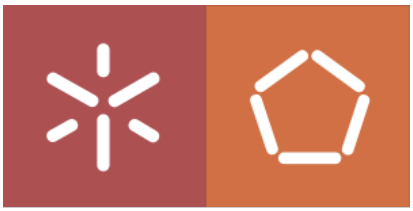

Universidade do Minho
Escola de Engenharia

Universidade do Minho

Mestrado em Engenharia Informática

Aplicações e Serviços de Computação em Nuvem
Grupo 25 - https://github.com/RuiFCMoreira/ASCN

Bernardo Saraiva (PG50259) José Gonçalves (PG50519)
João Ferreira (PG50461) João Lourenço (PG50464)
Rui Moreira (PG50736)

dezembro, 2022

Conteúdo

Capítulo 1

Introdução

O presente relatório serve como estrutura de explicação e apresentação da realização do Trabalho Prático no âmbito da unidade curricular de Aplicações e Serviços de Computação em Nuvem.

O Trabalho Prático terá como objetivo primordial a instalação, monitorização e avaliação da aplicação **Ghost**. Além disso, e para alcançar com sucesso a realização deste projeto, serão ainda colocados em prática todos os assuntos e conceitos lecionados nas aulas ao longo do semestre, por forma a garantir a automatização completa de processos de instalação, monitorização e avaliação da aplicação.

Desta forma, o relatório estará dividido em 5 secções, explicando e apresentando cada uma delas sequencialmente:

- **primeira secção** - arquitetura e componentes da aplicação;
- **segunda secção** - ferramentas e abordagens utilizadas de forma a obter a instalação e configuração automática da aplicação;
- **terceira secção** - abordagem escolhida para o mecanismo de replicação;
- **quarta secção** - ferramentas de monitorização e métricas escolhidas;
- **quinta secção** - ferramentas de avaliação utilizadas e análise de resultados.

Capítulo 2

Arquitetura e componentes principais da aplicação

A estrutura da aplicação recai sobre uma arquitetura **Cliente-Servidor**, sendo que neste caso o servidor equivale ao Cluster do Kubernetes. Dentro deste cluster estão separados a aplicação Ghost e a base de dados, aumentando assim a possibilidade de escalabilidade, já que desta forma é possível escalar independentemente, de acordo com as necessidades de cada um. Além disso, garante uma maior flexibilidade, isto é, faz com que seja possível desenvolver e/ou alterar funcionalidades de forma independente, uma vez que se trabalha apenas com o respetivo pod, em vez de refazer a aplicação inteira.

Como componentes principais estão a aplicação **Ghost** e a sua base de dados **MySQL**, ambos instalados e configurados automaticamente utilizando a ferramenta Ansible no serviço *Google Kubernetes Engine* (GKE).

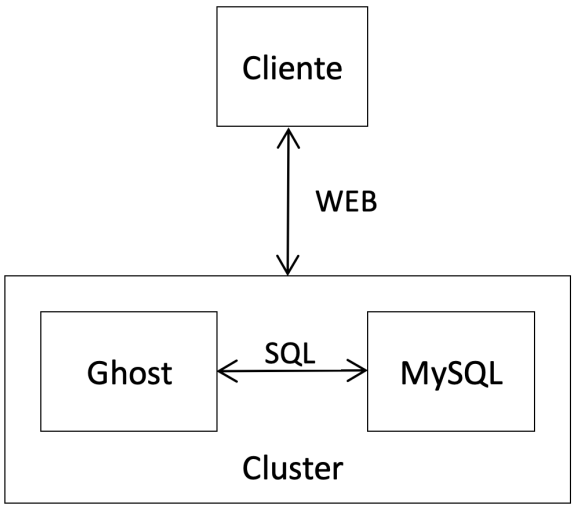

Figura 2.1: Arquitetura *Cliente-Servidor* da aplicação.

Capítulo 3

Ferramentas e abordagem utilizadas para a instalação e configuração automática da aplicação

De forma a proceder a uma instalação automática dos componentes da aplicação, foi usada a ferramente Ansible, sendo este um recurso que permite automatizar o deployment e escalamento de aplicações de uma maneira eficiente e simplificada.

Neste caso, a instalação automática começa pela criação do cluster, o qual é iniciados com 2 nodos, sendo que cada um destes é iniciado com uma imagem Ubuntu.

Status	Nome	Local	Número de nós	Total de vCPUs	Memória total
✓	ascn-cluster	us-central1-a	2	4	4 GB

Figura 3.1: Cluster

Status	Nome	Zona	Recomendações	Em uso por	IP interno
✓	gke-ascn-cluster-default-pool-20cbf9fa-13sl	us-central1-a		gke-ascn-cluster-default-pool-20cb...	10.128.0.20 (nic0)
✓	gke-ascn-cluster-default-pool-20cbf9fa-vjvw	us-central1-a		gke-ascn-cluster-default-pool-20cb...	10.128.0.21 (nic0)

Figura 3.2: Nodos

Finalmente, é possível prosseguir para o deployment da aplicação, na qual são criados 2 pods e 2 serviços: um para a aplicação **Ghost** e outro para a sua base de dados. Isto permite descentralizar a carga da aplicação, distribuindo-a por vários containers, obtendo assim um melhor balanceamento desta.

Nome	Status	Tipo	Pods	Namespace	Cluster
ghost-deployment	OK	Deployment	1/1	default	ascn-cluster
mysql-deployment	OK	Deployment	1/1	default	ascn-cluster

Figura 3.3: Cargas de trabalho (pods)

Nome	Status	Tipo	Pontos de extremidade	Pods	Namespace	Clusters
ghost-service	OK	Balanceador de carga externo	34.123.107.20:80	0/0	default	ascn-cluster
mysql-service	OK	IP do cluster	10.0.1.159	1/1	default	ascn-cluster

Figura 3.4: Serviços

Capítulo 4

Abordagem escolhida para o mecanismo de replicação

4.1 LoadBalancer

No que toca a acesso externo à nossa aplicação, foram considerados 3 serviços disponibilizados pela *Google Kubernetes Engine*:

- ClusterIP
- NodePort
- LoadBalancer

O ClusterIP é o serviço default do Kubernetes. Fornece um serviço dentro do cluster ao qual as outras aplicações dentro do mesmo couster conseguem aceder. O tráfego externo apenas é alcançado com o uso de um proxy.

O NodePort é a maneira mais primitiva de obter tráfego externo. Como o próprio nome indica, abre uma porta específica em cada nodo. Qualquer tráfego enviado para essa porta é posteriormente encaminhado para o serviço.

Neste caso, foi decidido usar o LoadBalancer. Na GKE, este ativará um Network Load Balancer que distribuirá o tráfego entre vários pods, aumentando assim a disponibilidade e o desempenho da aplicação. Além disso, permite adicionar ou remover facilmente pods do deployment, permitindo que a aplicação seja escalada de acordo com as necessidades. Esta funcionalidade é implementada no playbook *scale-ghost-app.yml*, que recebe um número de pods e que escala automaticamente o deployment até esse número.

```yaml
- name: Scale deployment
  hosts: localhost
  gather_facts: no
  vars_prompt:
    - name: "replica_count"
      prompt: "Enter the number of replicas:"
      private: no
  tasks:
    - name: Scale deployment
      command: kubectl scale deployment ghost-deployment --replicas={{ replica_count }}
```

Figura 4.1: scale-ghost-app.yml

4.2 Session Affinity

Além disso, foi implementado um serviço de Session Affinity. Session Affinity é uma técnica utilizada no LoadBalancer que permite manter uma conexão entre um cliente e um pod específico por um período de tempo determinado, neste caso, 10 minutos. Esta técnica permite uma maior otimização de cache e consistência de dados, aumentando assim a eficiência da aplicação. Neste caso, foi escolhida para conseguir uma persistência de sessão.

```yaml
spec:
  type: LoadBalancer # Makes the service accessible on a static port on each Node in the cluster.
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 600
```

Figura 4.2: Implementação de LoadBalancer e Session Affinity

Tentou-se também implementar um HorizontalPodAutoscaler, que escalaria a aplicação de acordo com as necessidades, que na definição utilizada tinham relação com o uso do CPU, mas um problema com a obtenção das métricas não o permitiu.

Capítulo 5

Monitorização, métricas e visualização

5.1 Ferramentas de monitorização

A monitorização de uma aplicação pode ser dividida em quatro etapas: observação, recolha, análise e apresentação. Para a supervisão deste aplicação foi usada a ferramenta Kubernetes Metrics Dashboard, disponibilizada pela própria Google Cloud. Esta dashboard é primeiramente instalada e acoplada aos nodos da aplicação. Finalmente, são instalados agentes de monitoramento no primeiro e segundo nós do cluster, permitindo coletar métricas de desempenho de cada nó.

```
- name: Install Metrics Dashboard
  shell : gcloud monitoring dashboards create --config-from-file=dashboard.json
  register: ghost_port_aux

- name: Get First Node
  shell : kubectl get nodes |  awk '{ print $1}' | grep -v NAME | sed -n '1p'
  register: firstNode

- name: Get Second Node
  shell : kubectl get nodes |  awk '{ print $1}' | grep -v NAME | sed -n '2p'
  register: secondNode


- name: Install Agent in first default Node
  shell : :> agents_to_install.csv && \echo '"projects/{{gcp_project}}/zones/{{gcp_zone}}/instances/{{firstNode.stdout}}","[{""type"":""ops-agent""}]"' >>
agents_to_install.csv && \curl -sSO https://dl.google.com/cloudagents/mass-provision-google-cloud-ops-agents.py && \python3 mass-provision-google-cloud-ops-
agents.py --file agents_to_install.csv

- name: Install Agent in second default Node
  shell : :> agents_to_install.csv && \echo '"projects/{{gcp_project}}/zones/{{gcp_zone}}/instances/{{secondNode.stdout}}","[{""type"":""ops-agent""}]"' >>
agents_to_install.csv && \curl -sSO https://dl.google.com/cloudagents/mass-provision-google-cloud-ops-agents.py && \python3 mass-provision-google-cloud-ops-
agents.py --file agents_to_install.csv
```

Figura 5.1: Playbook da instalação da Kubernetes Metrics Dashboard e respetivos agentes

5.2 Métricas escolhidas

Tendo em conta o problema e as métricas disponibilizadas pelo Google Monitoring, resolveu-se utilizar as seguintes métricas, sendo que as que se considerou mais pertinentes no problema atual:

- Tempo de utilização do CPU do MySQL e Ghost
- Utilização de memória pelos containers MySQL e Ghost
- Utilização do CPU para os nodos do cluster
- Utilização de memória para o Ghost
- Utilização de memória do container MySQL
- Bytes transmitidos do MySQL Pod
- Bytes recebidos do MySQL Pod
- Volume usado do MySQL Pod

Figura 5.2: Visualização geral da gestão de recursos

Capítulo 6

Ferramentas de avaliação e testes desenvolvidos

Com vista a poder efetuar testes de desempenho da implementação desenvolvida, foi necessário criar testes que permitissem avaliar os diferentes componentes e funcionalidade da aplicação, tendo em conta a possível variação de usuários que possam estar a utilizar. Para tal, recorreu-se à ferramenta JMeter, que permite efetuar testes de carga/stress através de um ficheiro de input, assim como simular um grande fluxo de utilizadores na aplicação.

Deste modo, recorreu-se à ferramenta BlazeMeter, que através da sua extensão Web permite gravar uma série de ações sobre a aplicação e, a partir das mesmas, gerar um ficheiro de input para o JMeter com o formato *.jmx*. Com recurso a esta ferramenta geraram-se os seguintes testes:

1. Teste de navegação na plataforma - Este teste tem como objetivo submeter a principal funcionalidade da aplicação a um teste de carga, sendo que a maioria dos usuários, teoricamente, pretende aceder para navegar entre páginas, ler artigos ou explorar a plataforma.
2. Teste de escrita de artigos - Este teste simula a escrita de um artigo por um utilizador, sendo que um utilizador acede à página, efetua login, escreve um artigo e publica.

Posteriormente, já com os ficheiros de teste obtidos, abriu-se os ficheiros no JMeter e iniciou-se os testes, sendo atribuídos diferentes valores de threads na aba *"Thread Group"*, permitindo assim submeter a plataforma a uma maior ou menor carga. É de reforçar avaliação experimental da aplicação foi combinada com as métricas de monitorização, permitindo assim extrair observações mais completas e incisivas sobre os resultados observados, como será demonstrado em seguida.

6.1 Demonstração e análise de resultados

6.1.1 Análise dos testes efetuados com auxilio do JMeter

De modo a ter uma percepção e análise dos recursos recrutados pelo Ghost, efetuaram-se alguns testes com os ficheiros acima referidos, pelo que se demonstra cada um deles e se faz uma breve explicação em seguida:

Valores de referência

De modo a ter uma base de comparação, mediu-se os recursos utilizados sem que a plataforma estivesse submetida a qualquer tipo de carga ou utilização, permitindo assim fazer comparações com os valores obtidos nos vários testes.

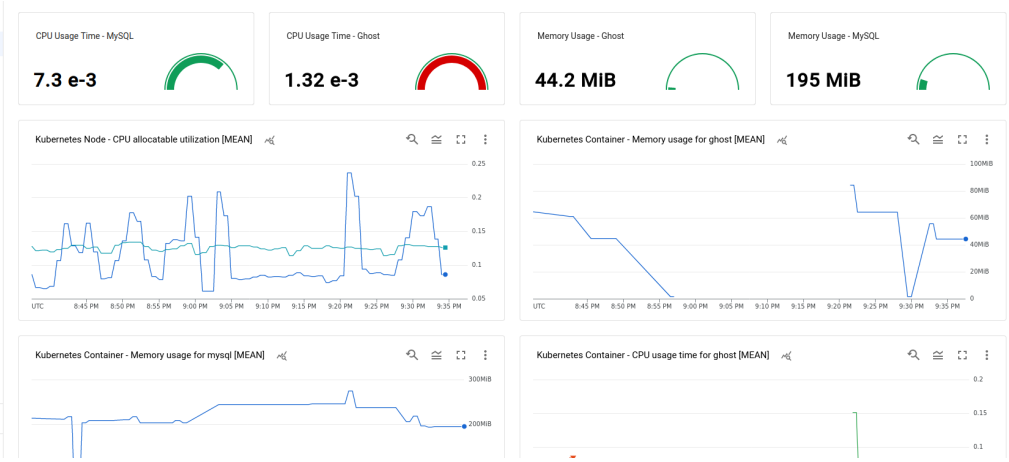

Figura 6.1: Teste sem qualquer utilização da plataforma

Teste de Navegação na plataforma com 100 threads

Com o primeiro teste, começou-se por 100 threads e obteve-se valores ligeiramente superiores aos valores de referência, demonstrando uma elevada capacidade para este valor.

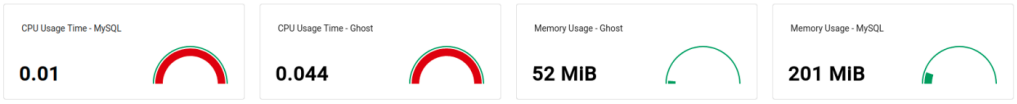

Figura 6.2: Valores obtidos no teste de navegação com 100 threads

Teste de Navegação na plataforma com 500 threads

Ao verificar a grande capacidade de lidar com as 100 threads, aumentou-se 5 vezes este valor, procurando analisar como escalam os valores com o aumento de threads. Deste modo, verificou-se um aumento em cerca de 3 a 4 vezes no CPU Usage Time de ambos, mas um aumento ligeiro na utilização de memória.

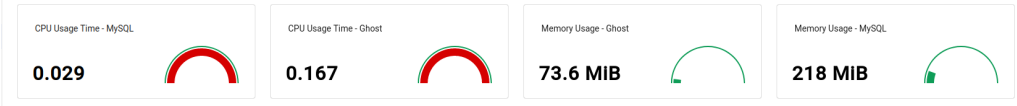

Figura 6.3: Valores obtidos no teste de navegação com 500 threads

Teste de Navegação na plataforma com 5000 threads

Por último, efetuou-se um teste extremo de threads, que foi difícil de ultrapassar estes valores de threads devido à falta de capacidade por parte dos computadores de teste.

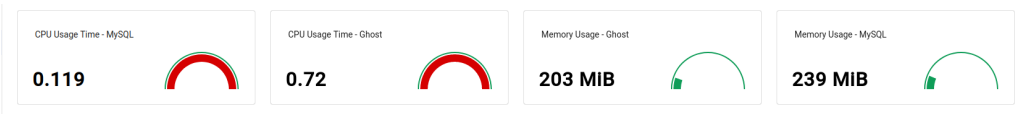

Figura 6.4: Valores obtidos no teste de navegação com 5000 threads

Teste de Login + Post com 10 threads

Passando para o segundo tipo de teste, começou-se com 10 threads, procurando sempre incrementar este valor e verificar as capacidades da plataforma.

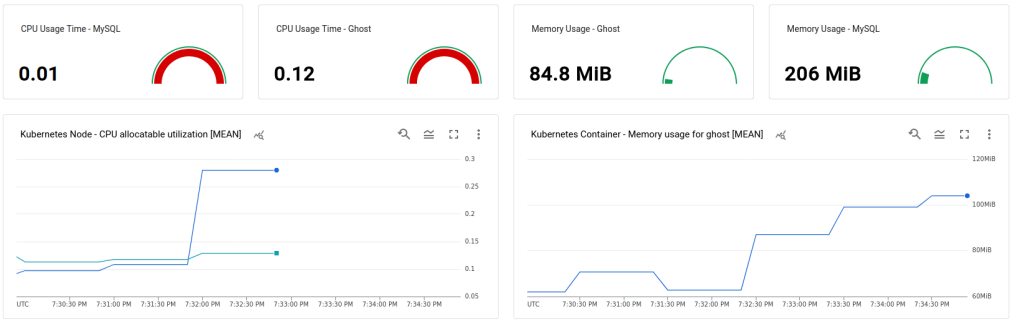

Figura 6.5: Valores obtidos no teste de login e publicação com 10 threads

De acordo com a figura acima, é possível verificar que se tem um incremento na utilização de CPU e Memória relativamente ao estado inicial. Foi também possível, através da análise dos gráficos de monitorização, verificar que há um incremento a nível geral.

Teste de Login + Post com 100 threads

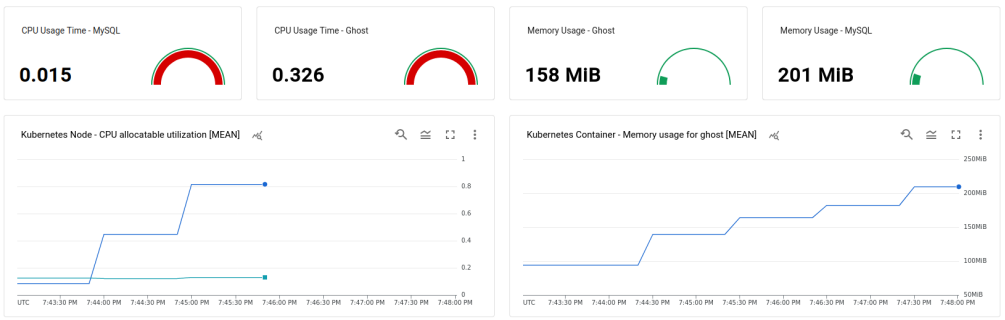

Figura 6.6: Valores obtidos no teste de login e publicação com 100 threads

No presente teste, aumentou-se o número de threads para 100, verificando-se que com um aumento de 10x no número de threads, se verifica um mínimo aumento no CPU do MySQL, mas um aumento de aproximadamente 3x no CPU do Ghost, o que se explica pela quantidade de posts efetuados.

Relativamente à memória, de igual modo se explica, sendo que é visível uma aumento mínimo no uso de memória do MySQL, mas um grande aumento no uso de memória do Ghost.

Análise final

Em suma, como é possível verificar pela generalidade dos resultados obtidos nos vários testes, o CPU Usage Time sofre muito mais utilização e variação, sendo a maior sobrecarga no Pod do Ghost.

6.1.2 Teste do Load Balancer e da Session Affinity

Para o teste da implementação do Load Balancer e do Session Affinity, criaram-se 2 réplicas do *ghost-app* e usaram-se 2 clientes, doravante C1 e C2. Utilizou-se um ciclo de modo a simular tráfego de alta intensidade do mesmo IP para o cluster. O C1 foi o primeiro a começar a enviar pedidos através do comando *wget*, a quem se seguiu o C2. Aquilo que observamos nos *logs* de cada um dos *pods* foi o esperado: O *pod 1* recebeu o tráfego do cliente *C1* e quando o cliente *C2* começou a enviar pedidos, foi encaminhado para o *pod 2* pelo *LoadBalancer*.

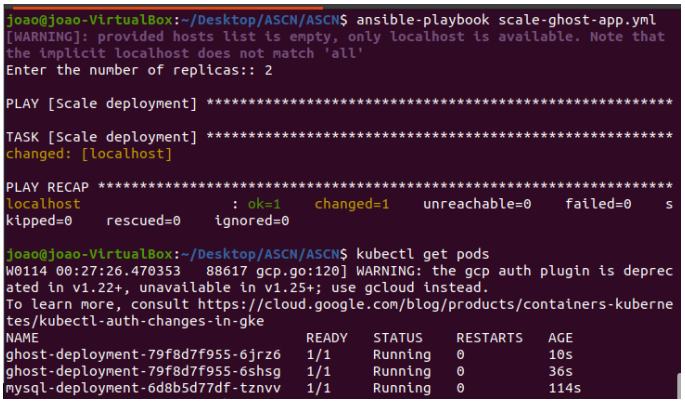

Figura 6.7: Playbook Ansible escala para os pods requisitados

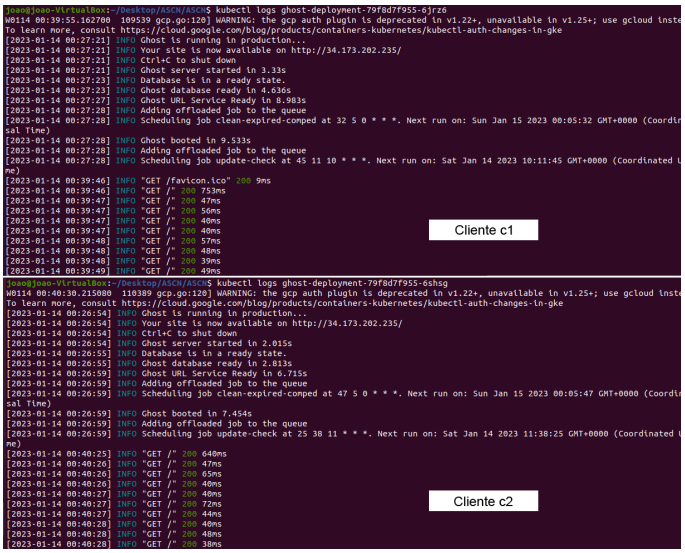

Figura 6.8: LoadBalancer encaminha cliente 1 para pod 1 e cliente 2 para pod 2

Após isso, para que a *Session Affinity* fosse testada, cada um dos clientes fez um pedido para a aplicação através do browser e mais uma vez, como esperado, o cliente *C1* foi encaminhado para o *pod 1* e o cliente *C2* foi encaminhado para o *pod 2*.

Figura 6.9: LoadBalancer encaminha cliente 1 para pod 1 e cliente 2 para pod 2 mesmo estando os dois pods sem qualquer tráfego

Capítulo 7

Conclusão

Após a conclusão deste trabalho prático, atribui-se um balanço positivo ao trabalho desenvolvido, já que permitiu praticar e consolidar os conceitos abordados tanto a nível teórico como prático.

Além disso, foi também possível constatar que os objetivos foram de uma forma geral alcançados, deixando ainda uma ideia na equipa de realização e cumprimento face às escolhas e decisões tomadas ao longo da realização da conceção do projeto.

A principal dificuldade encontrada foi a implementação do e-mail no botão subscribe da aplicação Ghost, problema que eventualmente foi resolvido. No entanto, houve um objetivo que não foi alcançado, sendo este a implementação de Horizontal Pod Autoscaling (HPA). Este permitiria escalar automaticamente com base em métricas de utilização de recursos, como CPU e memória.

Em suma, é importante novamente realçar a importância do projeto face à consolidação obtida dos conceitos e técnicas dados durante todo o semestre, ficando a certeza de que mais tarde, no mundo do trabalho, poder-se-á aplicar correta e devidamente quando assim for necessário.