Universidade do Minho
Escola de Engenharia

System Deployment and Benchmarking

2017/2018

Trabalho Prático

Odoo

Grupo 6

João Coelho A74859

Paulo Guedes A74411

Pedro Cunha A73958

Contextualização

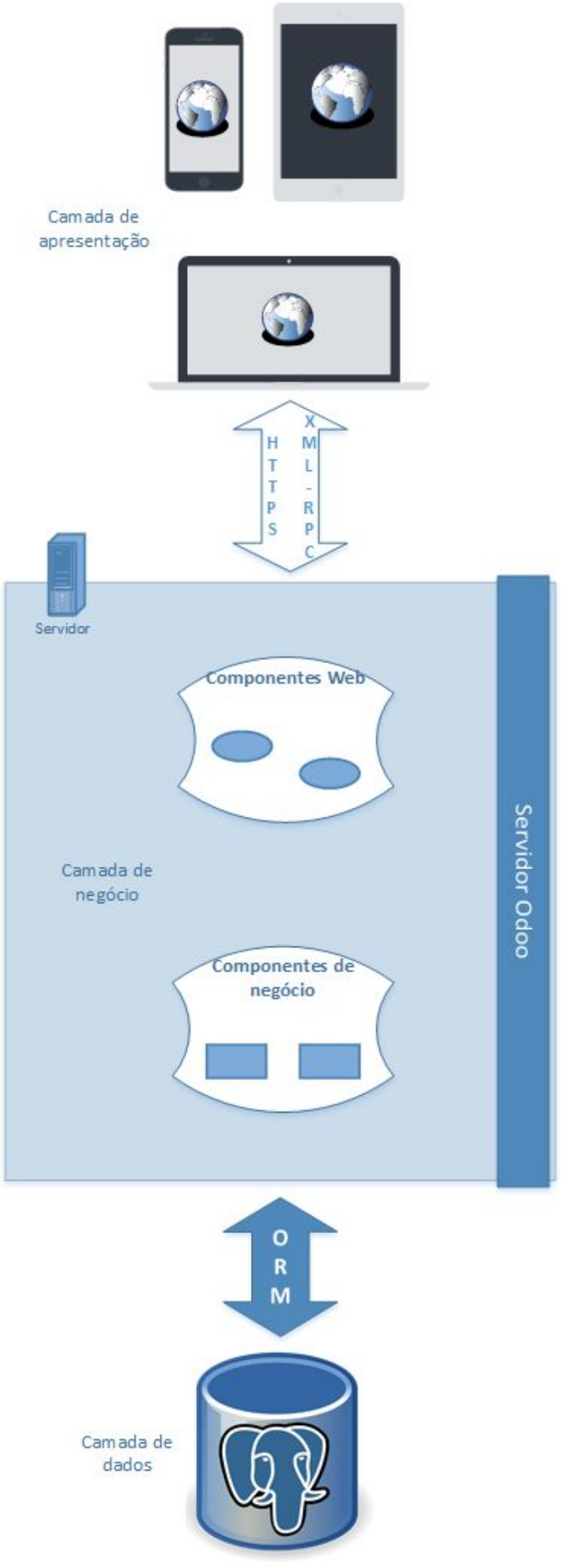
Odoo é um software de gestão empresarial, desenvolvido pela Odoo S.A., que oferece ao utilizador um conjunto de aplicações auxiliares na área dos negócios, com uma estratégia que se foca essencialmente no cliente e capaz de responder às necessidades básicas de uma empresa. Software de CRM, e-Commerce, contabilidade, faturação, construção de websites, gestão de inventários e recursos humanos são algumas das aplicações que compõem este pacote de software disponibilizado pelo Odoo. Habitualmente apelidadas de módulos, estas aplicações conseguem facilmente adaptar-se às necessidades, dada a sua variedade, flexibilidade e, também importante, abertura à comunidade (*open-source*). Por outro lado, esta modularidade associa-se à facilidade de uso de cada módulo para tornar bastante célere a adaptação e entrada na fase de produção.

Em oposição ao que é implementado em muitas outras empresas do setor, que se focam em vertentes específicas, o Odoo disponibiliza ao seu cliente um modelo ERP (*Enterprise Resource Planning*), ou seja, integra todos os dados e vertentes da empresa num único sistema tecnológico. Por aqui percebe-se o motivo pelo qual o Odoo foi inicialmente publicado com o nome Tiny ERP, passando ainda pelo nome OpenERP, antes de, a partir da versão 8, ser intitulado de Odoo (curiosidade: os criadores da Odoo S.A. analisaram os nomes das maiores companhias na internet e descobriram uma relação direta entre o valor da companhia e o número de 'O' no seu nome).

A Odoo S.A. lança duas versões do Odoo: a **Community Edition** e a **Enterprise Edition**. A primeira é gratuita, *open-source* e essencialmente suportada pela comunidade Odoo. Já a segunda versão, sendo também *open-source*, requer uma licença, com base no número de utilizadores e da região. Em termos da aplicação Odoo em si, sumariamente, a melhoria apresentada pela versão *Enterprise* está na melhor interface, no melhor desempenho de algumas aplicações, no suporte fornecido pela Odoo S.A. e na migração para as novas versões do Odoo. Naturalmente, este projeto irá categorizar e analisar a versão Community.

Seguindo a tendência das aplicações atuais, o Odoo oferece a flexibilidade à empresa, ou, mais genericamente, ao utilizador, de escolher utilizar o seu próprio hardware para *host* ou os *web services* do Odoo. O uso de tecnologias modernas, como PostgreSQL e Python, e a forte aposta no desenvolvimento para *mobile* e na integração de *web services* são também aspetos positivos do Odoo, que serão abordados no decorrer deste documento.

Arquitetura e componentes da aplicação



A arquitetura do software Odoo é composta por três camadas: apresentação, negócio e dados. De seguida, é feita uma análise pormenorizada de cada uma das camadas referidas anteriormente.

Camada de apresentação

Assumindo-se como a camada mais simples da arquitetura, a camada de apresentação é composta por páginas de *front-end,* com HTML e CSS, e uma aplicação Javascript em *back-end*, que emite pedidos a um servidor e apresenta o resultado ao cliente. A camada de apresentação do Odoo é, portanto, *web-based*, correndo num browser.

Camada de negócio

No que diz respeito à camada de negócio, é aqui que se encontra o servidor do Odoo, capaz de suportar aplicações auxiliares (módulos), onde se concentra toda a lógica do negócio. O servidor de aplicação assume-se igualmente como uma framework de desenvolvimento que disponibiliza recursos bastante úteis no que diz respeito à implementação das aplicações referidas anteriormente. A existência destes recursos, onde podemos destacar o mapeamento objeto-relacional, a arquitetura model-view-control ou software de geração de relatórios, faz com que seja possível destacar três sub-camadas.

Numa primeira camada encontra-se o ORM (Mapeamento objeto-relacional), através do qual grande parte dos acessos à base de dados se realizam (posteriormente é descrito o processo de acesso). Já no que diz respeito à segunda camada, encontram-se os módulos, onde estão descritos os *business object* que representam todos os dados do programa (modelos, vistas, workflows, etc) relativos a um determinado requisito de negócio. Finalmente, relativamente à terceira camada, encontra-se aqui o servidor web do Odoo e a interface que permite a comunicação entre o servidor aplicacional e os clientes web, os browsers. É o servidor web do Odoo que, através do werkzeug, uma libraria Python, processa todos os pedidos web provenientes do cliente.

Ainda relativo à camada de negócio, podemos destacar a presença de motores de fluxos de trabalho, grafos capazes de descrever um conjunto de dinâmicas e processos associadas a um *business object*.

Camada de dados

A camada de dados da plataforma é implementada através da utilização de uma base de dados PostgreSQL, onde se encontram os dados correspondentes aos vários módulos e grande parte os ficheiros de configuração do Odoo. No decorrer da instalação de um módulo, as estruturas das bases de dados necessárias para as suas funcionalidades são definidas num modelo. Posteriormente, a framework Odoo consulta esse modelo e utiliza o ORM referido acima para criar essas mesmas estruturas na base de dados PostgreSQL.

Padrões de distribuição e formas de comunicação

O Odoo adota uma arquitetura MVC (*Model-View-Controller*), em que divide o software em três partes interligadas: Modelo, Visão e Controlador. Numa aplicação complexa como o Odoo, com grandes quantidades de informação a apresentar ao utilizador, é habitualmente desejável separar os dados (m*odelo*) das preocupações associadas à *user interface* (*visão*). O princípio base é: alterações na interface não devem ter impacto na gestão dos dados e os dados devem poder ser reorganizados na base de dados sem alterarem a interface. O MVC resolve este problema separando o acesso aos dados e a lógica de negócio da apresentação dos dados e da interação com o utilizador, introduzindo um intermediário - o controlador.

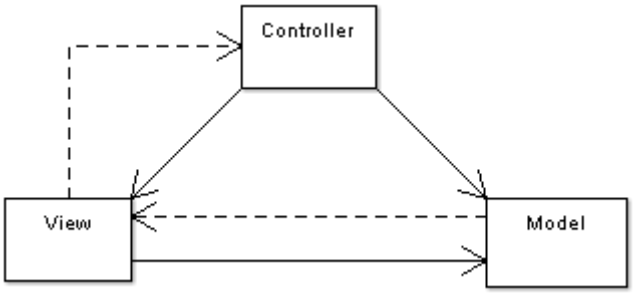


Diagrama do Model-View-Controller

Aplicando ao software em estudo, modelos são aquilo que armazenam e gerem os dados da aplicação - as tabelas PostgreSQL. Por exemplo, quando uma ordem de venda é guardada no Odoo, os dados associados a essa venda são armazenados num modelo apropriado. O controlador é composto pelo servidor do Odoo, que contém todo o poder de processamento do software (*web services* e módulos), o que facilita a manutenção e o desenvolvimento do Odoo. Por fim, o componente da visão consiste no Web Browser, representando a interface gráfica apresentada ao cliente (vistas desenvolvidas em XML).

Estes três componentes interagem entre si através de camadas próprias, específicas para certos tipos de comunicação. Entre modelos e o controlador, é utilizada a técnica do ORM (Object-relational-mapping). Já a comunicação entre a visão e o controlador ocorre pela interação entre os browsers dos clientes e os *web services* fornecidos pela API do Odoo, por meio de protocolos de chamada de procedimento remoto (RPC).

ORM (Mapeamento objeto-relacional)

O ORM está associado ao controlador, fazendo a ligação entre a base de dados e o servidor do software. É responsável por realizar pedidos à base de dados, fazendo uma "tradução" da linguagem Python para queries em SQL.

Todas as operações sobre a base de dados são realizadas recorrendo ao ORM, quer seja a leitura, escrita, procura ou a criação de registos, existindo métodos em Python que fazem a "tradução" para SQL, realizando as queries indicadas para cada operação, facilitando, assim, os acessos à base de dados.

Para uma melhor eficiência e redução dos acessos à base de dados, o Odoo mantém uma cache, que contém registos acedidos recentemente, guardando alguns antecipadamente com base em algumas heurísticas.

Camada Web

Também associados ao controlador temos os *web services* do Odoo, que estabelecem a ligação entre o cliente e o servidor. *Web services* são um conjunto de ferramentas e especificações que permitem que o software corra em diferentes sistemas operativos, em diferentes ambientes, para fazer chamadas de procedimento remoto (RPC) pela Internet. No caso do Odoo, o cliente pode comunicar com o servidor através do protocolo XML-RPC, JSON-RPC (igual ao protocolo XML-RPC, apenas codificado em JSON).

Dada a semelhança entre os protocolos, explicar-se-á apenas um: o XML-RPC. Este é o procedimento chamado remotamente através de HTTP, codificado em XML e estruturado de maneira a ser o mais simples possível, permitindo que estruturas de dados complexas sejam transmitidas, processadas ou retornadas. Os pacotes HTTP são enviados através da porta 8069 sempre que o servidor, em resposta ao cliente, desejar comunicar através dos protocolos disponibilizados pela web service.

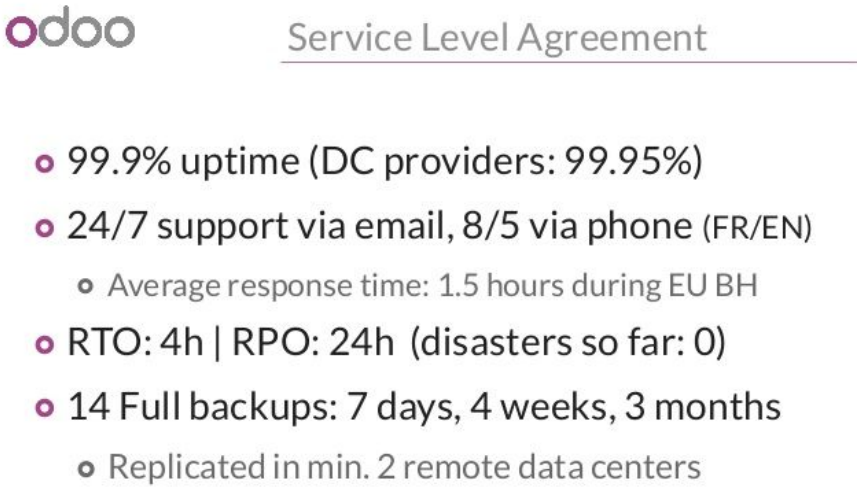
Cada evento realizado pelo cliente é comunicado ao servidor, respondendo este com a nova ação a ser realizada pelo cliente, seja esta um pedido de dados, uma ação realizada pelo cliente (e.g.: clicar num botão) ou uma vista (dados sobre a nova estrutura que o cliente necessita de carregar).

Operações críticas

No que diz respeito às operações críticas associadas à plataforma Odoo, estas surgem maioritariamente num contexto ligado à utilização do software por parte de grandes companhias, onde a problemática da escalabilidade e disponibilidade é uma realidade.

Este tipo de operações podem ser encontradas nas várias camadas da plataforma. No que diz respeito, por exemplo, à camada de aplicação, os pedidos que chegam das várias aplicações web, e que devem ser respondidos pelo *web service*, assumem-se como operações de elevado nível crítico para todo o sistema, já que um grande número de pedidos dificilmente consegue ser suportado por um único servidor. Em aplicações associadas a negócios, facilmente existem picos, de encomendas, vendas, candidaturas, entre outros. Assim, a inserção de mais servidores web, aliada à existência de um balanceador de carga capaz de distribuir os pedidos pelos vários servidores, pode solucionar a problemática e aumentar a escalabilidade do sistema.

Porém, além da escalabilidade, existe também a preocupação com a disponibilidade das infraestruturas associadas ao Odoo, quer sejam os *web services* ou a base de dados. Quando se coloca uma empresa a depender de um software, naturalmente espera-se que este esteja acessível a qualquer momento. Neste aspeto, a criticidade da aplicação está, não no utilizador, mas nos fornecedores dos serviços. Em baixo, é possível consultar alguns dados estatísticos relativos à disponibilidade do serviço da Odoo em 2014 (RTO - *Recovery Time Objective;* RPO - *Recovery Point Objective*).



Uma das características deste tipo de plataforma é a enorme quantidade de dados, que a utilização dos diferentes módulos exige que sejam armazenados. É fácil então concluir que este processo de armazenagem de grandes quantidades de dados é, igualmente, uma operação crítica. Com muita informação armazenada, acessos à base de dados que impliquem o varrimento de mais tabelas podem aumentar a latência da operação. Imagine-se o envio simultâneo de pedidos de filtragem das faturas de valor superior a 100 euros, no último ano, e pedidos de listagem de inventário de todos os produtos. Numa empresa com dimensão considerável, a carga colocada no servidor seria exagerada. A solução é bastante semelhante ao caso dos *web services*, e passa pela utilização de clusters na base de dados, com a implementação de sharding.

Outro aspeto importante associado a uma aplicação de gestão empresarial é a segurança dos dados. Num pacote de software onde podemos ter aplicações de controlo de receitas, inventários, recursos humanos, entre outras, há muita informação com valor que o utilizador certamente não pretende divulgar. Como tal, as operações executadas pelo utilizador devem ser confidenciais. O uso do protocolo HTTPS, na comunicação entre o browser e a camada Web do servidor Odoo, é uma das medidas que contribui para a segurança dos dados e operações.

Pontos de configuração

Alguns dos ficheiros de configuração que compõem a plataforma Odoo têm como função relacionar componentes da arquitetura entre si.

port:	Porta a utilizar pelo servidor
database:	Nome da base de dados
init:	Iniciar um módulo (usar "all" para todos os módulos)
update:	Atualizar um módulo (usar "all" para todos os módulos)
upgrade:	Atualizar/instalar/desinstalar módulos
db_name:	Especificar o nome da base de dados
db_password:	Especificar a password da base de dados
db_port:	Especificar a porta da base de dados

Na figura acima, podemos observar alguns dos campos que é possível adicionar ao ficheiro de configuração do servidor de aplicação Odoo. Campos como db_port e init mostram claramente a configuração que permite a interação entre diferentes componentes da arquitetura, neste caso, o servidor, a base de dados e os

diferentes módulos. No que diz respeito ao ficheiro de configuração do servidor web, aqui também é possível verificar a existência de alguns campos que permitem a sua configuração, como podemos verificar pela figura seguinte:

```
[global]
# Some server parameters that you may want to tweak
server.socket_host = "0.0.0.0"
server.socket_port = 8080

# Sets the number of threads the server uses
server.thread_pool = 10

server.environment = "development"

# Simple code profiling
server.profile_on = False
server.profile_dir = "profile"

# if this is part of a larger site, you can set the path to the TurboGears instance here
server.webpath = ""

#[logging]
#log.access_file = "/var/log/openerp-web/access.log"
#log.error_file = "/var/log/openerp-web/error.log"

# OpenERP Server
[openerp]
host = 'localhost'
port = '8070'
protocol = 'socket'
```

Que componentes foram replicados e porque motivo

Na fase anterior do projecto foram enunciadas algumas operações críticas associadas à aplicação Odoo e, consequentemente, apresentadas soluções relacionadas com determinadas problemáticas como a falta de escalabilidade e/ou disponibilidade do serviço. De entre as várias soluções propostas, podemos destacar, por exemplo, a replicação de componentes presentes em determinadas camadas.

Na presente fase, atendendo ao que era proposto pelo enunciado do trabalho, foi posta em prática a replicação ao nível da camada aplicacional. O principal objectivo associado à replicação é permitir uma maior escalabilidade da plataforma. Tal como foi referido anteriormente, a plataforma Odoo, associada a um contexto empresarial de grande escala, com um elevado número de pedidos a um único servidor web, pode sofrer uma quebra significativa da sua performance, caso esta não seja replicável. A replicação da camada de aplicação, onde está presente toda a lógica aplicacional da plataforma (servidor web, módulos, etc), por si só não torna a plataforma escalável. A presença de um balanceador de carga como, por exemplo, HAProxy, torna-se assim fundamental para distribuir a carga entre os diversos servidores.

Num cenário real, a camada de dados também deveria ser replicada, até porque a aplicação pode comportar grandes volumes de dados. Por uma questão de gestão de tempo e pelo facto de o projeto não exigir a replicação das várias camadas, optou-se por replicar a camada web, mas, estando conscientes da sua necessidade, explorou-se como é que esta replicação da base de dados poderia ser levada a cabo.

Assumindo uma arquitetura *master/slave*, os dados armazenados na base de dados instalada na máquina *master* seriam copiados automaticamente para uma ou mais máquinas - os *slaves*. No PostgreSQL, isto seria uma *streaming replication*, que é unidirecional (do *master* para o(s) *slave(s)*), porque apenas o servidor *master* tem permissões de escrita de dados, transmitindo-os por WAL *streaming* para os *slaves*. Contudo, esta replicação oferece a possibilidade de os acessos para leitura serem distribuídos por vários *slaves*, o que acarreta vantagens em termos de escalabilidade, mas também para situações de *failover* ou até para a análise de dados fora do *master*, não o sobrecarregando.

Para configurar a replicação, há alterações em vários ficheiros que teriam de ser feitas. Em primeiro lugar, seria necessário configurar o *master* e os *slaves* para a *streaming replication*, editando o ficheiro postgresql.conf:

```
listen_address = '*'

wal_level = hot_standby

max_wal_senders = 3

checkpoint_segments = 8  #por exemplo
```

```
wal_keep_segments = 8    #por exemplo

hot_standby = on
```

Aqui, além de permitir WAL *streaming* e editar parâmetros específicos da mesma, está-se a definir o conjunto de endereços que podem aceder à base de dados. De seguida, ter-se-ia de, no ficheiro pg_hba.conf do *master*, adicionar uma entrada semelhante à seguinte para cada *slave*, substituindo o 'ip-slave' pelo IP real:

```
host  replication  postgres       ip-slave    md5
```

No mesmo ficheiro do(s) *slave(s)*, adicionar-se-ia algo semelhante, mas com o endereço do *master*:

```
host  replication  postgres       ip-master   md5
```

Este seria o *setup* inicial. O próximo passo seria transferir os dados do *master* para as bases de dados alojadas nas máquinas dos *slaves,* correndo uma script como esta:

```
sudo service postgresql stop

sudo -u postgres rm -rf /var/lib/postgresql/9.5/main/*

sudo -u postgres pg_basebackup -h ip-master -D
/var/lib/postgresql/9.5/main -U postgres
```

Para que o *slave* consiga conectar-se ao *master*, seria também necessário um ficheiro de recuperação, criado como /var/lib/postgresql/9.5/main/recovery.conf:

```
standby_mode = 'on'
primary_conninfo =
      'host=ip-master port=5432 user=postgres password=password'
trigger_file = '/tmp/failover.trigger'
```

Por último, antes de iniciar a base de dados no *slave* com `service postgresql start`, seria conveniente confirmar a correção dos privilégios na diretoria do PostgreSQL:

```
chmod -R g-rwx,o-rwx /var/lib/postgresql/9.5/main/

chown -R postgres.postgres /var/lib/postgresql/9.5/main/
```

Ansible

Como veremos de seguida, para fazer o deployment da aplicação Odoo, foi utilizado, entre outros, o Ansible, mais propriamente sua configuração - o ansible-playbook. Este permite executar, em sequência, de modo remoto, várias tarefas, como instalar os pacotes necessários para o bom funcionamento do software, na sua versão mais atualizada, não necessitando de mudança de código, caso seja

lançada uma nova versão. O ansible-playbook, para além de ser uma ferramenta muito versátil, é ideal para executar num grande número de máquinas, apenas necessitando de receber os endereços (especificados por username@ip, username da máquina na qual executará o playbook, e ip da máquina para o qual será executado o mesmo playbook), podendo mesmo agrupar ip's, executando assim um playbook específico, indicando que o alvo do playbook será o grupo, através de --limit grupo. Podem, também, ser dados argumentos ao playbook, modificando certos pontos, como portas na qual uma aplicação estará à escuta, ou o nome de uma imagem a ser construída.

Deployment da aplicação num ambiente de *Cloud Computing*

A instalação da aplicação pretendia-se automática e configurável, para os diversos componentes da aplicação, devendo ser efetuada no menor número de passos possíveis. Como tal, neste processo de instalação são utilizadas três ferramentas fundamentais: Vagrant, Docker e Ansible.

A plataforma de *cloud* da Google (cloud.google.com) foi escolhida para anfitrião das máquinas virtuais de teste deste processo automático de instalação, o que trouxe desde logo uma reflexão acerca dos IPs das máquinas a virtualizar. A criação de uma nova instância de VM na cloud leva à atribuição de dois IPs: um interno e outro externo. O IP externo a atribuir a uma dada VM pode ser reservado, tornando-o estático, porém, a sua inutilização acarreta custos. Sabendo que o período de utilização deste IP seria bastante curto, evitou-se esta burocracia. Assim, o processo de *deployment* requer o acesso à plataforma para consulta dos IPs atribuídos às VM's criadas.

Esta questão dos IPs aleatórios teve impacto na ideia inicial de tornar o *deployment* totalmente automático após a execução de um **`vagrant up.`** No entanto, não foi abandonada a ideia de usar Vagrant para dar início à automatização da instalação. Para tal, recorreu-se a um *plugin* - https://github.com/mitchellh/vagrant-google - que adiciona um *provider* ao Vagrant que lhe permite controlar e provisionar instâncias no *Google Computer Engine.* Isto torna-se útil porque, assim, sabendo de antemão a configuração pretendida para as instâncias de VM a criar, basta alterar o número de instâncias a virtualizar e, com um **`vagrant up`**, todas as instâncias são criadas na *cloud*. Este é o ficheiro usado para tal:

```
# define the number of Google Cloud instances to create
$NUM_VM = 4

Vagrant.configure("2") do |config|
   config.vm.box = "google/gce"
   # define and provide as much VM's as specified in $NUM_VM
   (1..$NUM_VM).each do |i|
      config.vm.define "vm#{i}" do |node|
        node.vm.provider :google do |google|
          google.name = "vm#{i}"
        end
      end
    end
```

```
# Google Cloud VM instances properties
   config.vm.provider :google do |google, override|
      google.google_project_id = "group6-182510"
       google.google_client_email = "810361359414-compute@developer.gserviceaccount.com"
       google.google_json_key_location = "~/Documents/SDB/group6-key.json"
      google.zone="europe-west1-d"
      google.image = "ubuntu-1604-xenial-v20171212"
      google.machine_type = "f1-micro"

      # SSH username
      override.ssh.username = "jccoelho"
      # SSH private key path
      override.ssh.private_key_path = "~/.ssh/id_rsa"
   end
# Install docker in the machines
   config.vm.provision "shell", inline: <<-SHELL
     curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
     sudo apt-key fingerprint 0EBFCD88
                              sudo      add-apt-repository      "deb      [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
     sudo apt-get -y update
     sudo apt-get -y install docker-ce
     sudo curl -L
https://github.com/docker/compose/releases/download/1.18.0/docker-compose-'uname
-s'-'uname -m' -o /usr/local/bin/docker-compose
     sudo chmod +x /usr/local/bin/docker-compose
    SHELL
end
```

Este ficheiro é, então, responsável por automatizar a criação das VM's que recebem os diversos componentes da aplicação. Com as máquinas virtuais disponíveis para albergar os componentes necessários para a aplicação, o passo seguinte no processo de *deployment* foi abordado anteriormente e consiste em consultar os IPs externos atribuídos na *cloud* às máquinas virtuais, inserindo-os no ficheiro /etc/ansible/hosts, colocando como prefixo de cada ip, o *username* de acesso à *google cloud*. Neste, os IPs são agrupados consoante a sua função, dando origem a um ficheiro com o seguinte aspeto:

```
[haproxy]

username@ip_externo_haproxy

[odoo]

username@iip_externo_odoo1

username@ip_externo_odoo2

...

[db]

username@ip_externo_dbr
```

No ficheiro acima descrito, estão especificados três grupos, podendo uma máquina pertencer a mais que um grupo:

[haproxy] - A máquina que suportará o serviço de *load balancing* fornecido pelo software HAProxy.

[odoo] - Todas as máquinas que suportarão os containers que estarão a correr a imagem do software odoo. Esta será a camada que será replicada.

[db] - A máquina onde será instalado a base de dados PostgreSQL. Neste caso, apenas uma máquina foi atribuída para a base de dados, porque não foi esta camada que se optou por replicar.

Numa primeira fase, tentou-se realizar a implementação da replicação através de uma das ferramentas disponibilizadas pelo Docker, o Docker Swarm. Num contexto onde são utilizados containers, a ferramenta anteriormente enunciada fornece mais garantias, realizando internamente o balanceamento de carga. Apesar das vantagens que apresenta, após várias tentativas, o grupo não conseguiu configurar adequadamente a ferramenta, com problemas ao nível da conexão à base de dados. A alternativa passou pela utilização do software HAProxy.

Configuração/*Deployment* da base de dados

Anteriormente, foram enunciados alguns pontos de configuração que permitiram a ligação entre a base de dados e a camada aplicacional. Alguns parâmetros do ORDBMS (*object-relational database management system*) desempenham um papel fulcral nesta conexão. Por default, o PostgreSQL apenas aceita conexões com origem no localhost. Dado que o deploy da base de dados é realizado numa máquina diferente de onde as várias instâncias aplicacionais se encontram, é necessária uma configuração diferente da default. Após uma breve análise, foram identificados os parâmetros capazes de assegurar a permissão de ligações com origem em máquinas exteriores. A seguir são enunciadas as alterações realizadas:

Ficheiro de configuração postgresql.conf:
listen_addresses = 'localhost' -> listen_addresses = '*'

Ficheiro de configuração pg_hba.conf:

host all all 127.0.0.1/32 md5 -> host all all 0.0.0.0/0 md5

Para além das configuração do ORDBMS, também são necessárias pequenas alterações dos ficheiros utilizados pelo Dockerfile para o deploy do Odoo, ao nível aplicacional. No ficheiro entrypoint.sh são definidas variáveis de ambiente utilizadas na conexão à base de dados como:

$HOST

$PORT

$USER

$PASSWORD

A variável "$HOST" deverá ser ajustada de acordo com o ip da máquina que suportará a base de dados.

Para automatizar o *deploy* da base de dados, foi utilizado o software Ansible, criando um *playbook*, "postgres.yml", onde será instalada a base de dados PostgreSQL e os pacotes necessários para a realização do *deployment*. Será criado o utilizador "odoo", com permissões de criação de bases de dados dentro do postgres, com a sua respetiva password, modificando também os ficheiros de configuração pg_hba.conf e postgresql.conf, conforme descrito acima, recomeçando a base de dados de seguida, de modo a adotar as novas configurações.

Com o playbook, para instalar e configurar a base de dados PostgreSQL, apenas terá de executar o seguinte comando:

Ex: **ansible-playbook postgres.yml --limit db**

Assim, a base de dados está pronta a receber e a responder a pedidos da aplicação web Odoo.

Configuração/*Deployment* do balanceador

No que diz respeito ao balanceador de carga, a escolha recaiu sobre o software HAProxy. De modo a que o balanceador inicie automaticamente sempre que a máquina onde se encontra reinicie, é necessário adicionar a flag "ENABLED=1" ao ficheiro de configuração HAProxy (/etc/default/). Para além desta simples configuração, é igualmente necessário estabelecer alguns parâmetros, agora no ficheiro haproxy.cfg (/etc/haproxy/), como podemos verificar no seguinte exemplo:

listen odoo

```
bind *:8082
mode http
balance leastconn
option httpclose
option forwardfor
cookie SERVERNAME insert indirect nocache
server odoo1 10.0.0.101:8069 cookie s1 check
server odoo2 10.0.0.102:8069 cookie s2 check
```

Como facilmente é perceptível, é presença desta configuração indica a porta onde ocorre o *bind* (onde serão recebidos os pacotes, para se proceder à distribuição pelos servidores web da aplicação), o tipo e a forma como os pedidos devem ser distribuídos (mode and balance), e a lista de *containers* para onde devem ser encaminhados os pacotes (últimas duas linhas da configuração). Outras opções podem ser adicionadas como, por exemplo, a utilização de cookies para efeitos de teste.

Para automatizar o processo de *deployment* do software HAProxy, foi utilizado novamente o software Ansible, em semelhança ao *deployment* da base de dados PostgreSQL. Sendo assim, foi criado um playbook "haproxy.yml" que instala o software, acrescenta a flag "ENABLED=1" ao ficheiro de configuração HAProxy (/etc/default/), cria o ficheiro de configuração haproxy.cfg (/etc/haproxy/), iniciando sem nenhum *container* pré-definido para onde devem ser encaminhados os pacotes.

De modo a acrescentar *containers* ao ficheiro haproxy.cfg, foi criado um novo *playbook* e um *script* "haproxy_scr.sh" em shell. O playbook adiciona um *container* ao ficheiro de configuração (com a composição "server odoo <ip>:<port> cookie s check"), e o script torna possível a utilização do playbook em conjunto com um ficheiro que contém todos os IPs internos das máquinas que albergam os containers e as portas nas quais os mesmos estão à escuta, executando o playbook de modo a adicionar todos esses ips/portas ao ficheiro haproxy.cfg. É de salientar que terá de modificar o *script*, na linha 13 (poderá mudar também a linha 12, mas apenas serve como notificação do comando a ser executado, não intervém na execução do mesmo), modificando, a seguir a "-- limit", o *username* e o IP da máquina onde está a ser executado o software HAProxy. Sendo assim, caso possua bastantes *containers* a serem adicionados ao ficheiro de configuração, basta criar um ficheiro com as especificações mencionadas acima, e executar "./haproxy_scr.sh ficheiro.txt". A configuração do ficheiro utilizado em conjunto com o script deverá ser a seguinte:

```
ip1 porta1
ip1 porta2
ip2 porta1
ip3 porta2
```

Para a instalação do balanceador de carga HAProxy, apenas terá de executar o comando:

ansible-playbook haproxy.yml --limit haproxy

Para adicionar servidores odoo ao balanceador de carga, deverá criar e preencher o ficheiro "hosts.txt" (nome aleatório) e executar o seguinte comando:

./haproxy_scr.sh hosts.txt

Após executados, o software HAProxy estará instalado e configurado, na máquina desejada.

Configuração/Deployment do software Odoo

Para usufruir das funcionalidades disponibilizadas pelo software Odoo, necessitamos ter pelo menos uma aplicação Odoo a correr, conectado à base de dados PostgreSQL mencionada acima. O software de balanceamento de carga (HAProxy) ajuda a distribuir os pedidos à aplicação, caso haja mais que servidor ou container a corrê-la.

Para suportar a aplicação web Odoo, utilizamos o software Docker, de modo a ser possível a criação de vários containers a correr a imagem relativa à aplicação, dentro da mesma máquina, melhorando a escalabilidade. Aliás, mesmo que numa dada VM exista apenas um container, isto acarreta vantagens, que contribuíram decisivamente para a nossa escolha. Um container consiste num ambiente virtual computacionalmente leve, que encapsula processos e recursos (CPU, disco, memória, etc), isolando-os dos restantes containers ou do *host*. A isto estão associadas vantagens como a possibilidade de termos diferentes versões isoladas do Odoo a correr na mesma máquina (já mencionado), a facilidade de "empacotar" num só container todas as dependências associadas a uma aplicação - útil quando, por exemplo, se pretende alterar um software de monitorização e, assim, a desinstalação total deste está apenas dependente da remoção do container - e a portabilidade entre servidores, permitindo conservar a mesma instância do Odoo se se pretender migrar o container para uma máquina diferente. Além das melhorias em termos aplicacionais, existem também aquelas associadas a custos mais reduzidos e à melhor utilização dos recursos.

De modo a ser automatizar o *deployment* da aplicação Odoo pelas máquinas desejadas, foi utilizado novamente o software Ansible. Para tal, foi criado um playbook "odoo.yml", que constrói a imagem relativa à aplicação e corre o container com a mesma imagem a correr na máquina, expondo para a máquina *host* do container a porta, especificada pelo criador, que receberá os pedidos dos utilizadores da aplicação.

De modo a que imagem, criada relativa ao software Odoo, saiba onde se situa a base de dados, foi criada uma pasta própria para a construção da imagem, especificando no ficheiro "entrypoint.sh", no atributo $HOST, o IP da máquina onde a base de dados estará a correr. O Dockerfile criado para a construção da imagem, expõe a porta 8069 e 8071, visto que estas são as portas, definidas pelo software, onde aceitarão pedidos. Assim, o playbook copiará a pasta odoo/ para a máquina, para inicializar o processo de *deployment*.

O utilizador deverá modificar à mão, o atributo "$HOST", do ficheiro entrypoint.sh, situado dentro da pasta "odoo", colocando o ip interno da máquina onde foi executado o playbook "postgres.yml", descrito acima.

Assim, após especificar no ficheiro hosts do ansible (/etc/ansible/), poderá executar o playbook nas máquinas especificadas pelo grupo "odoo". Deverá também especificar a porta através da qual, o *container* a ser criado irá enviar os pacotes para a máquina host. Esta especificação será feita através de argumentos extra, funcionalidade disponibilizada pelo ansible-playbook, indicando a porta através do argumento "port". Assim, para executar o playbook, deverá executar o seguinte comando:

ansible-playbook odoo.yml --limit odoo --extra-args "port=8069"

Neste caso, será criado um container em todas as máquinas especificadas no grupo "odoo", em que esses mesmos containers encaminharão os pacotes do software para a máquina host, através dessa porta.

Caso queira criar um container em apenas uma máquina, deverá executar especificando o endereço no local do grupo odoo:

ansible-playbook odoo.yml --limit ubuntu@10.0.0.101 --extra-args "port=8069"

Assim, apenas será criado um container, na máquina cujo ip externo é 10.0.0.101, conectado à porta 8069 da máquina *host.*

Para uma maior automatização do processo deployment, foi criado um *bash script*, "deploy.sh", que executa todos os passos acima descritos, sequencialmente, após a configuração do ficheiro entrypoint.sh, /etc/default/hosts, haproxy.sh e hosts.txt, como descrito acima, executando de seguida o seguinte comando:

./deploy.sh hosts.txt 8069 8071 8072

Neste caso, iriam ser criados em todas as máquinas do grupo odoo, três containers, conectados à porta 8069, 8071 e 8072.

Foi criado um ficheiro “README.md”, explicando os passos descritos acima mais resumidamente.

Quais as ferramentas de monitorização utilizadas, as métricas escolhidas e a necessidade das mesmas.

No que diz respeito às ferramentas responsáveis pela monitorização do sistema, a escolha recaiu sobre a utilização do ElasticSearch, Kibana e Beats. Foi escolhida uma arquitectura baseada, em certo modo, numa monitorização centralizada, com apenas uma máquina a suportar as duas ferramentas responsáveis por armazenar e apresentar os dados (ElasticSearch e Kibana).

Para além destes componentes, em todas as restantes máquinas que fazem parte da camada aplicacional ou da camada de dados da plataforma Odoo, foram integradas vertentes da ferramenta Beats, mais especificamente, a MetricBeat e a PacketBeat. No que diz respeito à MetricBeat, disponibiliza vários módulos que permitem a coleção de diferentes métricas capazes de fornecer informação sobre o estado de um determinado servidor, sistema operativo, serviço e, até mesmo, container.

Na nossa opinião, no contexto do problema, faz sentido monitorizar o sistema como um todo. Os serviços presentes em todas as máquinas, essenciais para o bom funcionamento da plataforma Odoo, não partilham o sistema operativo com mais nenhum serviço crítico, havendo uma grande proximidade entre o nível do sistema com o nível da aplicação. Por outro lado, no que diz respeito aos containers, estes partilham o Kernel do seu sistema operativo base, sendo esta outra justificação para a monitorização das características gerais da máquina. Assim, no caso específico do MetricBeat, foi utilizado o módulo correspondente a métricas ao nível do sistema, como por exemplo, a utilização do cpu, memória e disco.

As métricas escolhidas permitem-nos verificar se o sistema se encontra ou não em sobrecarga e consequentemente decidir quanto à necessidade de replicar a camada aplicacional, aspectos fundamentais para o principal ponto do trabalho, a escalabilidade. De modo a justificar todos os pontos enunciados anteriormente, podemos dar como exemplo a máquina responsável pelo base de dados, onde a métrica indicativa do estado da memória (ex: indicação da percentagem de memória livre e utilizada) é bastante importante. Como sabemos, a quantidade de memória total, livre e usada tem influência direta na performance da base de dados aquando da utilização do Odoo.

No que diz respeito ao PacketBeat, este permite avaliar o tráfego presente na infraestrutura onde é instalado, verificando os vários pacotes enviados e recebidos pelos containers a correr a aplicação Odoo. Dado que é completamente passivo, não intervêm na performance da aplicação, nem aumenta o overhead dos pacotes. No caso em específico da aplicação Odoo, a utilização desta ferramenta pode ser útil, por exemplo, na medição dos tempos entre os vários pedidos/respostas trocados entre a base de dados e as várias instâncias da aplicação. É assim capaz de fornecer informação da qual se pode retirar algumas conclusões relativamente ao estado dos componentes em causa.

A monitorização imposta pelo grupo incidiu assim, sobretudo, na análise do estado das máquinas que suportam a aplicação, de modo a ser possível uma opção mais sensata na hora de tomar decisões, no que diz respeito à problemática da escalabilidade. Uma análise mais pormenorizada de algum dos componentes como, por exemplo, informação do número de conflitos e de hit/read's de blocos relativamente à base de dados, poderia ter sido feita, mas no contexto do problema achou-se que não faria sentido.

Assim, com estes dados, poder-se-iam realizar estudos sobre a forma mais económica de albergar os vários containers.

O processo de instalação e configuração das aplicações de monitorização faladas foi automatizado através de playbooks. Como tal, foi criado um playbook, "elastic-kibana.yml", para automatizar a instalação e configuração do elasticsearch e kibana, na mesma máquina, bastando apenas fornecer o ip interno da máquina a ser instalada, como argumento extra.

ansible-playbook elastic-kibana.yml --limit ubuntu@35.195.91.186 --extra-vars "ip=10.132.0.5"

Neste caso, foi executado o playbook para a máquina cujo ip externo é 35.195.91.186 e cujo ip interno é 10.132.0.5.

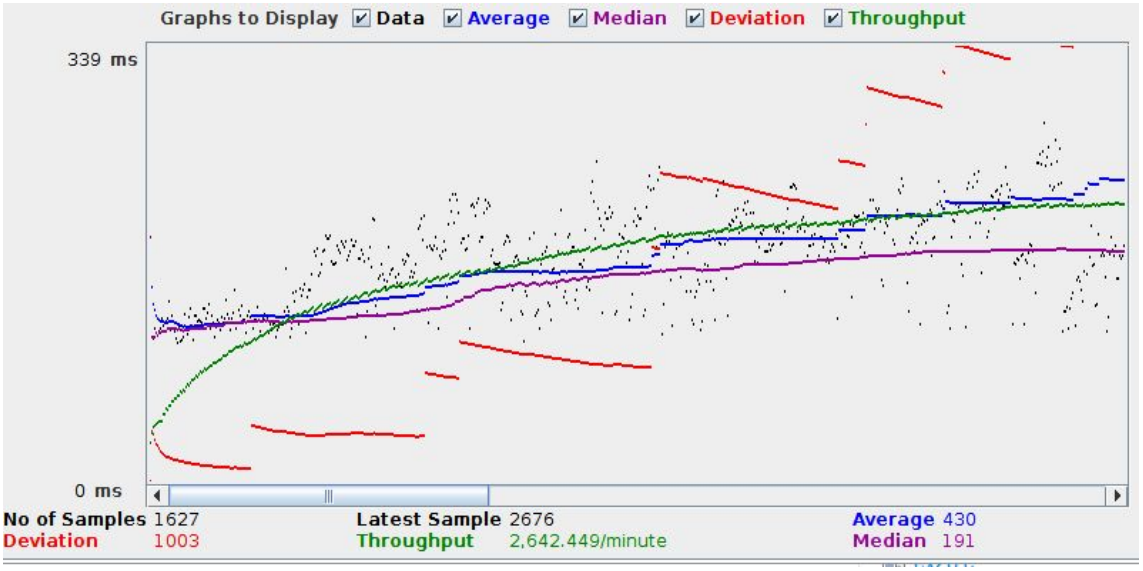
A instalação do packetbeat e metricbeat também foi automatizada, criando os playbooks "packet_beat.yml" e "metric_beat.yml" respetivamente, sendo necessário passar como argumento extra, na sua execução, o ip interno da máquina onde foi executado o playbook "elastic-kibana.yml".

Testes de performance à aplicação

Para criar e aplicar testes de performance ao Odoo, foi escolhida a Apache JMeter - http://jmeter.apache.org/ -, uma aplicação Java desenhada para testar

aplicações Web dinâmicas. Esta permite simular cargas pesadas num servidor ou grupo de servidores, obtendo valores do tempo de resposta da aplicação ao pedido HTTP, controlando fatores como o número de clientes num dado *ramp-up time*. Com isto, conseguem-se obter dados em função do tempo que permitem obter o throughput da aplicação, além de dados matemáticos resultantes da análise de múltiplos exemplos, como são a mediana e o desvio padrão. A figura abaixo representa um gráfico construído pelo Apache JMeter, onde se encontram representados os dados anteriormente falados, que resulta da aplicação do seguinte teste:

Encaminhou-se ao servidor do balanceador uma sequência de pedidos HTTP, em *loop*, com um *ramp-up time* de 60s e 60 clientes preparados nesse mm período, tendo sido esta a resposta dada pelos *webservers*:



Para ser possível ter acesso ao GUI do Apache JMeter, é necessário correr o software na máquina local, porque os terminais da Google Cloud não são gráficos. Uma vantagem do GUI é ser possível criar e editar com facilidade o teste a realizar, podendo corrê-lo de imediato. Para que seja possível avaliar a performance de máquinas remotas a partir da local, endereça-se o pedido HTTP a realizar ao IP e porta da VM devida. Naturalmente, antes é necessário instalar a aplicação na máquina nativa, algo que procuramos, também, automatizar, através de mais um playbook, “jmeter.yml”.

Conclusão

Para este trabalho prático foi escolhido o software *open-source* Odoo, desenvolvido pela Odoo S.A., que cumpre os requisitos colocados no enunciado - duas ou mais camadas lógicas, interface Web, sistema de base de dados e elasticidade em pelo menos uma camada -, e relativamente ao qual se irá, nesta primeira fase do trabalho, analisar a arquitetura, funcionalidades e operações críticas.

O Odoo apresenta-se como um software muito acessível e diversificável, pelos vários módulos existentes, fornecendo uma panóplia de funcionalidades às empresas que utilizam este software, mas também a possibilidade de, caso necessitem de um módulo com novas funcionalidades, desenvolverem um novo módulo, configurando o seu software à sua medida.

Sintetizando a arquitetura do Odoo, deve-se referir as três camadas que a compõem: a camada de apresentação, responsável pela interface gráfica apresentada ao cliente, a camada de negócio, que contém a maioria do peso de processamento do software, uma vez que contém o servidor da aplicação, e a camada de dados, que possui todos os dados do software, guardados numa base de dados PostgreSQL.

No que diz respeito à segunda fase do trabalho, a interação com a Google Cloud Plataform decorreu sem qualquer problema, tendo sido realizadas as configurações necessárias. A utilização de um plugin especial, integrando na ferramenta Vagrant, veio permitir o deploy das várias máquinas na plataforma, de forma automatizada.

Realizada uma breve análise e correção à arquitectura projetada na fase anterior, foram tomadas decisões no que diz respeito à utilização de containers, no processo de deployment dos vários componentes da plataforma Odoo. Posteriormente, após decidida a camada a sofrer replicação, foram sentidas algumas dificuldades com a utilização da ferramenta *Docker Swarm*. Apesar de ter sido conseguida a replicação de várias instâncias da camada aplicacional através do cluster construído pela ferramenta, a conectividade com a base de dados não estava a ser conseguida e, ao fim de algumas tentativas, o grupo decidiu mudar de estratégia. A utilização do balanceador de carga HAProxy, capaz de distribuir os vários pedidos por diferentes instâncias aplicacionais do Odoo (presentes em várias máquinas e assentes em containers) aliada à existência de conectividade com uma base de dados alojada numa máquina virtual, veio permitir ao grupo de trabalho implementar a arquitetura desejada.

O ponto anteriormente referido teve consequências negativas nas seguintes fases do trabalho, dado que uma grande porção de tempo foi dispendida na configuração de uma ferramente que, no final, não foi utilizada. Assim, foi realizado

um pequeno número de testes, mas capazes de ilustrar as operações base propostas no enunciado.

Finalmente, no que diz respeito à monitorização e performance, foram tidas em consideração as melhores métricas e ferramentas capazes de caracterizar e fornecer informação passível de ser utilizada no futuro, por exemplo, em decisões relativas ao processo de deployment, caso a plataforma tenha que lidar com problemas de escalabilidade.

Em todas as etapas anteriormente enunciadas foi conseguido um elevado grau de automatização das tarefas, tendo sido assim alcançado o principal objectivo do trabalho.