UMinho

Mestrado Engenharia Informática
Requisitos e Arquiteturas de Software (2022/2023)

Grupo 8

PG50334, Diogo Manuel Brito Pires
PG50393, Gonçalo André Rodrigues Soares
PG50196, Ana Paula Oliveira Henriques
PG51242, Hugo André Sousa Gomes
PG50184, Ana Filipa Ribeiro Murta

Braga, 14 de janeiro de 2023

1 Introdução e objetivos

Neste trabalho prático procuramos automatizar ao máximo o processo de instalação, configuração, monitorização e avaliação da aplicação *Ghost*. Para isto, utilizamos a ferramenta *Ansible* que nos auxiliou neste processo de automatização. Assim, cumprimos os seguintes passos:

- Automatizar o processo de instalação da aplicação *Ghost*;
- Tornar o mais fácil possível a configuração da aplicação;
- Criar *dashboards* que permitam monitorizar a aplicação;
- Testar se a aplicação é acessível;
- Fazer testes de carga para avaliar a escalabilidade da aplicação;
- Tornar fácil escalar a aplicação;
- Tornar o uso dos *playbooks* seguro através do *ansible-vault*.

Para além disto, também definimos algumas boas práticas que iremos seguir:

- Facilitar a adição de novas *Tasks* aos *playbooks*;
- Usar *templates* do *ansible* para que a manipulação dos ficheiros seja tanto mais fácil de modificar como de expandir;
- Evitar o uso de soluções que não são sustentáveis quando a aplicação escala;
- Tornar os objetivos descritos em cima o mais automáticos possível (executando-os no menor número de passos manuais possíveis).

2 Arquitetura

No planeamento da arquitetura da aplicação *Ghost*, procuramos separar ao máximo as várias componentes da aplicação para que fosse mais fácil escalar o sistema no futuro. Dessa forma, decidimos executar cada componente da aplicação num *pod* distinto.

2.1 Setup do ambiente necessário

Antes de começarmos com a instalação da aplicação, é necessário garantir que os pré-requisitos de alguns módulos são instalados. Como tal, utilizámos o módulo *pip* para instalar as bibliotecas de *Python* necessárias. Para além disso, criámos um *namespace* para facilitar uma divisão de futuros componentes. Visto que, por agora, apenas temos dois componentes e estes encontram-se no mesmo *namespace*, decidimos colocar este *namespace* como o predefinido.

Também foi criada uma pasta temporária para a criação de ficheiros que possam ser necessários durante alguma etapa. Todas as tarefas deste processo de *setup* encontram-se no *role* ***setup***.

2.2 Instalação do container *MySQL*

Para a instalação da aplicação *Ghost*, é preciso instalar o *MySQL* para que este seja usada como base de dados em produção.

Primeiramente, começámos com a geração dos ficheiros de *deployment* do *MySQL*, a criação de um PVC (*PersistentVolumeClaim*) para garantirmos persistência de dados e criação do serviço do *MySQL*. Para facilitar a criação dinâmica destes ficheiros, usamos *templates* para passar dinamicamente vários parâmetros como o *namespace*, nome e *password* da base de dados.

Todas as tarefas deste processo de instalação do *container MySQL* encontram-se no *role* ***deploy_mysql***.

2.3 Instalação do container Ghost

Tal como na instalação do container *MySQL*, gerámos os ficheiros *deployment* e de criação do serviço.

Após isto, para a instalação deste componente, decidimos criar primeiro o serviço visto ser necessário o IP externo para o *deployment* do *Ghost*. De seguida, obtemos o IP externo do mesmo, atualizamos o ficheiro que armazena o IP da aplicação e fazemos o *deployment* do *Ghost* com a variável de ambiente *url* configurada como:

```
http://{{ ghost_ip }}:{{ ghost_port }}
```

Todas as tarefas deste processo de instalação do container *Ghost* encontram-se no *role* ***deploy_ghost***.

2.4 Undeploy da aplicação

Por fim, também implementámos o *undeploy* do *ghost*, que começa por apagar *pods*, *serviços*, e *deployments*. O próximo passo foi apagar os dados persistentes, no caso do parâmetro *delete_data* tomar o valor de *true*, seja usada.

Todas as tarefas deste processo encontram-se no *role* ***undeploy_ghost***.

3 Ferramentas utilizadas

Nesta secção, serão descritas as principais aplicações e módulos utilizados:

- **JMeter** - Utilizado para realizar testes à aplicação simulando vários clientes a realizar pedidos *http* ao mesmo tempo;

- **Módulo uri** - Utilizado no *ansible* para testar a conexão à aplicação;
- **Módulo k8s_info** - Utilizado no *ansible* para obter algumas informações, como o nome dos *pods* e *ip*'s externos;
- **Módulo k8s_exec** - Utilizado para executar comandos no *pod*, nomeadamente *queries* à base de dados;
- **Módulo de templates** - Utilizado para gerar dinamicamente ficheiros a partir de *templates* previamente definidos;
- **Módulos do gcp** - Necessário para o uso dos serviços do *Google Cloud Platform*;
- **Módulo do pip** - Necessário para instalar *Python packages* que serão utilizados noutras tarefas;
- **Módulo do shell** - Utilizado para correr comandos numa *shell*. Tentámos utilizar este módulo apenas quando não conseguíamos implementar as funcionalidades com outros mais adequados.

3.1 Configuração automática

Para facilitar a configuração automática, decidimos usar variáveis sempre que possível de modo a ser mais fácil de configurar a aplicação:

- **Variáveis da base de dados** (*group_vars*) - Nome da base de dados, utilizador e *password*;
- ***Namespace*** (*group_vars*) - *Namespace* utilizado atualmente, e usado em vários sítios, daí ser útil estar guardado aqui;
- **Configurações do mail** (*group_vars*) - Para facilitar uma possível troca de serviço de mail, basta trocar aqui e não é necessário ir ao ficheiro onde é usado;
- **Número de réplicas** (*group_vars*) - Ao armazenarmos aqui esta variável, o processo de efetuar uma futura troca do número de réplicas é facilitado, o que melhora a escalabilidade da aplicação;
- **Variáveis do GCP** (*inventory*) - Para facilitar a troca de parâmetros nas máquinas onde corre a aplicação. De notar que foi adicionada tambem a variavel do nome do cluster, permitindo assim que pudessem ser criados vários cluster de forma automática. Também armazenamos o ip externo da aplicação, bem como a porta podendo esta ser mudada;
- ***first_blog_admin*** (vars do *create_blog_admin*) - Assim torna-se mais fácil alterar o as variáveis do administrador do sistema.

4 Mecanismo de replicação

De modo a garantir um bom desempenho e resiliência, o sistema foi dotado de mecanismos de replicação, o que, para tal, foi escolhido o componente *Ghost*.

A definição do número de réplicas a serem criadas durante o *deploy* é definida no respetivo ficheiro de *deployment* do *Ghost* através da propriedade *replicas*, cujo valor é definido pela variável *n_replicas* definida em *group_vars/all.yml*, como mostra a Figura 1.

```
---
# Deployment of Ghost
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ghost-deployment
  namespace: {{ k8s_namespace }}
  labels:
    app: ghost
spec:
  replicas: {{ n_replicas }}
  selector:
    matchLabels:
      app: ghost
  ...
```

Figure 1: Snippet do ghost-deployment

Também poderá ser necessário escalar o componente do *Ghost* após este ter sido *deployed*. Para auxiliar essa tarefa, foi criada a *role scale_ghost*, que ao ser executada lê o número de réplicas, através do argumento *–extra-vars "replica_count=x"*, para o qual se pretende escalar o componente.

Um problema que pode surgir com a criação de várias réplicas do *Ghost* é que quando um utilizador fizer *login* numa réplica e, posteriormente, executar uma ação que seja redirecionada para outra réplica, este vai aparecer como não autenticado. Para corrigir isso foi utilizado o *Session Affinty* no serviço do *ghost*. Este serviço é do tipo *LoadBalancer*, o que permite fazer o balanço de carga entre as réplicas. O *Session Affinity* permite que o mesmo cliente faça pedidos ao mesmo *pod* com base no seu endereço IP, o que permite que o cliente mantenha a sessão válida entre pedidos. Adicionalmente, foi definido que, num máximo de 3h, este comportamento é verificado.

Figure 2: Definição de Session Affinity

5 Monitorização

Esta tarefa consistiu em monitorizar o comportamento da aplicação *Ghost* através da ferramenta de monitorização disponibilizada pela plataforma *Google Cloud*. Os testes foram realizados com dois nós, com um total de 4 CPUs e uma memória total de 4 GB.

5.1 Métricas

Deste modo, com o uso da ferramenta *Metric Explorer* foram obtidos gráficos para as diferentes métricas, nomeadamente, para o CPU, a utilização da memória, I/O e tempo de resposta, permitindo, assim, identificar possíveis pontos de falha e uma melhor gestão de recursos.

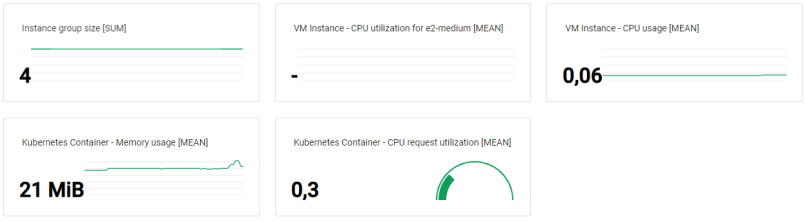
A seleção da métrica de monitorização do CPU deve-se ao facto de esta permitir medir a carga de trabalho do sistema. Deste modo, em caso de sobrecarga proporciona a identificação de processos que estão consumindo mais recursos do que o esperado, sendo depois necessário tomar medidas para otimizar o uso dos recursos.

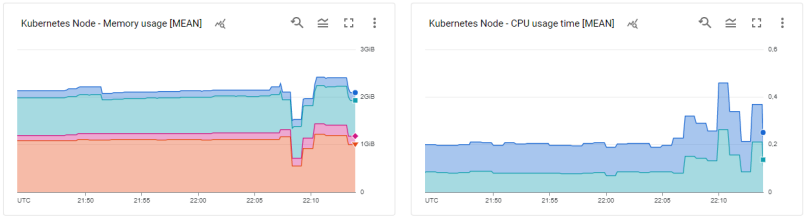
No caso da seleção da métrica de utilização de RAM, esta reflete indiretamente a eficácia de resposta do sistema. Ou seja, permite garantir que o sistema tem capacidade suficiente para executar os seus processos sem estes sofrerem por falta de recursos. Para além disto, esta permite otimizar o uso de memória através da identificação de processos que estão consumindo mais memória.

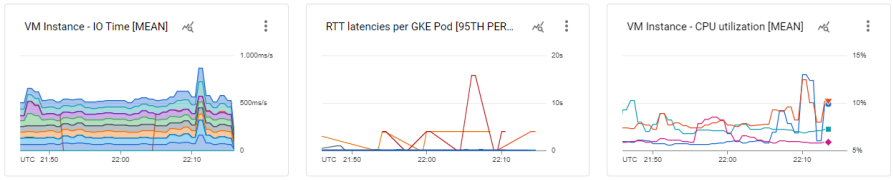
Por fim, temos que a utilização de I/O é uma boa métrica de monitorização, uma vez que esta permite medir a velocidade com que o sistema está a ler e a gravar os dados.

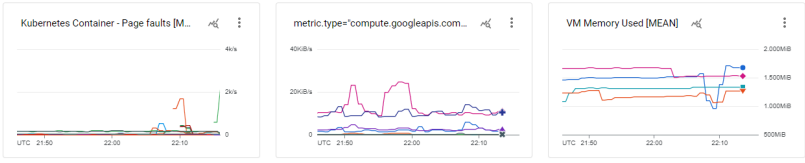
5.2 Visualização

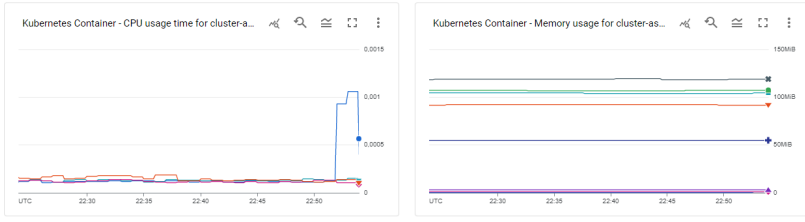
Nas seguintes Figuras, podemos analisar as métricas aplicadas ao projeto.


Instance group size [SUM]
4
VM Instance - CPU utilization for e2-medium [MEAN]
-
VM Instance - CPU usage [MEAN]
0,06
Kubernetes Container - Memory usage [MEAN]
21 MiB
Kubernetes Container - CPU request utilization [MEAN]
0,3


Kubernetes Node - Memory usage [MEAN]
Kubernetes Node - CPU usage time [MEAN]


VM Instance - IO Time [MEAN]
RTT latencies per GKE Pod [95TH PER...
VM Instance - CPU utilization [MEAN]


Kubernetes Container - Page faults [M...
metric.type="compute.googleapis.com...
VM Memory Used [MEAN]


Kubernetes Container - CPU usage time for cluster-a...
Kubernetes Container - Memory usage for cluster-as...

5.3 Análise de resultados

Seguidamente, fomos monitorizar a aplicação, através da observação da evolução e consumo do CPU durante o deploy e undeploy, tendo sido fornecido um pequeno intervalo de tempo entre ambos para permitir um cool-down.

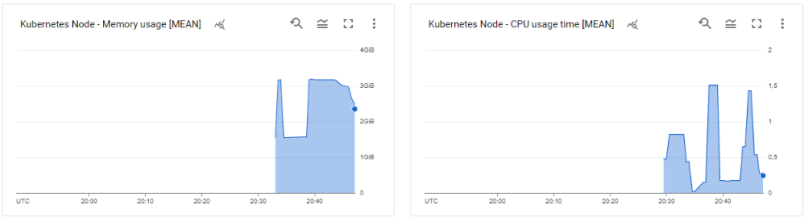


Figure 3: Depois do *deploy*

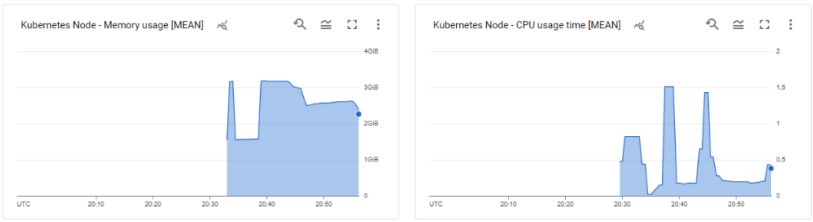


Figure 4: Depois do *undeploy*

Como podemos averiguar pelos Gráficos, durante o *cool-down*, denota-se um aumento ligeiro do consumo de memória pelo nodo, tendo este diminuído após *undeploy*. Relativamente ao CPU, verifica-se que este se manteve praticamente constante entre o decorrer do *deploy* e *undeploy*, tendo havido um aumento depois da execução de *undeploy*. No entanto, ainda é possível observar uma paragem deste aumento a partir de um certo ponto.

6 Avaliação

De seguida, procurámos analisar a disponibilidade do nosso sistema depois da sua instalação, bem como testes que verificassem o comportamento da aplicação quando utilizada por vários utilizadores, com testes de carga. Também testámos a persistência da memória, para que não se perdessem dados de utilizadores ao se efetuar *undeploy*.

6.1 Teste de persistência da memória

Para testarmos se os dados são corretamente mantidos e apagados conforme a vontade do administrador do sistema, criamos o *playbook* ***test-pvc***. Neste teste, alteramos temporariamente o *username* do admnistrador, e fazemos *undeploy* da aplicação. De seguida, fazemos *deploy* da aplicação outra vez e verificamos

se a alteração se manteve. Por fim, fazemos *undeploy* da aplicação ao apagar os dados e esperamos que a alteração temporária deixe de estar na aplicação, assim como o administrador volte a ter os dados corretos.

6.2 Testes de carga

Como previamente descrito, decidimos testar a resiliência do nosso *site* quando sujeito a uma grande quantidade de pedidos, o que, para esse fim, utilizamos a ferramenta *jmeter*. Por um lado, consideramos os testes realizados como importantes, pois permitem testar a escalabilidade da aplicação. Por outro lado, também consideramos relevante saber qual o limite de pedidos suportado pela aplicação, mantendo uma qualidade mínima de serviço.

Os testes de carga realizados foram com 10, 5000, 50000 e 100000 clientes, sendo que cada cliente é simulado por uma *thread*. Em cada teste, cada cliente acede à pagina principal da aplicação. Todavia, não tentamos simular mais ações dos clientes e, por isso, embora simplistas, estes testes são bastante apelativos à análise da escalabilidade que podemos fazer com eles.

6.3 Análise de resultados

Analisando os testes de carga, é possível observar na Figura 5 que o sistema consegue facilmente suportar até 100000 pedidos em simultâneo e é capaz de distribuir a carga por vários *pods* (neste caso entre 3 *pods*). É de salientar que devido à utilização de *Session Affinity* todos os pedidos feitos numa execução dos testes de carga vão ser redirecionados para o mesmo *pod*.

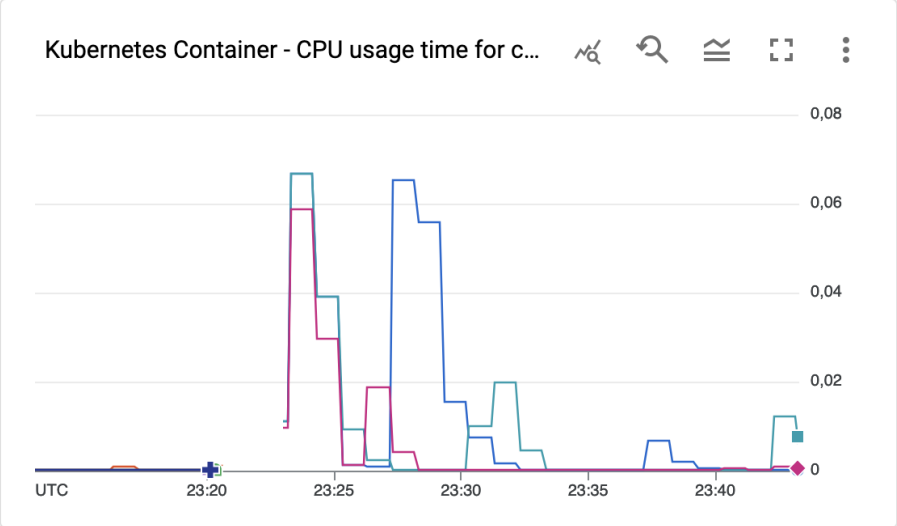


Figure 5: Utilização de CPU por pod nos testes de carga

Também podemos observar na Figura 6 que existem alguns picos no tempo de I/O e de latências RTT para cada pod sempre que a carga aumentou devido aos testes, o que era esperado.

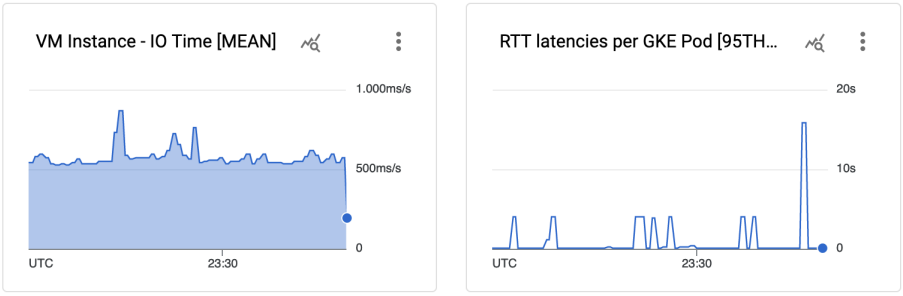


Figure 6: Tempo de I/O e latências RTT

7 Reflexão final

Por fim, consideramos importante refletir sobre o trabalho realizado, e analisar os pontos fortes e fracos da nossa solução.

7.1 Pontos fortes

- Conseguimos aceder à aplicação quando a metemos a correr, e isso é testado automaticamente;
- É possível monitorizar os principais recursos da aplicação;
- Privilegiámos a utilização dos módulos do *ansible*, em vez de comandos da shell;
- Utilizámos variáveis globais, e locais aos *roles*, para que quando seja necessário alterar algum valor seja mais fácil;
- A permanência dos dados funciona corretamente, como os testes o demonstram;
- Através de testes podemos monitorizar a resposta da nossa aplicação a diferentes quantidades de pedidos.

7.2 Pontos a melhorar

- Testar mais funcionalidades da aplicação, nomeadamente o login e a publicação de posts;
- Acrescentar mais medidas de segurança, como encriptação;

- Gerar posts automáticos para que tivéssemos mais possibilidades de testes automáticos;
- Analisar melhor o balanceamento da carga, entre diferentes pods.