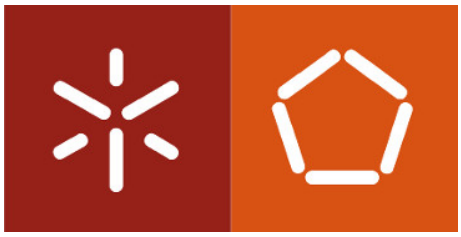
System Deployment and Benchmarking

Grupo nr.	5
a85370	Hugo Matias
a84802	João Nuno Abreu
a84577	José Pedro Silva
a84783	Pedro Rodrigues
a84485	Tiago Magalhães



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Contents

1 Introdução

No âmbito da Unidade Curricular de System Deployment and Benchmarking, foi-nos proposta a realização de um trabalho que consiste na automatização do processo de deployment e de benchmarking para uma aplicação à escolha do grupo, tendo este optado pela aplicação **GitLab**.

Numa primeira abordagem do projeto, ainda não tendo a aplicação escolhida, foi feita a análise de um conjunto de aplicações fornecidas pelos docentes.

Unanimemente o grupo optou pela aplicação **GitLab**, uma vez que é a aplicação com a qual estamos mais familiarizados e também por ser uma aplicação mundialmente conhecida, tendo por isso, mais recursos *online* que nos facultam e facilitam o processo de pesquisa.

2 GitLab

2.1 O que é o GitLab

> *GitLab is a complete DevOps platform, delivered as a single application. This makes GitLab unique and creates a streamlined software workflow, unlocking your organization from the constraints of a pieced together toolchain. Learn how GitLab offers unmatched visibility and higher levels of efficiency in a single application across the DevOps lifecycle.*

2.2 Arquitetura e componentes

Nesta secção será feita uma abordagem à arquitectura da aplicação, bem como às suas componentes e à forma como elas se relacionam, de maneira a identificar as principais operações e pontos de configuração.

2.2.1 Modelo do sistema

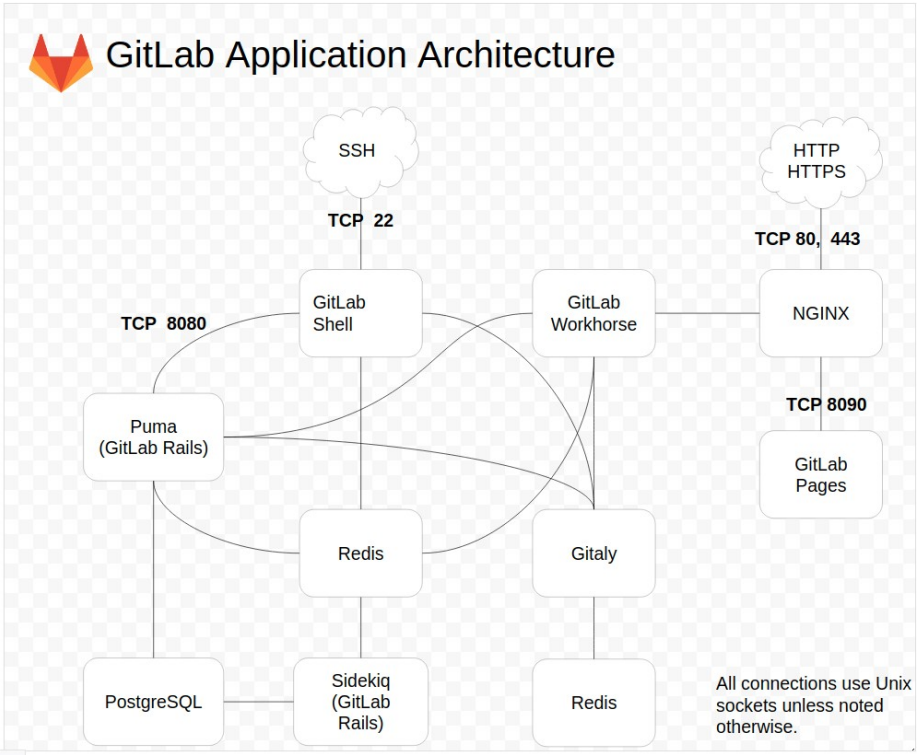


Figure 1: Modelo do Sistema

2.2.2 Descrição da arquitetura

Através do modelo do sistema podemos observar que existem 10 componentes principais sendo estas:

- **NGINX Ingress**: É um servidor *web*, que neste sistema atua como *proxy* entre as conexões dos utilizadores pela interface *web* e a componente GitLab Workhorse;
- **GitLab Workhorse**: Componente que lida com pedidos "grandes" de HTTP como *downloads*, ficheiros e *push/pull requests*;
- **Puma**: Servidor *web* atavés do qual são servidas as páginas *web*.
- **GitLab Shell**: Funciona a nível aplicacional semelhante ao GitLab Workhorse, no entanto lida com os utilizadores que estão a utilizar ssh.
- **Gitaly**: É um serviço desenvolvido pelo GitLab que fornece RPC(*Remote procedure call*) para repositórios Git. Sem isto, as componentes não podem ler ou escrever Git data.
- **PostgreSQL**: Base de dados utilizada pelo GitLab para armazenar utilizadores, permissões, meta-dados, entre outros.
- **Sidekiq** : Serviço que atua em *background* e processa os *jobs* do Redis, permitindo que o GitLab ofereça um rápido ciclo de repostas e pedidos.
- **Redis**: base de dados não persistente utilizada pelo Sidekiq para *job information*, *metadata*, e *incoming jobs*.

2.3 Formas de comunicação

Todas as conexões da aplicação são feitas através de sockets `Unix`.

Ao nível da camada de transporte, as comunicações são feitas utilizando o protocolo **TCP/IP**. A nível aplicacional, podem ser usados os protocolos **HTTP** e **HTTPS** (em comunicação com o `NGINX`) ou **SSH** (em comunicação com `GitLab Shell`).

3 Padrões de distribuição

O GitLab pode tanto ser configurado num único servidor como também escalado para servir mais utilizadores. As equipas de qualidade e suporte do GitLab construíram e verificaram diferentes padrões de distribuição recomendados, consoante o número de utilizadores (todos os testes foram efetuados através da `GitLab's Performance Tool`).

Para instâncias do GitLab com menos de 2000 utilizadores, é recomendado o uso da configuração padrão, instalando o GitLab numa única máquina para minimizar os custos de manutenção e recursos.

Para instâncias com mais de 2000 utilizadores, é recomendado escalar as componentes do GitLab em múltiplas máquinas. A adição de máquinas aumenta o desempenho e a escalabilidade do GitLab, no entanto, é necessário ter em atenção, o equilíbrio entre desempenho, escalabilidade e custo de manutenção e recursos.

Segundo os resultados do estudo referido acima, a solução ideal para uma instância com mais de 50000 utilizadores seria a seguinte:

Service	**Nodes**
External load balancing node	1
Consul	2
PostgreSQL	3
PgBouncer	3
Internal load balancing node	1
Redis - Cache	3
Redis - Queues / Shared State	3
Redis Sentinel - Cache	3
Redis Sentinel - Queues / Shared State	3
Gitaly	2 (minimum)
Sidekiq	4
GitLab Rails	12
Monitoring node	1

A esta distribuição, podem ser ainda adicionadas algumas funcionalidades do GitLab para aumentar a escabilidade e desempenho:

- **Database Load Balancing** - Balanceamento de queries de leitura em diferentes instâncias da base de dados. Isto resulta numa diminuição da carga na instância principal da base de dados e, consequentemente, aumenta a responsividade.
- **GitLab Geo** - Cria um clone completo do GitLab server, apenas com privilégios de leitura, que se mantém sempre sincronizado. Este add-on aumenta a velocidade das operações `clone` e `fetch`. A GitLab publicou um vídeo onde explica e exemplifica os benefícios deste serviço.

Para a distribuição do sistema tivemos em conta a escalabilidade a nível de performance e disponibilidade. Para o sistema proposto, iremos isolar e virtualizar as seguintes componentes: NGINX, uma vez que é a das componentes que está em contacto com o cliente directamente e é necessário garantir a sua disponibilidade (replicação), PostgreSQL, Redis, e serviço de email que ao contrário das outras componentes ficará noutro *container* mas na mesma máquina da aplicação, uma vez que é recomendação do GitLab. Ao usar *conatainers* garantimos **escalabilidade**, **isolamento** dos serviços e **portabilidade**. Assim a arquitetura proposta é a seguinte (**Figura 2**):

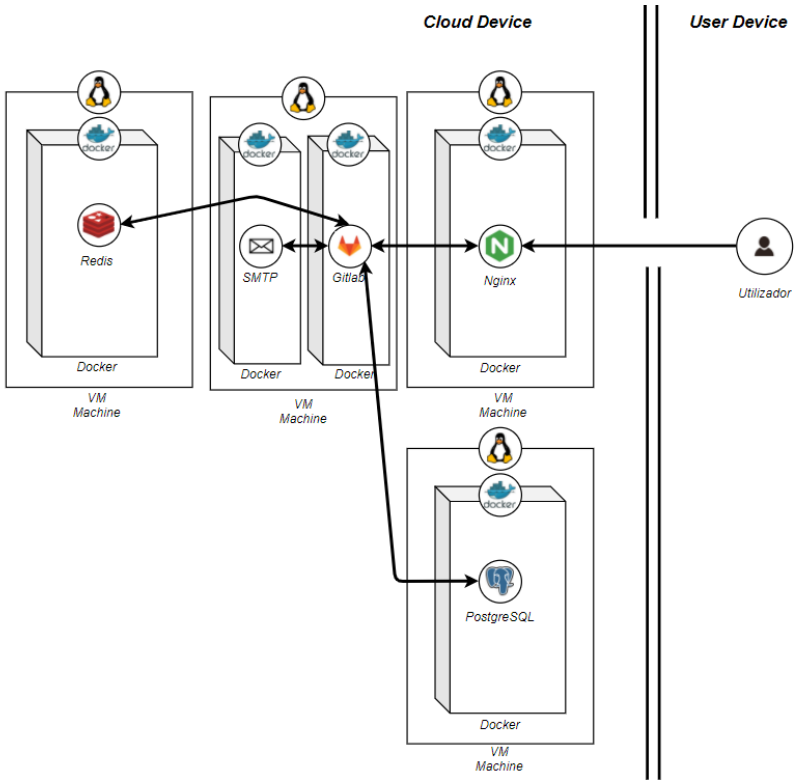


Figure 2: Arquitetura proposta para o sistema

4 Pontos de configuração

Para configurar e instalar uma versão simples do GitLab (uma máquina para todos os componentes), é apenas necessário seguir as instruções da documentação do GitLab.

Primeiramente, é necessário criar uma máquina que cumpra os requisitos mínimos de hardware:

- 4 cores
- 4GB RAM

Após a verificação do hardware da máquina, é necessário seguir os passos apresentados na documentação.

Como exemplo, foi utilizada uma máquina `ubuntu 20.04`. Neste caso, o processo de instalação mais simples é seguir a instalação recomendada (via pacakge). Para tal, foi necessário instalar algumas dependências:

```
$ sudo apt-get update
$ sudo apt-get install -y curl openssh-server ca-certificates tzdata
```

Para serviço de email, é necessário instalar o `postfix`:

```
$ sudo apt-get install -y postfix
```

Depois, basta adicionar o pacote do GitLab ao repositório e instalar:

```
$ curl https://packages.gitlab.com/install/repositories/gitlab/
gitlab-ee/script.deb.sh | sudo bash

$ sudo EXTERNAL_URL="10.0.0.102" apt-get install gitlab-ee
```

No final, o serviço GitLab, fica disponível na página `http://10.0.0.102`.

4.1 Base de dados externa

A instalação apresentada acima é a mais simples possível, com todas as componentes presentes na mesma máquina, no entanto, é possível configurar o GitLab para utilizar alguns componentes em máquinas externas.

Para utilizar a base de dados numa máquina externa, foi necesssário criar uma nova máquina e instalar e iniciar o serviço `postgreSQL`:

```
$ sudo apt-get install -y postgresql postgresql-client libpq-dev
postgresql-contrib

$ sudo service postgresql start
```

Depois da instalação, foi necessário configurar a base de dados:

```
$ sudo -u postgres psql -d template1 -c "CREATE USER vagrant CREATEDB;"

$ sudo -u postgres psql -d template1 -c "CREATE EXTENSION
IF NOT EXISTS pg_trgm;"

$ sudo -u postgres psql -d template1 -c "CREATE EXTENSION
IF NOT EXISTS btree_gist;"

$ sudo -u postgres psql -d template1 -c "CREATE DATABASE
gitlabhq_production OWNER vagrant;"

$ sudo -u postgres psql -d template1 -c "ALTER USER vagrant
WITH PASSWORD 'password'"
```

Após a configuração, foi necessário ativar o acesso remoto ao `postgreSQL server`:

1. Adicionar a linha `listen_addresses = '*'` no fim do ficheiro `/etc/postgresql/10/main/postgresql.conf`
2. Adicionar as seguintes linhas no fim do ficheiro `/etc/postgresql/10/main/postgresql.conf`:

```
host    all     all             0.0.0.0/0                   md5
host    all     all             ::/0                        md5
```

3. Reiniciar o servidor `postgreSQL` através do comando:

```
$ systemctl restart postgresql.service
```

Com a base de dados criada e acessível remotamente, era necessário configurar o GitLab para aceder a esta base de dados. Para isto, foram adicionadas as seguintes linhas ao ficheiro `/etc/gitlab/gitlab.rb`:

```
postgresql['enable'] = false
gitlab_rails['db_adapter'] = "postgresql"
gitlab_rails['db_encoding'] = "unicode"
gitlab_rails['db_username'] = "vagrant"
gitlab_rails['db_password'] = "password"
gitlab_rails['db_host'] = "10.0.0.103"
```

No final, apenas foi necessário reiniciar o serviço GitLab, de modo a que as altearações surtissem efeito:

```
$ sudo gitlab-ctl reconfigure
```

4.2 Serviço REDIS externo

Foi também testado o uso de um serviço REDIS externo à máquina do sistema. Para isso, foi criada uma VM e instalado, na mesma, o serviço REDIS:

```
$ sudo apt-get install redis-server
```

Depois de instalado, foi necessário, então, configurar o REDIS:

```
$ sudo cp /etc/redis/redis.conf /etc/redis/redis.conf.orig
$ sudo sed 's/^port .*/port 0/' /etc/redis/redis.conf.orig
    | sudo tee /etc/redis/redis.conf
$ echo 'unixsocket /var/run/redis/redis.sock'
    | sudo tee -a /etc/redis/redis.conf
$ echo 'unixsocketperm 770' | sudo tee -a /etc/redis/redis.conf
$ sudo mkdir -p /var/run/redis
$ sudo chown redis:redis /var/run/redis
$ sudo chmod 755 /var/run/redis
$ if [ -d /etc/tmpfiles.d ]; then
        echo 'd  /var/run/redis  0755  redis  redis  10d  -'
            | sudo tee -a /etc/tmpfiles.d/redis.conf
  fi
$ sudo usermod -aG redis vagrant
```

Após a instalação e configuração, foi necessário correr o servidor REDIS sem modo de proteção:

```
$ redis-server --protected-mode no
```

Para ficar tudo funcional, foram adicionadas as seguintes linhas ao ficheiro `/etc/gitlab/gitlab.rb`:

```
redis['enable'] = false
gitlab_rails['redis_host'] = 10.0.0.104
```

5 Desempenho crítico

Os pontos de desempenho crítico são: as bases de dados tais como o *Post* e o *Redis* uma vez que elas irão armazenar e processar dados, devido a isto é necessário garantir a sua disponibilidade e performance, outra componente crítica é o serviço *Gitlab Workhorse*, já que este lida com grandes pedidos pedidos HTTP como: *pull* e *request*. O *NGINX* é o balanceador de carga e ponto que lida diretamente com o cliente, assim em caso de falha, pode originar um grande *bottleneck* na aplicação, assim é necessário que a arquitectura tenha presente uma tolerância de falha desta componente.

6 Conclusão

Com a realização deste trabalho ficamos a perceber o modo de funcionamento da aplicação e as suas necessidades de modo a torná-la mais escalável. Percebemos também o quão importante é a necessidade de optimizar o processo de *deployment* para uma aplicação desta dimensão. Deste modo, esta análise permitir-nos-á organizar e optimizar a implementação da aplicação para a segunda fase do projeto.