Universidade do Minho
Escola de Engenharia

Arquitectura de Sistemas de Computação em Nuvem

João Pedro da Santa Guedes PG47329
Luís Pedro Oliveira de Castro Vieira PG47430
Carlos Miguel Luzia Carvalho PG47092
Bárbara Ferreira Teixeira PG47038
Maria Beatriz Araujo Lacerda A89535

PG47329 PG47430 PG47092 PG47038 A89535

11 de janeiro de 2022

Índice

1 Introdução

Este trabalho prático tem como objetivo principal a caracterização, análise, instalação, monitorização e avaliação da aplicação Wiki.js (https://js.wiki). O pretendido é aplicar os conceitos lecionados na UC de Arquitectura de Sistemas de Computação em Nuvem de forma a garantir a automatização completa do processo de instalação, monitorização e avaliação da aplicação.

Este projeto foi nos então apresentado em 4 fases:

- Numa primeira fase, compreender vários pontos como qual a arquitectura e componentes da aplicação, como também a identificação dos padrões de distribuição da aplicação e dos diferentes componentes, e ainda a identificação de componentes e operações cujo desempenho é critico.
- Numa segunda fase, definir as ferramentas mais apropriadas, a nosso entender, que permitam a instalação automática e configurável dos diversos componentes da aplicação num ambiente de computação em nuvem, no caso em especifico a Google Cloud Plataform.
- Numa terceira fase, instalar ferramentas de monitorização que permitam observar a aplicação num ambiente de testes e/ou produção.
- Por fim, numa quarta fase, instalar ferramentas de avaliação da aplicação com os seguintes objetivos:
  - Avaliar a aplicação em causa de forma realista e com múltiplos testes que simulem diferentes comportamentos de utilizadores.
  - Avaliar métricas de monitorização para extrair observações mais completas sobre os resultados observados.
  - Executar os testes definidos de forma automática e configurável.

2 Arquitectura e Componentes da Aplicação

2.1 Arquitetura Multi-tier

Para conseguir cumprir o objetivo a que nos propusemos, realizamos o nosso trabalho sobre uma arquitetura Multi-tier, onde cada servidor actua como cliente do nível seguinte, esta permite a implementação independente e o escalonamento de diferentes funcionalidades.

Ou seja temos então uma das VMs com a parte da WEB no caso específico do projeto, Wiki.js, uma outra fica com a parte relativa à base de dados e uma terceira VM sendo o Master, que serve para correr o playbook e dar deploy ao Wiki.js e à Base de Dados. Além disso vai ser responsável por receber dados do Metricbeat provenientes dos workers (sendo estes as outras duas VMs).

2.2 Identificação dos Padrões de Distribuição e Componentes

Como componentes desta arquitetura temos como já referido em cima o Wiki.js e a Base de dados (postgres). Em cada VM, é feito o pull da respetiva imagem (numa é feito o download da imagem do wikijs e na outra do postgres) e, posteriormente, é feito o deploy através do kubernetes instalado pelo master através do playbook.

2.3 Operações Críticas

Começamos por instalar o ***Docker***, pelo facto de esta ferramenta ser uma dependência do ***Kubernets*** e, de seguida, instalamos o ***Kubernets***. No Master, é dado init ao ***Kubelet*** para que posteriormente seja possível aplicarmos a config da pod network (kube-fannel), lecionada nas aulas práticas. Depois, através do Master, é dado o deploy do ***Postgres*** e da ***Wiki.js*** para as outras VMs através da aplicação dos config files respetivos.

3 Ferramentas e Abordagem utilizada para a configuração Automática da Aplicação

3.1 Estrutura Adoptada

Com o intuito de automatizar a instalação dos diversos componentes da aplicação num ambiente de computação em nuvem, nomeadamente, na *Google Cloud Platform* (GCP), o grupo procedeu a uma análise dos recursos que seriam necessários para que não ocorressem falhas durante o mesmo.

Assim, decidimos criar três máquinas virtuais da série **e2-medium** (2 vCPU, 4GB RAM). Para perceber melhor a estrutura desenvolvida apresentamos o seguinte esquema:

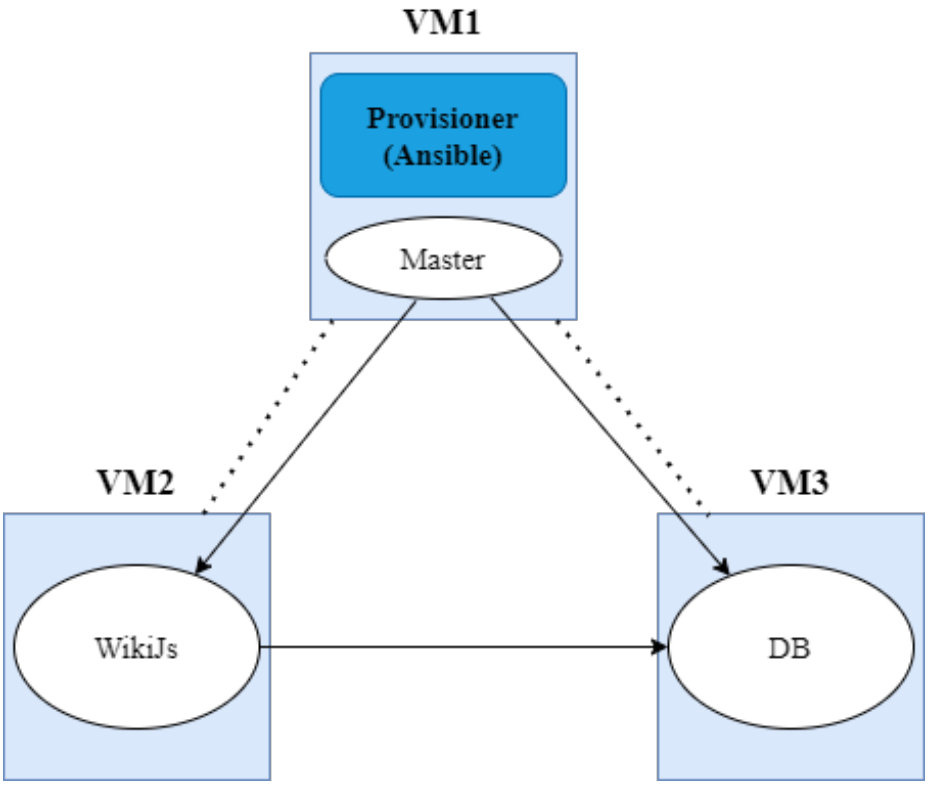


Figure 1: Esquema da Estrutura

	Status	Nome ↑	Zona	Recomendações	Em uso por	IP interno	IP externo	Conectar	
☐	●	master	europe-central2-a			10.186.0.40 (nic0)	Nenhum	SSH	⋮
☐	●	w1	europe-central2-a			10.186.0.41 (nic0)	Nenhum	SSH	⋮
☐	●	w2	europe-central2-a			10.186.0.42 (nic0)	Nenhum	SSH	⋮

Figure 2: Máquinas Virtuais criadas na GCP

Como podemos entender pelo esquema, cada uma destas máquinas virtuais terá um papel diferente na instalação e funcionamento da aplicação:

- **Master**
  Máquina Virtual onde será corrido o playbook responsável por toda a automatização do processo de instalação (Provisioner), bem como a responsável pelo deployment e controlo dos nodes e pods do Kubernetes.

- **W1** e **W2**
  Máquinas Virtuais onde estarão presentes os nodes necessários à implementação da base de dados e da aplicação.

3.2 Ferramenta Escolhida

Para este trabalho, o grupo decidiu usar o ***Ansible***, pois era a ferramenta de deploying mencionada e utilizada nas aulas, escolhendo também o uso do ***Kubernetes*** para deployment dos pods responsáveis por correr o serviço web e a base de dados.
O projeto construído encontra-se dividido da seguinte forma:

- **group_vars**: Contém referências aos IPs das máquinas virtuais para uso nas tasks
- **roles**: Contém as diferentes roles da instalação, especificadas a seguir
- **postgres_files**: Contém o ficheiro de deploy da base de dados
- **wikijs_files**: Contém o ficheiro de deploy da aplicação
- **hosts.inv**
- **playbook.yml**: Playbook principal pela automatização da instalação

Passamos a explicar as responsabilidades de cada role:

- **common**: role responsável por instalar todos os packages necessários para a instalação do docker e do kubernetes
- **master**: role corrida apenas no master e responsável por dar init ao kubelet, instalar a *Pod network* e guardar os *join commands* usados nos workers
- **worker**: corrida apenas nos workers onde é copiado e usado o código para juntar o respetivo nodo ao cluster
- **db**: role onde é dado apply aos ficheiros .yml relativos ao postgres, dando deploy do respetivo pod
- **app**: role onde é dado apply aos ficheiros .yml relativos à wikijs, dando deploy do respetivo pod
- **elastic**: para efeitos de controlo e gestão, esta é a role onde é instalado o Elasticsearch e o Kibana no master, para fazer uso do Metricbeat que irá ser instalado posteriormente nos worker nodes
- **beats**: role onde é instalado o Metricbeat nos worker nodes

3.3 Abordagem

A instalação da ***Wikijs*** e da ***Postgres*** em diferentes containers tem como objetivo aproveitar as vantagens da ultilização de uma arquitetura *Multi-tier*. Desta forma, podemos garantir que se um destes elementos for abaixo, o cluster continua em funcionamento. A segurança também é melhorada separando os dados do código. E torna-se fácil aumentar a capacidade da mesma porque basta acrescentar nodos ao cluster.

Considerando as várias ferramentas disponibilizadas pela Google Cloud, como bases de dados e ferramentas de monitorização e métricas, que podíamos ter aproveitado e que não chegamos a tirar proveito de, apenas consideramos que não estávamos familiarizados com estas e decidimos reaproveitar as ferramentas mencionadas na UC.

Por fim, acreditamos ser importante mencionar que tanto a Base de Dados como a Wikijs podem ser replicadas, alterando o número de réplicas no config do deployment do Wikijs e correndo um outro processo de instalação que faça o deployment do pod. Infelizmente, não aplicámos nenhuma réplica no nosso trabalho, mesmo conhecendo as vantagens da utilização de replicação tais como melhorar a fiabilidade, tolerância a falhas, e acessibilidade.

4 Ferramentas de Monitorização e Métricas

Como ferramenta de monitorização utilizamos a stack ***ELK***. Esta escolha adviu-se do facto de ser este o conjunto de ferramentas utilizado durante a realização das aulas práticas da unidade curricular, sendo portanto algo com o qual já estavamos familiarizados.

As ferramentas utilizadas são as que abaixo se encontram descritas, as quais permitem, em conjunto, obter resultados completos, organizados e de fácil acesso, o que proporciona uma monitorização em tempo real da aplicação completa.

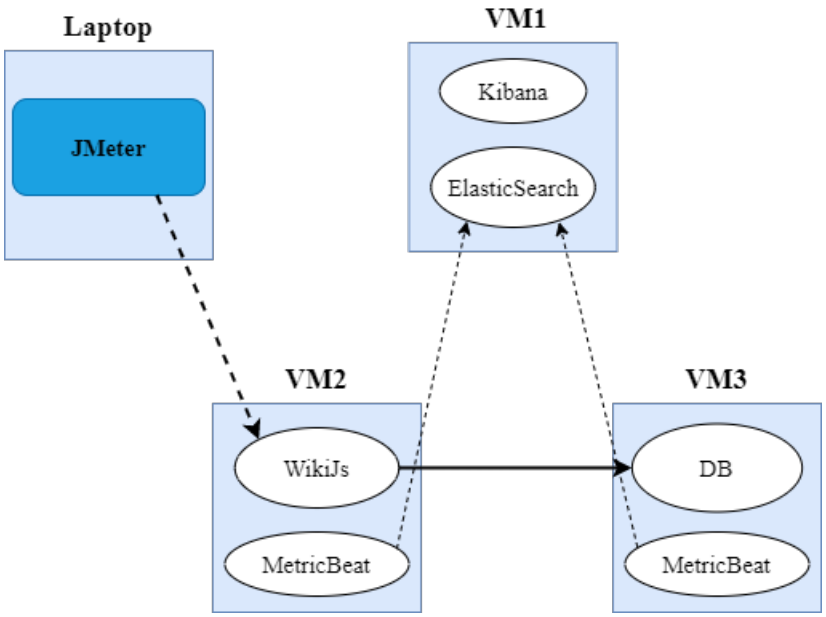


Figure 3: JMeter e Monotorização

- **Elasticsearch**
  Os dados recolhidos fluem para o Elasticsearch. Estes mesmos dados serão analisados, normalizados e, consequentemente, indexados. O Elasticsearch foi instalado e configurado numa VM.
- **Kibana**
  Ferramenta de visualização de dados através da qual podemos visualizar, em tempo real, diagramas e gráficos no ElasticSearch.
- **Metricbeat**
  Shipper leve, instalado nas máquinas virtuais, com o objetivo de enviar periodicamente métricas sobre os serviços que estão a correr. A ferramenta, depois de coletar esta informação, envia para o Elasticsearch, permitindo posteriormente a sua visualização no Kibana.

Esta ferramenta permitiu-nos analisar e explorar as métricas escolhidas para verificar a eficácia do nosso trabalho. Como métricas escolhemos *CPU Usage* e *Memory Usage* que nos permitem analisar os recursos computacionais utilizados pela aplicação.

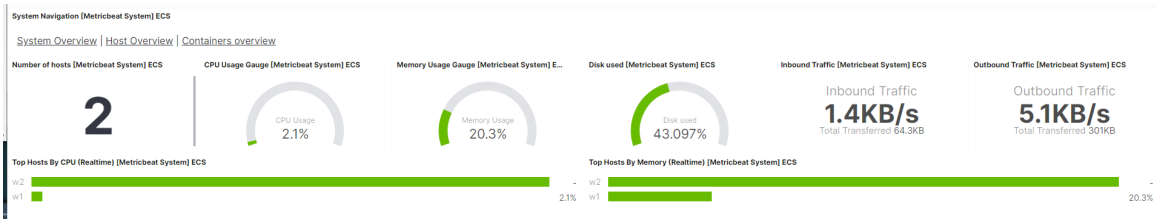


Figure 4: Visão Geral dos Recursos

Fazendo desde já uma pequena análise de resultados, podemos observar que a aplicação não é exigente em grande escala a nível geral, uma vez que se encontra a consumir apenas cerca de 2% da capacidade do CPU e cerca de 20% da capacidade da memória RAM.

No entanto é importante mencionar que aquando da obtenção destes resultados as máquinas mão se encontravam em "esforço", isto é, não se encontravam processos a correr que pudessem afetar a sua performance.

5 Ferramentas de Avaliação e Testes

Tal como na monitorização, decidimos escolher uma ferramenta com a qual estivéssemos familiarizados. Por este motivo, a ferramenta escolhida para realizar os teste foi o ***Apache JMeter***.

De forma a melhor percebermos como funciona os testes apresentaremos um diagrama que apresenta a relação entre a ferramenta de avaliação e as ferramentas de monitorização.

Para testarmos a sobrecarga da aplicação realizámos testes com 20, 500 e 1000 threads (com o intuito de simular utilizadores) a realizarem GET requests, a correr num loop infinito.

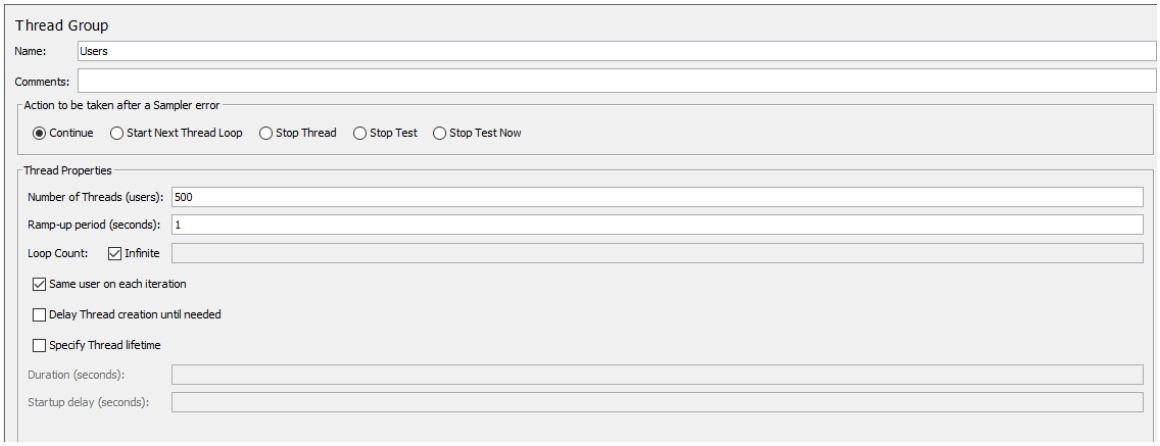


Figure 5: Exemplo de Teste Utilizado

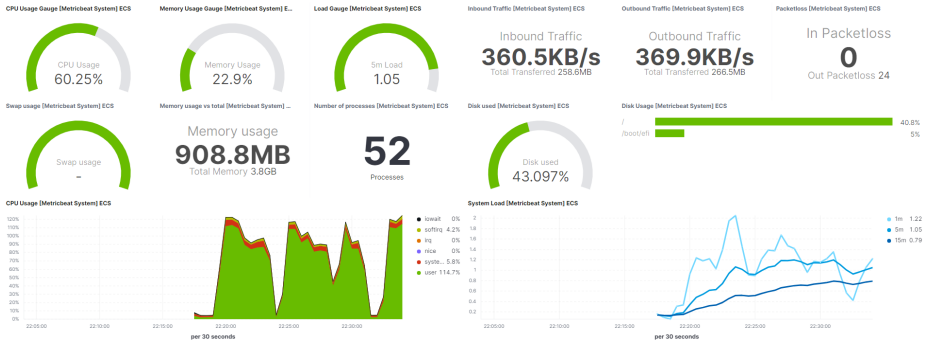
5.1 Apresentação e Análise de Resultados

De seguida apresentaremos os resultados obtidos, visualizados no dashboard do Kibana, e faremos uma breve discussão dos mesmos.

5.1.1 Threads = 20

5.1.2 Threads = 500

5.1.3 Threads = 1000



5.1.4 Análise de Resultados

Como podemos comprovar pelas imagens acima, e como seria expectável, os valores de recursos foram aumentando de teste para teste, tendo-se notado sobretudo uma maior carga a nível do CPU. Quanto à memória RAM utilizada, manteve-se basicamente na mesma gama de valores, 21-23%. No entanto estávamos à espera que os valores de CPU fossem mais díspares entre o diferente número de threads.

Todavia, esta série de teste realizados permitiu ao grupo perceber que a dada altura a aplicação iria começar a ser sobrecarregada o que poderia levar à falha da mesma e, consequentemente, o não cumprimento eficiente das suas funcionalidades.

6 Conclusão

Este trabalho prático teve como objetivo principal a caracterização, análise, instalação, monitorização e avaliação da aplicação Wiki.js (https://js.wiki). O pretendido era então aplicar os conceitos lecionados na UC de Arquitectura de Sistemas de Computação em Nuvem de forma a garantir a automatização completa do processo de instalação, monitorização e avaliação da mesma.

Tendo dado por concluído o nosso trabalho conseguimos alcançar a grande maioria dos objectivos propostos pelos docentes para este projeto, sendo estes: a identificação dos padrões de distribuição da aplicação e operações cujo desempenho é crítico, a utilização de ferramentas apropriadas para a instalação automática e configurável dos diversos componentes, a instalação de ferramentas de monitorização e por fim a instalação de ferramentas de avaliação.

Apesar de tudo isto, e atendendo ao facto de como explicitado acima termos atingido praticamente todos os objetivos, existem certas melhorias ou anotações que temos em relação ao trabalho desenvolvido. Uma delas, com muito pesar nosso, o facto de não termos conseguido realizar a instalação do Metriceat nas VMs workers. Este foi um foi um problema em que investimos algum tempo a tentar solucionar, porém não tivemos sucesso.

Relativamente a possíveis melhorias ou funcionalidades que gostávamos de ter conseguido implementar no trabalho, seriam coisas como o deploy automático das VMs na Google Cloud Platform, o uso de um maior número de módulos do Ansible, ao invés de recorrer apenas a comandos shell, e ainda explorar outras métricas de monitorização e avaliação.

Para concluir, o grupo considera que mesmo existindo espaço para melhorias, conseguimos realizar um bom trabalho, cumprindo os objetivos a que nos propusemos.