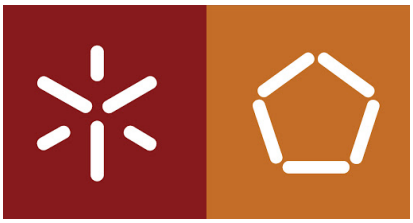

Universidade do Minho

Mestrado em Engenharia Informática

Aplicações e Serviços de Computação em Nuvem

Trabalho Prático

Automatizar o processo de instalação, configuração, monitorização e avaliação da aplicação *Ghost*

Grupo 17

Rodrigo Rodrigues (PG50726) Daniel Azevedo (PG50311)
Rui Monteiro (PG50739) Joana Oliveira (PG50460)
Bruno Alves (PG46798)

GitHub

Ano Letivo 2022/2023





Conteúdo

1 Introdução

Este trabalho prático surge no âmbito da Unidade Curricular de Aplicações e Serviços de Computação em Nuvem. Os seus objetivos passam por automatizar o processo de instalação, configuração, monitorização e avaliação da aplicação *Ghost*.

Ao longo deste relatório e após este capítulo introdutório, será feita uma descrição mais detalhada da aplicação *Ghost*, e explicado o processo de instalação e configuração automática da aplicação no serviço **Google Kubernetes Engine (GKE)** da *Google Cloud*, com recurso à ferramenta **Ansible**. No capítulo seguinte, é abordado o processo de monitorização da aplicação, uma vez que foi necessário realizar a monitorização da aplicação através da utilização de ferramentas que permitissem observar a instalação da aplicação *Ghost*. Por fim, no último capítulo, é feita uma análise aos resultados deste trabalho prático e uma reflexão final sobre os objetivos atingidos e sobre possíveis melhorias.

2 Arquitetura e componentes da aplicação

A aplicação *Ghost* é uma plataforma *open source* de publicação de *blogs* baseada em Node.js. O *Ghost* usa um design moderno e responsivo projetado para ser simples e fácil de usar. Os utilizadores da aplicação podem criar, editar e publicar conteúdo usando um editor integrado compatível com a formatação *markdown*.

O *Ghost* é altamente personalizável e pode ser estendido com uma variedade de temas gratuitos e pagos, bem como código personalizado. Adicionalmente, é também conhecido pelo seu ótimo desempenho e escalabilidade, tornando-o uma boa opção para *bloggers* e pequenas empresas.

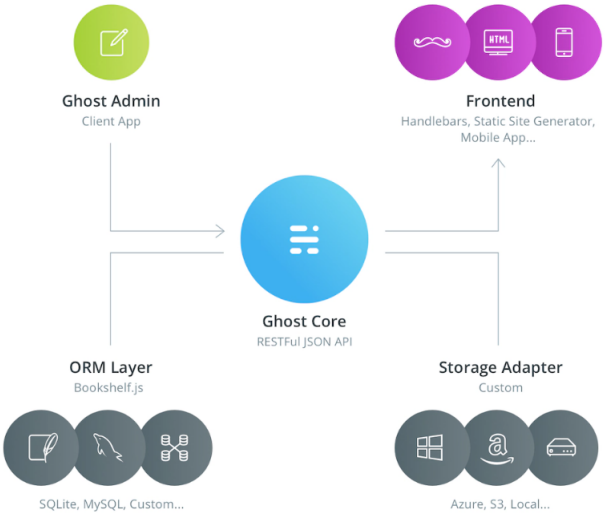

Figura 1: Arquitetura da aplicação *Ghost*

A arquitetura da aplicação *Ghost* segue essencialmente um padrão multi camada, e consiste em três camadas principais: a camada de *front-end* (interface para os utilizadores), a camada de *back-end* (serviço da aplicação) e finalmente uma camada responsável pela persistência dos dados.

A camada de *front-end* é responsável por exibir o conteúdo do *blog* ao utilizador final. Esta camada é desenvolvida com Handlebars.js, um motor de *templates* JavaScript que permite criar páginas web dinâmicas.

A camada de *back-end* é responsável por gerir os pedidos dos utilizadores, processar as informações e fornecer as respostas adequadas. Esta camada é desenvolvida com Node.js e Express.js, um *framework* web para Node.js que facilita a criação de aplicações web.

A camada de base de dados é responsável por armazenar e gerir os dados do blog em MySQL, por exemplo.

Além dessas camadas, o *Ghost* também utiliza o Nginx como servidor web e o *Ghost-CLI* como ferramenta de gestão. O Nginx é utilizado para lidar com as solicitações HTTP e HTTPS, enquanto o *Ghost-CLI* é utilizado para instalar e gerir a aplicação *Ghost*.

3 Instalação e configuração

Para automatizar a instalação da aplicação com as ferramentas referidas foram utilizados diversos ficheiros yaml. Além disso, podemos, em primeiro lugar, destacar a pasta *roles*. No Ansible, as *roles* são uma forma de organizar e estruturar playbooks e tarefas. Elas permitem agrupar tarefas, arquivos e variáveis de maneira lógica e reutilizável. As *roles* podem ser partilhadas e reutilizadas em vários *playbooks*, o que ajuda a reduzir a duplicação de código no Ansible. Assim sendo, temos as seguintes *roles*:

- **gke_cluster_create:** criação de um cluster na Google Cloud
- **gke_cluster_destroy:** destruição de um cluster na Google Cloud
- **create_namespace:** criação do *namespace*
- **create_secret:** criação de um *vault* do ansible para guardar passwords e variáveis sensíveis
- **ghost:** realização de tarefas para fazer *deployment* do Ghost num cluster Kubernetes com configurações específicas, usando módulos GCP e kubernetes.
  - Primeiro, criar um *GCP compute address* com o nome "ghost-address"na região "us-central1", usando o "gcp_project"e o "gcp_cred_file"especificados para autenticação.
  - Depois, usa o endereço criado acima para definir as variáveis "ghost_ip"e "ghost_port"para serem usadas mais tarde.
  - Depois, cria um serviço Kubernetes "ghost-blog"com load balancer IP definido para o endereço criado acima e selector app: ghost-blog, portas com protocolo TCP, porta 80, *targetPort* definida para a porta *default ghost port*.
  - De seguida, cria um volume de armazenamento persistente Kubernetes com o nome "ghost-pvclaim"definido para o *namespace* especificado para o Ghost, com modo de acesso *ReadWriteOnce* e armazenamento de 10Gi.

  - Depois disso, faz *deploy* do blog Ghost usando um *deployment* Kubernetes com o nome "ghost-blog", no *namespace* pretendido, e com o número de réplicas pretendido. O *deployment* usa a versão ghost:5.14.1, as portas do *container* são definidas para a porta *default* do ghost, e são definidas varíaveis de ambiente (*env*) para o *container*.
  - Finalmente, faz-se uma procura por todos os *pods* do mysql usando o módulo *kubernetes_info* e regista-se o resultado para uma variável, que será usada na *task* relativa à conta do admin.

- **mysql:** realização do *deployment* de um serviço mysql num cluster kubernetes e conectá-lo ao Ghost, com configurações específicas, usando módulos kubernetes.
  - Primeiro, cria-se um serviço Kubernetes chamado "ghost-mysql"no namespace especificado, na porta 3306, e o *selector* é definido para a app: ghost.
  - Depois, cria-se um volume de armazenamento persistente Kubernetes chamado "mysql-pvclaim", como modo de acesso *ReadWriteOnce* e capacidade de armazenamento de 10Gi.
  - De seguida, faz-se *deploy* de MySQL usando um *deployment* kubernetes com o nome "ghost-mysql", e o selector definido para fazer *match* com os *labels* app: ghost e tier: mysql, usando uma estratégia de *recreate*. Nota: A estratégia "recreate"primeiro exclui todos os pods existentes antes de criar novos. Existe um *trade-off* pois isto garante que todos os pods estejam a executar a versão mais recente da aplicação, mas também significa que todos os pods em execução serão encerrados de uma vez, o que pode causar alguma interrupção do serviço.
  - Finalmente, cria-se um *volume mount* que mapeia a diretoria do container /var/lib/mysql para a *claim* do volume persistente "mysql-pvclaim".

- **test_ghost:** testar o funcionamento do *deployment* do ghost
- **undeploy-ghost:** fazer *undeploy* do ghost

3.1 Requisitos

A primeira tarefa deste trabalho prático consiste em utilizar a ferramenta *Ansible* para automatizar a instalação e configuração da aplicação *Ghost* no serviço *Google* ***Kubernetes Engine*** *(GKE)* da Google *Cloud*. Assim sendo, foi elaborada a seguinte lista de requisitos para a realização desta tarefa:

- Utilização de Ansible
- Utilização do serviço Kubernetes Engine
- Execução dos diferentes componentes da aplicação *Ghost* em pods distintos
- Automatização da instalação e configuração da aplicação *Ghost* no menor número de passos manuais possíveis
- Proteção de dados críticos da aplicação em caso de paragem
- Respeitar a estrutura e funcionalidades dos *playbooks* Ansible disponibilizados

3.2 Descrição das ferramentas utilizadas

Efetuar o provisionamento e *deployment* da aplicação *Ghost* passo a passo, de forma manual, pode ser um processo repetitivo e moroso pelo que surgiu a necessidade de recorrer à ferramenta *Ansible*. O *Ansible* é uma ferramenta que permite configurar e fazer o *deploy* de aplicações nos ambientes de desenvolvimento, de testes e de produção, feito a partir de uma máquina. Esta ferramenta disponibiliza componentes úteis como o *inventory*, *tasks*, *module*, *role* ou o *playbook* onde se definem as *tasks* a aplicar em *targets* específicos.

Tirou-se partido de serviços disponibilizados pela plataforma do *Google Cloud Platform*, tal como o *Google Kubernetes Engine*. O GKE oferece um ambiente pronto para fazer *deployment*, gerir e escalar aplicações com *containers*, usando a infraestrutura do Google. Este ambiente consiste em várias máquinas agrupadas que formam

um *cluster* (que pode ser gerido através de ferramentas do GKE). Adicionalmente, o GKE permite a distribuição de carga automática pelos *pods* (*load balancing*).

4 Mecanismo de replicação

No contexto de um *deployment* em Kubernetes, a replicação refere-se ao processo de criação e gestão de várias cópias (ou réplicas) de um *pod*, para garantir alta disponibilidade e escalabilidade da aplicação. A replicação garante que, se um *pod* falhar, outra réplica estará disponível para substituí-lo, minimizando o tempo de inatividade e garantindo que a aplicação esteja sempre disponível para os utilizadores.

A estratégia de replicação usada é a estratégia de replicação padrão no *Kubernetes* chamada *Rolling Update*. O campo **replicas** na secção de especificação do *deployment* do *Ghost* é definido como "{{ n_replicas }}", o que significa que o número de réplicas é determinado pelo valor da variável *n_replicas*. O campo **selector** na secção de especificações também é definido para fazer match com *labels* com o nome da aplicação *ghost-blog*. Assim, o *deployment* irá criar e gerir *pods* com esse *label* e vai atualizar os *pods* um de cada vez para garantir alta disponibilidade da aplicação.

A estratégia de replicação *Rolling Update* permite atualizar a aplicação com o mínimo de tempo de inatividade, pois atualiza os *pods* um de cada vez. Isso significa que pelo menos uma réplica da aplicação está sempre disponível durante o processo de atualização. Por ser a estratégia *default* em *Kubernetes*, oferece facilidade de uso e requer mínima configuração. Para além disso, oferece flexibilidade uma vez que podemos controlar o ritmo das atualizações ajustando o número de réplicas atualizadas de cada vez e o atraso entre as atualizações. As atualizações contínuas reduzem o risco de erros ou falhas atualizando apenas um pequeno número de *pods* de cada vez. Assim, é possível reverter facilmente para uma versão anterior da aplicação se houver problemas com a atualização.

5 Monitorização

Ferramentas de monitorização de software são programas utilizados para analisar o desempenho e o uso de uma aplicação. Elas permitem aos administradores monitorizar a saúde e a disponibilidade de um sistema, bem como detetar problemas, de modo a poderem solucioná-los rapidamente.

Para realizar esta tarefa, foi utilizado um ferramenta de monitorização disponibilizada pela plataforma *Google Cloud*, nomeadamente o *Cloud Monitoring*.

O *Cloud Monitoring* permite aos utilizadores monitorizar a disponibilidade, o desempenho e o uso de recursos como máquinas virtuais, instâncias de *container* e serviços de nuvem, fornecendo dados em tempo real e históricos para ajudar o utilizador a entender como seus recursos computacionais estão a ser usados.

Foi construído um painel personalizado para monitorizar o *cluster Kubernetes*, dividido em dois *widgets*, exibidos como gráficos de linhas. Cada widget fornece uma visão detalhada do uso da memória da aplicação durante um período de tempo. Os dados são coletados da API do *Kubernetes* e filtrados para mostrar o uso de memória do *container* para o *Ghost* e para o MySQL.

No desenvolvimento desta tarefa, escolhemos visualizar a memória usada na base de dados e também a memória usada pela aplicação *Ghost*.

A memória de uma aplicação é um dos recursos mais importantes pois aplicações como o *Ghost* precisam de armazenar e acessar grandes quantidades de dados, o desempenho da memória também interfere de forma significativa na escalabilidade da aplicação. Se a memória não for suficiente ou não tiver o desempenho desejado, a aplicação pode se tornar lenta e instável.

6 Conclusão e Reflexão final

Dado por concluído o trabalho prático é importante fazer uma reflexão crítica sobre o mesmo, tendo em conta os aspetos positivos e negativos.

É possível considerar como um ponto positivo o facto do processo de *deployment* da aplicação *Ghost* estar completamente funcional. Além disso, é ainda utilizado um mecanismo de replicação que permite garantir a disponibilidade da aplicação e um *load balancer*, que permite o balanceamento da carga.

Por outro lado, temos os pontos negativos ou os pontos que gostaríamos de ter implementado. Nomeadamente, a realização da tarefa 2, relativa à avaliação experimental.