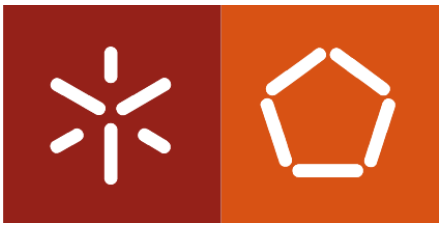

UNIVERSIDADE DO MINHO

SYSTEM DEPLOYMENT AND BENCHMARKING

OPENPROJECT

Grupo: 8
João Nunes - A82300
Joel Gama - A82202
Tiago Pinheiro - A82491
Vítor Gomes - A75362

Braga, Portugal
29 de Dezembro de 2019

Conteúdo

1 Introdução

No âmbito da UC de *Systems Deployment and Benchmarking* foi proposta a análise e caraterização de uma aplicação distribuída *opensource*. Como era necessário escolher uma aplicação *multi-tier*, com elasticidade em pelo menos uma das camadas e com um sistema de gestão de base de dados, foi escolhido portanto o *OpenProject*.

O *OpenProject* distingue-se como sendo um sistema de gestão de projetos. Permitindo que as equipas de desenvolvimento se coordenem de uma forma organizada, através deste software.A *app* permite definir, por exemplo,*timelines* para a entrega de projetos até à criação de diagramas de *Gantt*.

Em termos de análise da aplicação, numa primeira fase do projeto foi necessário identificar a sua arquitetura, os padrões de distribuição, os pontos de configuração e por fim a identificação das operações cujo o desempenho é crítico.

Por fim, na segunda fase é realizado o processo de instalação da aplicação e dos componentes necessários ao seu funcionamento. Depois do processo estar concluído, é feita a automatização de todo o processo de instalação. No final é feita a monitorização e avaliação do desempenho da aplicação.

2 Arquitetura da aplicação

A arquitetura da aplicação *OpenProject* é uma arquitetura *multitier*. Neste capitulo irão ser apresentados os principais componentes - *PostgreSQL*, *Apache2*, *phusion-passenger* e *memcached* - desta arquitetura onde assenta a aplicação.

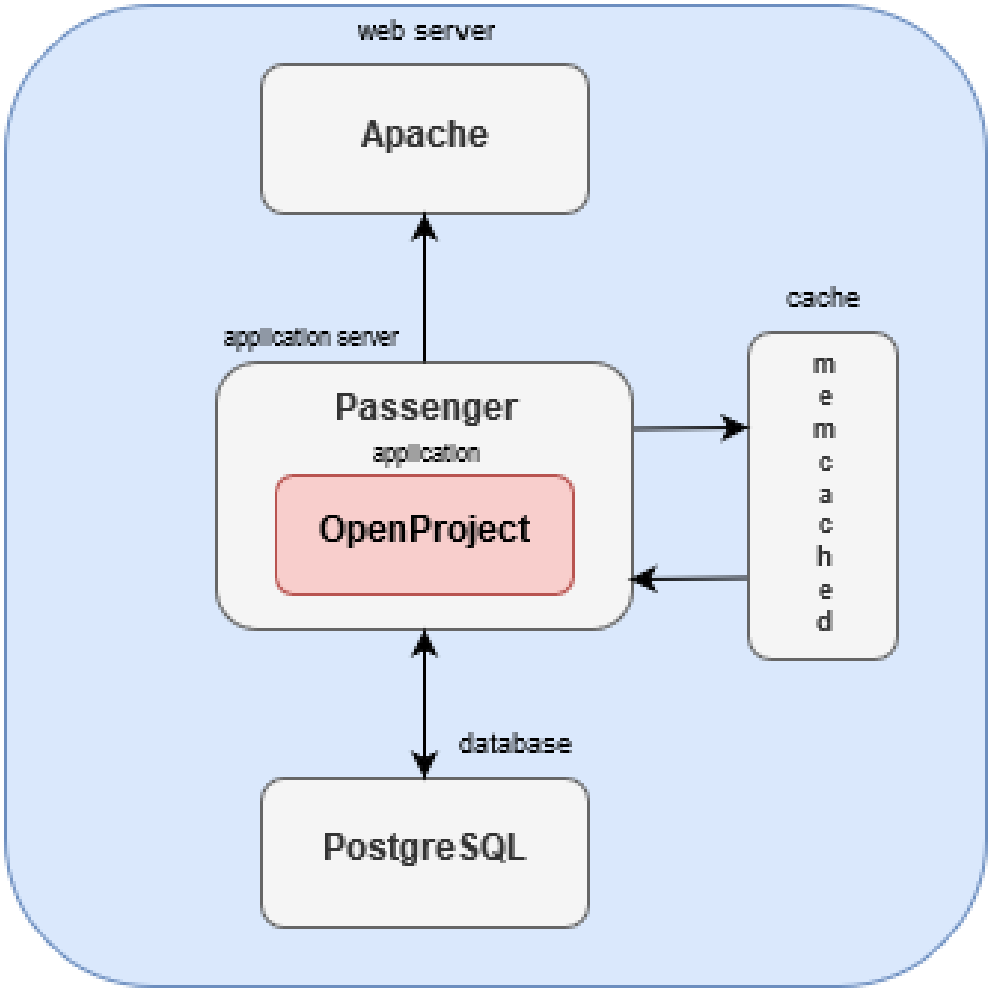

Figura 2.1: Arquitetura do *OpenProject*.

A base de dados usada, como foi dito anteriormente, é o *PostgreSQL*. O *PSQL* é um sistema de gestão de base de dados relacional, e oferece face a outras soluções melhor performance e robustez. A escolha do *PSQL* como o sistema de gestão de base de dados recai sobre o facto de o *OpenProject* ter sido concebido com o *PostgreSQL* como o sistema de gestão de base dados. É permitido mas não aconselhável, implementar um outro sistema de gestão de base de dados.

O componente que faz a comunicação com a base de dados é o *AppServer* neste caso, o *Phusion-Passenger*. O *passenger* faz a conexão do *WebServer* com a base de dados. Para além desta função, os *AppServers* também tem a função de tratar de todas as conexões, configurações e gestão da aplicação.

A aplicação aproveita-se do sistema distribuído de *caching* em memória, o *memcached*, para aumentar a velocidade e reduzir a quantidade de acessos necessários à base de dados principal. Ele consegue isso ao manter dados e objetos em RAM. Desta forma, o sistema consegue reduzir alguns dos gastos computacionais e diminuir o tempo de resposta da aplicação a um pedido.

Para que a aplicação possa comunicar com o cliente através da rede, é necessária a existência de um *WebServer*. Uma forma de o implementar é com o *Apache*. O *Apache* tem a função de comunicar com o *AppServer* e com um cliente através do protocolo *HTTP*. A cada pedido de um cliente para o *WebServer*, este aloca uma *thread* para processar do pedido. Por sua vez, o *AppServer*, que está sempre à escuta na porta do apache, processa o pedido reencaminhado do *WebServer*. Quando o *WebServer* recebe a resposta do *AppServer*, este fecha a *thread* alocada ao pedido já foi processado, e responde ao cliente através de uma mensagem HTTP.

3 Padrões de Distribuição e Pontos de Configuração

Cada componente da aplicação será isolado dos restantes componentes implementados em máquinas diferentes. Este isolamento permite tratar do escalamento e implementação de cada componente de forma independente.

O *PostgreSQL* permite implementar uma arquitetura *master-slave* onde o nodo *master* trata de todas as escritas para a base de dados. Estas escritas são depois replicadas para os *slaves*. A replicação da base de dados para os *slaves* é realizado através de *streaming* síncrono. Os nodos *slave* podem ser definidos em *hot standby* o que significa que estão prontos para substituir o *master* em caso de falha e disponíveis para realizar queries de leitura. Este padrão de distribuição oferece alta disponibilidade, graças aos *slaves* prontos a substituir o *master* em qualquer momento, e um aumento de performance para operações de leitura pelos vários nodos disponíveis para leitura. Como a forma de replicação é síncrona, o *master* espera que os *slaves* em *hot standby* efetuem as suas escritas, isto garante que os dados não serão perdidos em caso de falha no *master*, mas incorre num aumento do tempo de escrita na base de dados.

O *Phusion-Passenger* funciona com uma arquitetura distribuída. A razão para esta distribuição vem do facto de serem necessários vários componentes diferentes para o seu funcionamento. Se todos estivessem junto, uma falha num dos componentes iria afetar todo o sistema. Assim, estando separados, os componentes que falharem podem ser reiniciados sem afetar o funcionamento de todo o *Phusion-Passenger*.

O *Apache* processa pedidos e serve ativos e conteúdo *web* via *HTTP*. Para além disso, o *Apache*, ajuda a distribuir o tráfego através de pedidos *HTTP*.

Este tipo de distribuição é visto como o mais adequado para a aplicação *OpenProject* porque retira a possibilidade da perda de dados durante, por exemplo, a definição da *deadline* de um projeto e oferece alta disponibilidade permitindo assim o desenvolvimento de um projeto em qualquer momento.

4 Operações críticas

Operações críticas destacam-se como sendo operações que podem causar elevada lentidão ou até mesmo falha no sistema. Como tal, foi necessário analisar a arquitetura e o funcionamento geral do sistema de modo a identificar em que situações o desempenho e a disponibilidade do sistema podem ser postos em causa.

Uma operação crítica sobre o sistema é na base de dados, neste caso *PostgreSQL*. Caso esta falhe o serviço fica indisponível devido à impossibilidade de aceder aos dados da aplicação que nela estão guardados. Uma situação que pode causar sobrecarga na base de dados é um elevado número de pedidos feitos sobre a mesma.

Um outro ponto crítico do sistema é o *App Server Phusion-Passenger*. Este é responsável por realizar os pedidos à base de dados, ao *memcached* e comunica também com o *Web Server*. Tal como no caso anterior um excesso de pedidos ao *App Server* pode resultar numa falha do mesmo, que por sua vez torna a aplicação indisponível. O excesso de clientes a tentar aceder aos diferentes componentes pode ser uma das causas para uma falha no *App Server*.

Por fim, outro ponto crítico do sistema é o *Web Server Apache* visto que este é responsável por atender os pedidos *HTTP* do cliente. Se existir sobrecarga de pedidos neste módulo o serviço pode ficar indisponível. Segundo testes realizados neste componente, a sobrecarga pode ser gerada por um elevado número de clientes a tentar aceder ao *Web Server* ao mesmo tempo.

5 Instalação

5.1 Ferramentas

Relativamente ao processo de instalação do *OpenProject* bem como as dependências associadas, foi feito um *provisioning* de instâncias virtuais para a *Google Cloud Platform* e o *deployment* dos serviços, tudo feito através do *Ansible*. O *Ansible* permite a instalação automática dos componentes e dependências da aplicação escolhida, bem como as ferramentas necessárias para efetuar a monitorização da aplicação.

O *Ansible* possui alguns conceitos básicos e cruciais para o seu funcionamento:

- *Inventário* - registo das instâncias organizadas por grupo ou por categoria;
- *Tarefas* - contém as ações a realizar com um dado intuito;
- *Playbook* - conjunto de tarefas;
- *Templates* - ficheiros de configuração de um dado serviço.

O *Ansible* além de permitir a instalação automática dos componentes, permite também a sua configuração através dos *templates* presentes em cada role. Estes permitem facilmente configurar os serviços de acordo com as necessidades, sendo apenas necessário em muitos casos copiar os ficheiros já formatados da máquina local para a respetiva instância no *GCP*. Desta maneira facilita e automatiza o processo de configuração dos serviços.

5.2 Abordagem

Tal como foi referido anteriormente, o *Ansible* foi utilizado para efetuar o *deployment* para as instâncias *gcp* definidas pelo grupo. O grupo optou portanto, por não utilizar o *docker*, portanto a arquitetura adoptada pelo grupo não possui qualquer tipo de *containers*.

Portanto, foram provisionadas cinco instâncias distintas, duas instâncias para a base de dados *PostgreSQL*, ou seja uma base de dados principal e outra em *standby* para o caso de falha sendo que possuem também o *Metricbeat* para a recolha de informação acerca destas duas instâncias. Existe também uma outra instância responsável pela configuração e instalação do *pgpool* que é responsável por garantir a disponibilidade das base de dados *psql* e possui também o *Metricbeat* para além da sua própria base de dados *psql*. Há uma outra instância para a aplicação, ou seja o *OpenProject* onde é necessário correr primeiro o *role* responsável pela criação dos utilizadores e os grupos seguindo-se a instalação e configuração do *OpenProject*, e tal como aconteceu com as instâncias da base de dados, encontra-se também aqui instalado o *Metricbeat* para a recolha de métricas computacionais relativas ao uso do servidor aplicacional. A última instância contém os serviços de monitorização, ou seja o *Elasticsearch* e o *Kibana* bem como o servidor *web* ou seja o *Apache*.

O objetivo da divisão dos serviços por instâncias diferentes, ou seja, a aplicação, o servidor *web* e as bases de dados e o serviço *pgpool* é com o intuito de oferecer mais robustez, segurança e *performance* ao sistema que é algo inerente ao isolamento dos serviços em instâncias distintas, e por consequência também aumenta a disponibilidade do sistema.

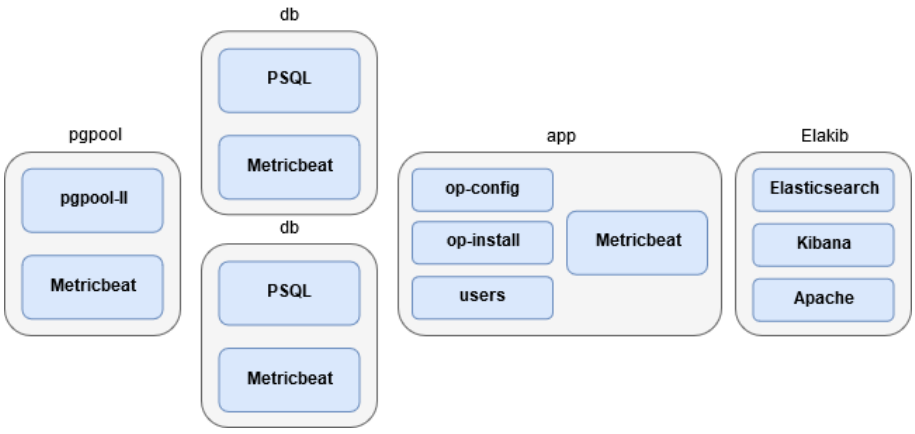

Figura 5.1: Inventário e as respetivas instâncias

De forma a elaborar o provisionamento das instâncias necessárias para implementar a arquitetura anterior foi necessário estruturar o *playbook* da seguinte maneira:

```
- hosts: localhost
  connection: local
  gather_facts: no
  roles:
    - role: gcp
      vars:
       ...
        instances:
          - { index: 1, tag: elakib }
          - { index: 2, tag: [db,be] }
          - { index: 3, tag: [db,be] }
          - { index: 4, tag: [pgpool,be] }
          - { index: 5, tag: app }
```

Listing 5.1: Provisionamento das instâncias

Onde, e tal como foi referido, na primeira instância ou seja a *elakib* é necessário correr os seguintes roles:

```
- hosts: elakib
  become: yes
  roles:
    - Elastic
    - Kibana
    - apache
```

Listing 5.2: *Deployment* dos serviços na instância *elakib*.

Nas próximas duas instâncias, ou seja as instâncias que possuem a base de dados primárias e secundária respetivamente, de modo a conferir a base de dados *PostgreSQL*, foi necessário elaborar o *deployment* da seguinte maneira:

```
- hosts: be
  become: yes
  roles:
    - {role: psql-install}
    - {role: psql-ssh}
    - {role: pgpool-config}
    - Metricbeat
```

Listing 5.3: *Deployment* roles para as instâncias das bases de dados.

A instância responsável por alojar o serviço *pgpool* bem como uma outra base de dados *PostgreSQL* que utiliza para depois aceder às outras bases de dados, ou seja a primária e a secundária, possui um esquema de *deployment* igual ao anterior.

A última instância, ou seja, a *app* que possui a aplicação propriamente dita, ou seja, o *OpenProject* no *playbook* o *deployment* foi feito da seguinte maneira:

```
- hosts: app
  become: yes
  roles:
    - users
    - op-install
    - { role: op-config&start, db_addr: "{{ groups['db'][0] }}" }
    - Metricbeat
```

Listing 5.4: *Deployment* dos serviços na instância *app*.

Todos os roles anteriormente apresentados do *deployment* para as respetivas instâncias irão ser abordados e explicados no capítulo seguinte.

6 Automatização da instalação

A instalação automática bem como a configuração de cada um dos componentes é feita através de *roles* específicos, como tal para cada um dos componentes da aplicação bem como as ferramentas de monitorização definiram-se os seguintes *roles*:

- *gcp* - Cria os discos, redes, endereços e as regras de *firewall* internas e externas necessárias para a criação das instâncias na *Google Cloud Platform*, é também responsável pela criação das instâncias propriamente ditas e faz uso dos módulos disponibilizados pelo *Ansible* que cobrem esta criação e gestão de recursos;

- *users* - Cria um grupo e utilizador denominado *openproject* bem como um novo utilizador *monitor* de modo a agrupar as ferramentas de monitorização num único utilizador;

- *op-install* - Trata de instalar as dependências do servidor aplicacional *OpenProject*, focando-se também particularmente na instalação do *rbenv* e do *nodenv* com o objetivo de gerir e escolher as versões do *Ruby* e do *NodeJS*. Após tratar das dependências aplicacionais, é possível efetuar o download da aplicação através do *git* sendo feito o *npm install* no final para instalar a aplicação. Este *role* é também responsável pela instalação do serviço *memcached* que permite reduzir o tempo de resposta uma vez que oferece um serviço para guardar os objetos em cache na RAM.

- *op-config&start* - Configuração do *OpenProject* utilizando para tal ficheiros que se encontram nos *templates* e nos *files* que possuem a configuração já tratada, sendo apenas necessário de seguida copiar da máquina local para a instância remota no *GCP*.

```
01 |     - name: Copy passenger.conf
02 |       template:
03 |         dest: /etc/apache2/mods-available/passenger.conf
04 |         src: passenger-conf.j2
```

Listing 6.1: Exemplo de uma cópia de um ficheiro de configuração do *passenger* da máquina local para a instância remota app.

É também elaborada a conexão à base de dados, a conexão ao serviço de email e *memcached* e a instalação do *passenger* e módulo do *apache* para o *passenger*. Desta maneira, e atendendo ao exemplo do serviço de email, que obviamente é crucial para a criação de novas contas na aplicação, é possível definir, por exemplo, as portas ao qual este serviço irá se encontrar à escuta, bem como o protocolo de transferência das mensagens, no caso do exemplo seguinte é o *smtp*. O *rails_cache_store* permite que a aplicação em *Rails* elabore e aproveite o serviço de *cache* disponibilizado pelo *memcached*.

```
01 |     production:
02 |         email_delivery_method: :smtp
03 |         smtp_address: smtp.mailtrap.io
04 |         smtp_port: 2525
05 |         ...
06 |         rails_cache_store: :memcache
```

Listing 6.2: Excerto da configuração do serviço de email e *memcached*

- *Elastic* - Download e instalação das dependências necessárias do serviço *Elasticsearch*, tal como, por exemplo, o *JDK 8*, e a respetiva configuração e inicialização do serviço *Elasticsearch*;
- *Kibana* - Download e instalação do *Kibana* bem como a sua configuração dos endereços e das portas.
- *Metricbeat* - Download, instalação e configuração dos endereços e das portas e respetiva inicialização do serviço;
- *psql* - Instalação da base de dados *PostgreSQL* e inicialização do serviço.
- *psql-install* - Configuração do *PostgreSQL* cujos ficheiros de configuração já preparados são provenientes da pasta *files*, sendo apenas necessário copiar da máquina local para a diretoria apropriada na instância remota .
- *psql-ssh* - Preparação para elaborar o *SSH* entre as instâncias da base de dados sem pedir password, o âmbito deste *role* é para depois facilitar a integração do *pgpool*.
- *pgpool-config* - Download e configuração do serviço *pgpool* que permite garantir elevada disponbilidade da base de dados *PostgreSQL*, no final é criado um utilizador para a base de dados e a própria base de dados para a utilização pela aplicação *OpenProject*.
- *apache* - Instalação do *web server apache*, onde numa primeira instância é instalado o serviço, sendo de seguida configurado com *Proyx Pass Reverse* na porta 5601.

```
01 |       path: "/etc/apache2/sites-enabled/000-default.conf"
02 |       insertafter: "</VirtualHost>"
03 |       block: |
04 |         <Location /kibana/>
05 |           ProxyPass http://localhost:5601/
06 |           ProxyPassReverse http://localhost:5601/
07 |         </Location>
```

Listing 6.3: Excerto da configuração do apache

Relativamente a um dos serviços mais críticos do sistema, ou seja, a base de dados *PSQL*, foi implementado o serviço *pgpool2* de modo a garantir uma alta disponibilidade da base de dados. Como tal, foi implementado um esquema *Master/Slave*, no qual se destaca a existência de uma base de dados principal, bem como uma base de dados em *standby* sendo esta uma réplica da base de dados principal. A base de dados em *standby* necessita, tal como seria expectável, de ter os mesmos dados que a base de dados principal uma vez que esta ao ser promovida a *master*, em caso de falha do *master* atual, terá que garantir a persistência dos dados. Para além disso, o *pgpool* garante também a distribuição da carga pelas duas base de dados.

A replicação da base de dados é facilmente obtida nativamente com *PostgreSQL*, sendo apenas necessário alterar o ficheiro de configuração *postgresql.conf* e adicionar o novo endereço da base de dados que se pretende adicionar no *pg_hba.conf*. Permitindo desta maneira adicionar mais um servidor de base de dados em *standby*, o grupo para efeitos de teste definiu apenas uma base de dados principal e uma outra em *standby*.

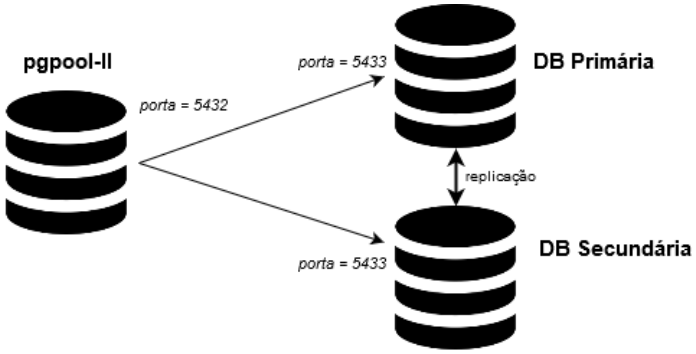

Figura 6.1: Esquema do funcionamento geral da base de dados.

Inicialmente estava previsto implementar um esquema de replicação *master-slave* que iria introduzir *pgpool* em cada uma das máquinas com processos *watchdog* de forma a que as bases de dados estivessem frequentemente em comunicação, monitorizando-se entre si para elegerem uma nova base de dados *master* ou uma nova instância *pgpool* ativa caso um destes deixe de comunicar com os restantes. Os utilizadores da base de dados teriam de acedê-la através de um IP virtual que iria apontar para a instância *pgpool* ativa. Na *Google Cloud Platform* encontraram-se dificuldades na implementação

deste IP flutuante pelo facto da sua rede ignorar as *frames* de *ARP* gratuito, isto levou a que fosse implementada outra organização da base de dados.

O esquema implementado envolve uma única instância *pgpool* que gere as bases de dados associadas, de forma a que, caso a BD *master* seja perdida, uma das BDs em *hot standby*, que até ao momento podiam apenas ser lidas, seja promovida a *master*. Este esquema é bastante mais simples, em que o cliente precisa apenas de comunicar com a máquina monitora. No entanto, este esquema possui um ponto único de falha na máquina monitora, pelo que esta máquina irá correr apenas a instância *pgpool* de forma a evitar o seu sobrecarregamento.

Relativamente ao *download* dos serviços propriamente ditos, o *Ansible* oferece uma panóplia de diretivas para efetuar o *download* tais como o *apt*, *get_url*, entre outros.

```
01 | - name: Download do elasticsearch
02 |   get_url:
03 |     url: https://artifacts.elastic.co/downloads/elasticsearch/
       elasticsearch-7.4.2-linux-x86_64.tar.gz
04 |     dest: /home/monitor/
05 |   become_user: monitor
```

Listing 6.4: Exemplo de um download de um dos serviços de monitorização.

7 Monitorização

De modo a avaliar o desempenho do sistema, é necessário recorrer a ferramentas capazes de apresentar métricas que permitem a avaliação do bem estar do sistema. Como tal, estas métricas servem para facilitar a tomada de decisão sobre mudar a arquitetura do sistema em caso de um desempenho insatisfatório.

7.1 Ferramentas

Com este intuito, foram utilizadas as seguintes três ferramentas que permitem a monitorização do sistema:

- *Metricbeat* - Permite a recolha de diversos tipos de métricas, desde a utilização do *CPU* até à utilização da memória e o *uptime* do sistema;
- *Elasticsearch* - Motor de busca, de agregação e recolha das métricas devolvidas pelo *Metricbeat*.
- *Kibana* - Implementa a interface gráfica de modo a visualizar e aceder à informação presente no *Elasticsearch*, permitindo a navegação e interpretação dos dados através de *dashboards* que possuem gráficos referentes a cada um dos serviços que foram monitorizados.

No ponto de vista da estrutura definida pelo grupo, o *Metricbeat* encontra-se instalado em todas as máquinas virtuais em que o grupo deseja ver as métricas. Por exemplo, para avaliarmos recursos computacionais gasto pelo *OpenProject*, o *Metricbeat* está instalado na mesma máquina virtual que o *OpenProject*.

Desta maneira, é possível, por exemplo, caso a taxa de utilização da memória seja elevada, adicionar mais memória à máquina de modo a colmatar esta ineficiência.

7.2 Métricas

Relativamente às métricas colecionadas pelo *metricbeat* estas são variadas, e permitem a recolha de informação tal como o uso de *CPU*, de memória, o número de processos atuais, e até mesmo o *IO* em disco. Através destas métricas, e tal como foi referido anteriormente, é possível tirar conclusões acerca da arquitetura do sistema.

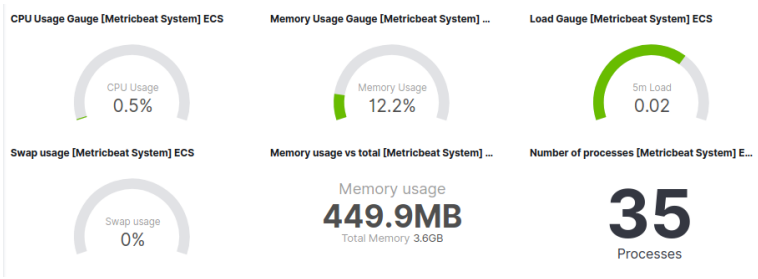

Figura 7.1: Visualização de um conjunto de métricas através do *kibana*.

O conjunto total de *dashboards* poderá ser consultado e visualizado através da interface gráfica disponibilizada pelo *Kibana*.

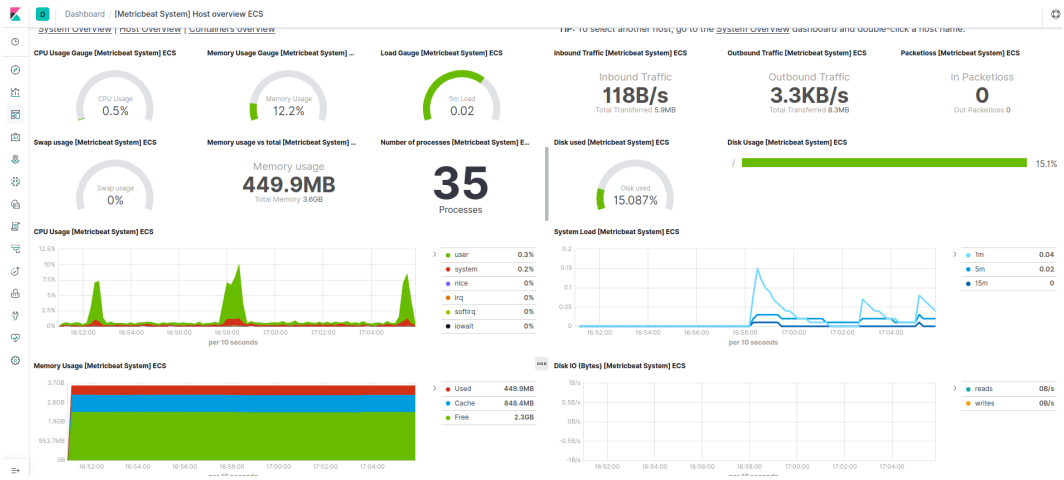

Figura 7.2: Conjunto de *dashboards* disponíveis no *kibana*.

Tal como seria de esperar, o *metricbeat* encontra-se instalado nas instâncias correspondentes ao servidor aplicacional e nas bases de dados uma vez que permite identificar anomalias nessas instâncias tal como uma sobrecarga do sistema. O *elasticsearch* e o *kibana* encontram-se numa instância separada das duas anteriores.

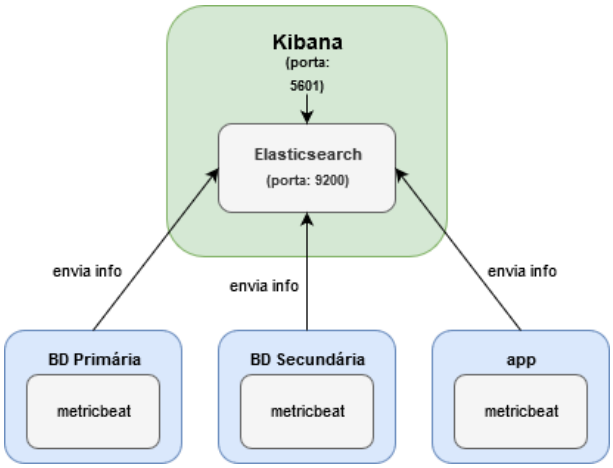

Figura 7.3: Esquema simples da disposição das ferramentas de monitorização.

8 Avaliação

Com o intuito de avaliar o desempenho e a eficácia da arquitetura implementada pelo grupo, foi necessário recorrer a uma ferramenta que simulasse a carga de trabalho sobre um sistema. A ferramenta escolhida com este intuito foi o *Apache JMeter*. O *Apache JMeter* permite simular facilmente os utilizadores no sistema, sendo apenas preciso aumentar o número de *threads*, e para além disso também permite especificar o tipo de interação que as *threads* terão com o serviço, como por exemplo, colocar as *threads* (utilizadores) a realizar *HTTP Requests*, de seguida e de forma a visualizar os resultados na ferramenta, é possível adicionar *listeners* para apresentar os resultados da simulação na forma de um grafo com o tempo de resposta ou até mesmo para gravar os resultados numa tabela.

Com o *JMeter* seria depois possível analisar métricas importantes no contexto real de uma aplicação como por exemplo o *throughput* e o *response time* que iriam indicar os aspetos a melhorar em termos de *performance* do sistema montado.

8.1 Resultados

A ferramenta mencionada permite a exportação dos resultados do *benchmark* para um ficheiro de modo a depois poder elaborar uma análise estatística sobre esses mesmos dados a fim de poder descobrir a *performance* do sistema. O próprio *Kibana* também é útil de modo a avaliar o esforço computacional das instâncias com uma carga de trabalho simulada, pelo que se poderá ver na seguinte imagem os resultados no *Kibana* após uma simulação através do *Apache JMeter*.

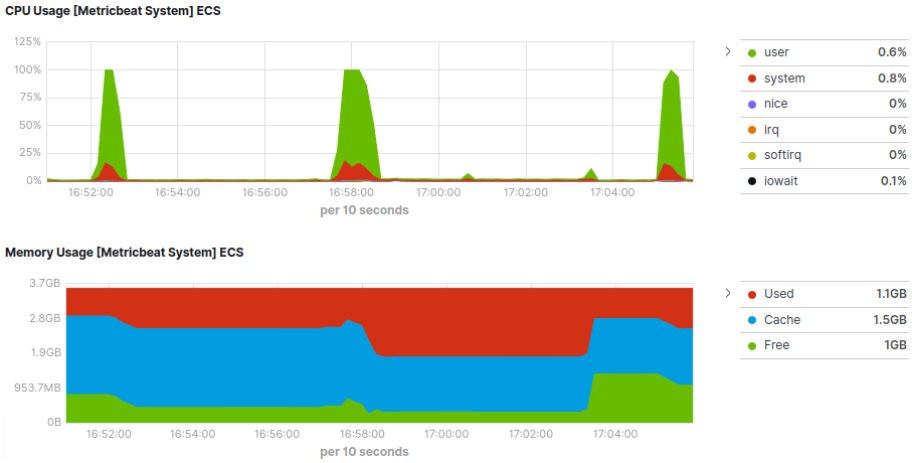

Figura 8.1: Kibana taxa de utilização do *cpu* e memória após simulação de *workload* com mil utilizadores.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
31	17:05:10.402	Thread Group 1-91	HTTP Request	2206		459	115	2206	244
542	17:05:10.403	Thread Group 1-92	HTTP Request	7004		19129	115	5426	246
470	17:05:10.404	Thread Group 1-93	HTTP Request	5790		19129	115	5475	248
477	17:05:10.405	Thread Group 1-94	HTTP Request	6116		19129	115	5836	249
475	17:05:10.406	Thread Group 1-95	HTTP Request	5833		19129	115	5751	247
700	17:05:10.407	Thread Group 1-96	HTTP Request	11385		19129	115	11303	246
11	17:05:10.408	Thread Group 1-97	HTTP Request	2072		459	115	2072	247
705	17:05:10.409	Thread Group 1-98	HTTP Request	11600		19129	115	11518	247
484	17:05:10.410	Thread Group 1-99	HTTP Request	6184		19129	115	6101	253
490	17:05:10.411	Thread Group 1-100	HTTP Request	6211		19129	115	6118	252
493	17:05:10.412	Thread Group 1-101	HTTP Request	6260		19129	115	6139	256
492	17:05:10.413	Thread Group 1-102	HTTP Request	6228		19129	115	6125	254
182	17:05:10.414	Thread Group 1-103	HTTP Request	3198		459	115	3198	3104
183	17:05:10.415	Thread Group 1-104	HTTP Request	3202		459	115	3202	3103
118	17:05:10.416	Thread Group 1-105	HTTP Request	2707		459	115	2707	1917
728	17:05:10.417	Thread Group 1-106	HTTP Request	12592		19129	115	12489	3103
498	17:05:10.418	Thread Group 1-107	HTTP Request	6319		19129	115	6226	251
499	17:05:10.419	Thread Group 1-108	HTTP Request	6386		19129	115	6289	252
502	17:05:10.420	Thread Group 1-109	HTTP Request	6454		19129	115	6371	251
593	17:05:10.421	Thread Group 1-110	HTTP Request	7408		459	115	7407	7302
185	17:05:10.422	Thread Group 1-111	HTTP Request	3214		459	115	3214	3101
186	17:05:10.423	Thread Group 1-112	HTTP Request	3218		459	115	3218	3105
507	17:05:10.423	Thread Group 1-113	HTTP Request	6597		19129	115	6453	251
194	17:05:10.425	Thread Group 1-114	HTTP Request	3289		459	115	3289	3147
230	17:05:10.426	Thread Group 1-115	HTTP Request	4025		459	115	4025	3146
754	17:05:10.427	Thread Group 1-116	HTTP Request	13732		19129	115	10410	250
691	17:05:10.428	Thread Group 1-117	HTTP Request	10931		19129	115	10406	249
195	17:05:10.429	Thread Group 1-118	HTTP Request	3290		459	115	3290	3143
508	17:05:10.430	Thread Group 1-119	HTTP Request	6667		19129	115	6580	248
196	17:05:10.431	Thread Group 1-120	HTTP Request	3291		459	115	3290	3142
188	17:05:10.432	Thread Group 1-121	HTTP Request	3240		459	115	3240	3131
12	17:05:10.433	Thread Group 1-122	HTTP Request	2050		459	115	2050	246
703	17:05:10.434	Thread Group 1-123	HTTP Request	11511		19129	115	11363	246
205	17:05:10.436	Thread Group 1-124	HTTP Request	3367		459	115	3367	246
198	17:05:10.438	Thread Group 1-125	HTTP Request	3287		459	115	3287	3136
779	17:05:10.439	Thread Group 1-127	HTTP Request	15181		19129	115	14828	243
509	17:05:10.440	Thread Group 1-126	HTTP Request	6688		19129	115	6585	243
10	17:05:10.441	Thread Group 1-130	HTTP Request	2038		459	115	2038	244
518	17:05:10.441	Thread Group 1-129	HTTP Request	6762		19129	115	6658	243
13	17:05:10.442	Thread Group 1-131	HTTP Request	2041		459	115	2041	246
189	17:05:10.443	Thread Group 1-132	HTTP Request	3232		459	115	3232	3123
527	17:05:10.443	Thread Group 1-128	HTTP Request	6821		19129	115	6736	253

Figura 8.2: Resultados de uma simulação no *JMeter* na forma de uma tabela com mil utilizadores.

Na figura apresentada anteriormente, foi elaborada uma simulação sobre o servidor aplicacional *OpenProject*, onde foram alocadas mil *threads* (utilizadores) no *thread group*, é possível observar na figura a latência dos pedidos, onde é medida a latência no instante antes de mandar o pedido até obter a primeira resposta e o tempo que as *threads* demoraram a estabelecer a coneção, estas são as métricas mais relevantes para analisar. Portanto, e observando os resultados é possível verificar que a tempo de conexão anda na ordem dos 250 milissegundos enquanto que a latência possui uma maior variação quando comparada com o tempo de conexão.

Através deste mesmo teste, verificámos que uma máquina com um *core* não era capaz de dar uma resposta nem a 50 % dos pedidos, o uso do *CPU* era de 100 % e o uso da memória era, em média, de 80/85 %. Portanto, trocámos para uma máquina de oito *cores* mas nessa máquina o uso do *CPU* era apenas de 25%, um número demasiado baixo. No final, decidimos seguir com máquinas de quatro *cores* que fornecem uma boa relação custo/desempenho.

Por outro lado, mesmo usando máquinas de oito *cores*, o nosso *web server* continuava sem conseguir responder a todos os pedidos sendo que isto é um indicador que este devia ser replicado ou, por outro lado, a replicação do *app server* também podia ajudar.

9 Trabalho Futuro

9.1 Métricas

Apesar do processo de instalação e de configuração das métricas (*Elasticsearch*, *Kibana* e *Metricbeat*) estar totalmente automatizado, durante este trabalho não foi possível deixar as aplicações das métricas a ligarem automaticamente, ou seja, para ser possível ver as métricas no *browser*, é preciso ir manualmente à máquina virtual do *Elasticsearch* e *Kibana* para ligar ambos os serviços e, depois disso, é preciso ir a cada um das máquinas que desejámos ver as métricas ligar o *metricbeat*.

9.2 *Apache JMeter*

Devido ao tempo disponível, não foi possível realizar uma avaliação de qualidade do sistema. O *JMeter* é um programa bastante poderoso que permitir testar com certeza vários aspetos de uma aplicação, como trabalho futuro, devemos realizar mais testes, mais diversos e mais focados em determinadas partes dos sistema, como a base de dados ou o *app server*. Um dos testes que o grupo tem mais curiosidade de fazer é o *login* na aplicação do *openproject* através do *JMeter*.

9.3 *Apache* e *Passenger*

Por questões de tempo, falta de informação e demora no processo de *debug* não foi possível separar o *apache* do *phusion-passenger*, tendo ficado instalados na mesma máquina. Assim, como trabalho futuro, o grupo pretende separar os dois componentes. Através da instalação do *phusion-passenger standalone* e da de configuração de *reverse proxy*, será possível fazer o redirecionamento entre os *IPs* e portas das máquinas.

9.4 Replicação da base de dados

O processo de replicação da base de dados encontra-se automatizado quanto à promoção de um novo *master* do sistema, no entanto ainda existem vários aspetos sobre os quais se podem melhorar ou automatizar. Nas condições atuais é necessária intervenção manual para recuperar uma base de dados que foi dada como morta, o que resulta numa máquina inativa durante este tempo. Outro ponto que iria melhorar a resiliência do sistema seria a remoção do ponto único de falha existente no monitor *pgpool* através do esquema de distribuição referido anteriormente.

10 Conclusão

Numa primeira instância do projeto após uma extensa análise da aplicação *OpenProject* foi possível definir a arquitetura geral da aplicação, onde foi possível identificar os principais módulos e componentes que compõem a aplicação, de seguida foi também possível identificar um possível padrão de distribuição para a aplicação em questão, bem como os respetivos pontos de distribuição. Por fim, foram identificadas as operações críticas da aplicação.

Neste processo descrito anteriormente, o grupo sentiu mais dificuldades a identificar os padrões de distribuição e a arquitetura da aplicação. Ambas as dificuldades foram causadas tanto pela falta de informação útil no site da aplicação como no processo de instalação, um processo quase totalmente automatizado e com poucas explicações sobre as escolhas feitas.

Prosseguindo para a fase seguinte, que era referente ao processo de *provisioning* e *deployment* da aplicação e todas as suas dependências na *Google Cloud Platform*, bem como ferramentas de monitorização. Por isso, foi imediatamente possível sentir a importância que a automatização trás aos projetos. Como tal, ferramentas como o *Ansible* através de alguns *roles* em questões de minutos conseguem efetuar o provisionamento das instâncias, e a instalação e configuração dos componentes, neste caso em instâncias remotas. Para além disso, é também evidente a importância que as ferramentas de monitorização trazem para a administração da qualidade e da performance de sistemas distribuídos, uma vez que são cruciais de modo a se poder identificar as potenciais falhas no sistema.

Em suma, e embora tenham ocorridas muitas situações indesejáveis na elaboração do trabalho e alguns objetivos que não foram possíveis ser alcançados (objetivos esses apresentados no capítulo anterior), este trabalho revelou-se importante para consolidar os conhecimentos acerca de *Ansible* e também sobre todo o processo de implementação de um sistema distribuído bem como a sua administração.