo seguinte da nossa solução, atribuímos ao nosso deployment a capacidade de criar réplicas para o *Pod* do servidor aplicacional da aplicação Ghost tendo em vista a taxa de utilização de CPU do mesmo. Deste modo, pretendíamos fornecer maior poder computacional e maior resiliência à nossa aplicação, dado que existem mais pods disponíveis e caso um *Pod* fique indisponível continuávamos a ter pontos de acesso à aplicação.

Utilizando o HorizontalPodAutoScaler, quando a aplicação Ghost se encontrar sobrecarregada, isto é, com uma taxa de utilização de CPU superior a 75%, será criada mais uma réplica do servidor aplicacional do Ghost. Este componente é iniciado com um pod e serão criadas réplicas da forma explicada anteriormente, sendo o número máximo criado 3. Na nossa solução não é possível criar mais pods, uma vez que o número de nós utilizados fornece CPU que permite a criação de apenas 3 réplicas, tendo em conta o CPU atribuido a cada Pod.

Como mencionámos anteriormente, quando implementámos o Ghost tornámos a aplicação acessível através de um browser utilizando um LoadBalancer. Graças a este, esperamos que a "carga de trabalho"seja distribuída pelas diferentes réplicas do servidor aplicacional do Ghost.

Para analisarmos o funcionamento do Horizontal Pod Autoscaler, utilizamos a avaliação experimental definida na etapa anterior. Definimos que o JMeter iria executar 100 pedidos a cada 300ms. Quando estes pedidos foram feitos, o container reagiu como na imagem apresentada de seguida:

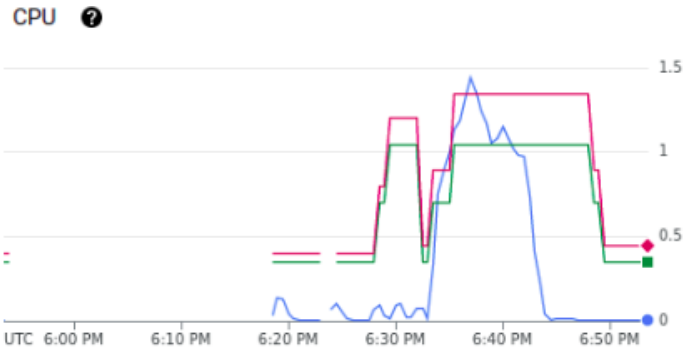

Figura 5.1: Variação do CPU usage

A linha azul representa o CPU utilizado para responder aos pedidos recebidos, a linha verde o CPU request para o container, e a linha cor de rosa o CPU limit para o container.

Como podemos observar quando o CPU utilizado aumenta para além do request, a quantidade de CPU request e CPU limit aumenta. Este aumento deve-se à inicialização de um novo Pod para este container. Como podemos observar, posteriormente, quando os pods não forem necessário, estes serão removidos.

Capítulo 6

Conclusão

Em modo de conclusão, realizamos uma visão crítica e refletida do trabalho realizado. Consideramos que os requisitos foram cumpridos e que o deployment da aplicação Ghost está funcional, é escalável e resiliente.

A nossa implementação recorre a funcionalidades do Google Cloud como o Google Kubernetes Engine e Ansible, que ajudaram a automatizar o deployment, escalabilidade e gestão da aplicação Ghost, balanceando também a carga através de um LoadBalancer.

Numa análise final gostaríamos de salientar que, infelizmente, verificámos que a nossa solução não apresenta uma performance ideal. Apesar do uso de um Horizontal Autoscaler e o resultado do *Deployment* automático conseguir reagir ao aumento de pedidos, com o número máximos de nodos que utilizámos (3), esta não consegue servir muito mais do que 100 pedidos simultâneos pois necessitaria de um maior número de pods para manter um uso de CPU razoável.