Projeto de System Deployment & Benchmarking

Instalação Zulip

Relatório de Desenvolvimento

José André Martins Pereira
(a82880@alunos.uminho.pt)

Ricardo André Gomes Petronilho
(a81744@alunos.uminho.pt)

Miguel Raposo Dias
(pg41089@alunos.uminho.pt)

Ricardo Cunha Dias
(pg39295@alunos.uminho.pt)

31 de Outubro de 2019

Resumo

Na Unidade Curricular de Systems Deployment & Benchmarking foi-nos proposta a elaboração de um projeto de análise e deployment da aplicação Zulip, bem como a interpertação da arquitetura da mesma.

Conteúdo

Capítulo 1

Introdução

Na unidade curricular de **Systems Deployment & Benchmarking** foi-nos proposto um trabalho prático de análise, deployment e benchmarking de uma aplicação.

A aplicação escolhida pela equipa foi o **Zulip**, na medida em que segue os requisitos do trabalho prático, bem como, tem uma boa documentação. O **Zulip** é uma aplicação de chat em tempo real, onde o principal objetivo é oferecer um boa experiência a organizações, empresas e projetos voluntários, de pequenas equipas de amigos, até dezenas de milhares de utilizadores.

O objetivo principal deste trabalho prático é perceber a arquitetura da aplicação, deployment/instalação da mesma e respetivas configurações, as formas de comunicação entre os diferentes componentes e por fim a identificação de operações onde o desempenho seja crítico.

Capítulo 2

Arquitetura

2.1 Especificação

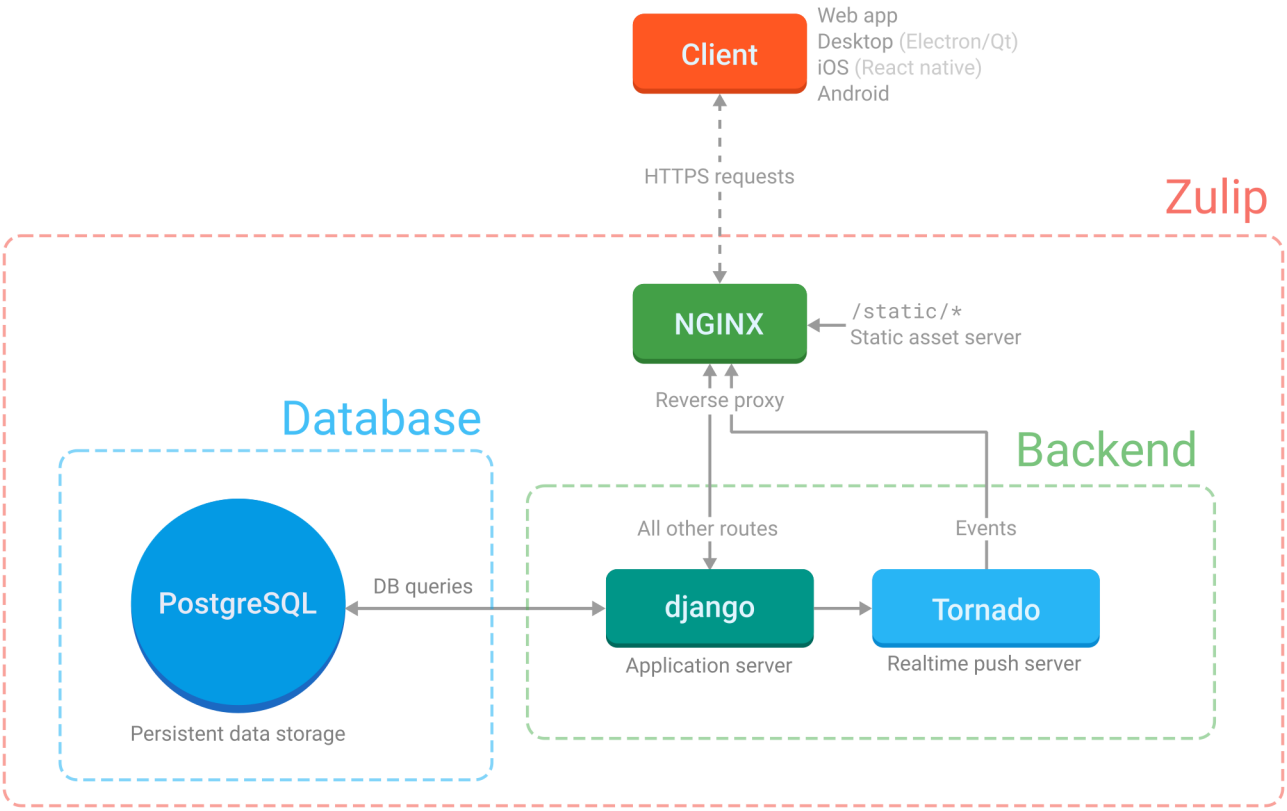

Figura 2.1: Arquitetura da aplicação Zulip.

A arquitetura da aplicação **zulip** contém vários **componentes**, sendo que, se consegue dividir os mesmos em diferentes subsistemas. No entanto, é perceptível pela figura 2.1, que existe uma separação entre o componente **Client** com os restantes, pois são subsistemas distintos.
O **Client** é responsável pela apresentação (*view*) da aplicação, enquanto que o restante subsistema é de uma forma geral, responsável pela lógica e tratamento de dados da mesma, sendo este mais complexo e composto

por subsistemas interiores (**Database** & **Backend**).
A divisão também deve-se ao facto do componente **Client** estar na máquina do utilizador, enquanto que os restantes estarão em máquinas administradas pela empresa de desenvolvimento.
O componente **NGINX** consiste num servidor *HTTP* intermediário (*middleware*) entre o componente **Client** com *HTTP requests* e o **Backend**. Na verdade, o **NGINX** permite uma independência entre os subsistemas, ou seja, o **Client**, pode ser definido para diferentes plataformas, nomeadamente: web, desktop, iOS e android.
Em relação à base de dados da aplicação, tem-se o componente **PostgreSQL**, um sistema de gerenciamento de dados persistente que apenas comunica com o servidor da aplicação **django**.
Em termos do sistema de **backend** da aplicação, este é composto por duas componentes, sendo elas **django** e **Tornado**, este primeiro **django** é de extrema importância tendo em conta que é a base da construção da aplicação **zulip**, e consiste num framework de web do lado do servidor de código aberto, desenvolvido em Python, fornecendo assim todo o framework necessário para ser mais tarde apresentado do lado do cliente, quanto ao **Tornado** é um servidor de web assíncrono e sem bloqueio de I/o que permite manter ativas milhares de ligações em tempo real, no caso do **zulip** é respónsavel pela entrega de mensagens e não muito mais.
Outros componentes secundários igualmente importantes no funcionamento do **zulip** são **Supervisor** que é um sistema cliente/servidor que permite monitorizar e controlar os processos em sistemas **unix**, no caso do **zulip** é utilizado para iniciar os processos do servidor, reinicializar-los em caso de falha e fazer *logging*, outro é **memcached** é um sistema distribuído de cache em memória, é usado para guardar em chache modelos de dados e invalidar-los caso sejam alterados.
Um outro componente secundário é a base de dados em memória **Redis** que armazena chaves com durabilidade opcional, é usado para guardar dados com tempo de vida bastante baixo.
O **RabbitMQ** é um software de mensagens com código aberto, que é usado para tratar de filas que requerem uma entrega fiável mas não são passíveis de fazer no sistema principal, é também usado para comunicações entre **django** e **Tornado**.
Por último, o componente **Thumbor** é um serviço de miniaturas de fotos open-source que tem como função administrar as fotos do sistemas ora via **upload** ora via **url**.

Capítulo 3

Padrões de distribuição e comunicação

A aplicação apresenta dois padrões bem definidos, nomeadamente o padrão **Cliente-Servidor** de onde advem a sua arquitera e o padrão ***Brooker*** usado na comunicação do componente **RabbitMQ** e serviços conectados a ele.

3.1 Padrão Cliente-Servidor

Este padrão consiste em duas partes. O servidor que fornece diferentes serviços e o cliente que os consome. A comunicação entre as duas partes, **Cliente** (aplicação móvel, web ou nativa) e **Servidor** (NGINX), é feita através do protocolo **HTTP**.

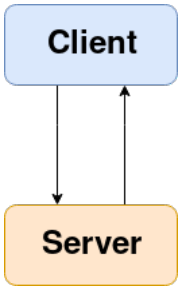

Figura 3.1: Padrão Cliente-Servidor

3.2 Padrão *Brooker*

Este padrão é utilizado na comunição de componentes independentes entre eles. A comunicação entre estes componentes é feita através de "mensagens"(**AMQP**) enviadas pelos componentes, as quais são depois recebidas pelo intermediário (**RabbitMQ**) que é responsável pela distribuição das mensagens pelo outros componentes.

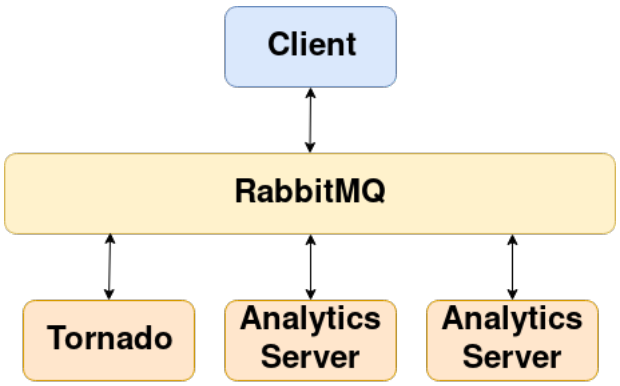

Figura 3.2: Padrão *Brooker*

Capítulo 4

Configuração

Os vários requisitos para correr o servidor Zulip estão detalhados na documentação oficial, vamos apenas especificar a configuração utilizada pelo nosso grupo.

4.1 Instalação

Tal como nas aulas práticas utilizou-se o **Virtual Box** criando um **Vagrantfile** com a configuração, desta forma a instalação é totalmente automática.
O sistema operativo é o **Ubuntu Xenial 16.04**, o hardware simulado tem **2 GB de RAM** e **2 CPUs** dedicados.
O endereço IP é estático: **10.0.0.101**. Para que outras máquinas externas ao host consigam ter acesso ao servidor instalado na VM recorreu-se à abertura das portas (**portforward**): **80** e **443**; estas são responsáveis pelo protocolo HTTP e HTTPS respetivamente. A seguinte figura ilustra a implementação deste passo:

```
vm1.vm.network :forwarded_port, guest: 80, host: 80
vm1.vm.network :forwarded_port, guest: 443, host: 443
```

Figura 4.1: Portfoward das portas 80 e 443.

Apesar da documentação oficial recomendar, **não se configurou um endereço DNS** uma vez que o ambiente é apenas de teste. O servidor zulip obriga o uso de um **certificado SSL**, pelo motivo referido anteriormente, foi gerado um **auto-certificado**. No entanto, por questões de segurança o Google Chrome e outros browsers por padrão não permitem o acesso a páginas com auto-certificados SSL ou desatualizados, desta forma foi necessário ativar essa funcionalidade acedendo (no Chrome) à página: chrome://flags/ tal como a figura mostra:

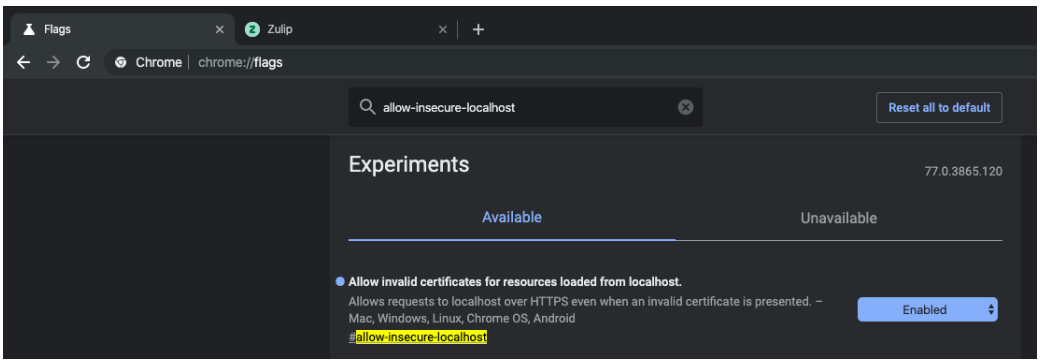

Figura 4.2: Ativação de páginas com auto-certificados SSL no Google Chrome.

Finalmente para instalar o servidor apenas é necessário correr os seguintes comandos, estando os mesmos no Vagrantfile para uma instalação automática.

```
wget https://www.zulip.org/dist/releases/zulip-server-latest.tar.gz
tar -xf zulip-server-latest.tar.gz
sudo ./zulip-server-*/scripts/setup/install --self-signed-cert --email=EMAIL --hostname=IP
rm -f zulip-server-latest.tar.gz
```

Figura 4.3: Label.

O **wget** descarrega os ficheiros necessários à instalação, o **tar** descompacta-os, **sudo ./...** corre o instalador, note-se que se usa **.*** na versão do instalador tornando o comando único/ genérico para qualquer versão e, finalmente, o **rm** elemina os ficheiros do instalador uma vez que já não são necessários.

No argumento –**email** deve ser referido o email de administrador que recebe mensagens de erros e suporte. O –**hostname** é o endereço utilizado para aceder ao servidor, no nosso caso é um endereço IP mas podia ser um endereço DNS. Tal como referido anteriormente o argumento –**sel-signed-cert** gera um auto-certificado SSL. Estes argumentos podem ser novamente configurados após a instalação.

4.2 Servidor email

4.2.1 Outgoing email

Após a instalação deve ser configurado um servidor **SMTP** - **S**imple **M**ail **T**ransfer **P**rotocol - uma vez que o servidor zulip necessita de enviar emails para os utilizadores para confirmação de contas, notificações, entre outros.
É possível instalar o servidor SMTP na mesma máquina (localhost) em que o servidor zulip se encontra, a equipa preferiu usar um servidor remoto já configurado. Como o ambiente de instalação é apenas de teste, recorreu-se a um servidor SMTP falso, isto é, um servidor que em vez de realmente enviar emails aos utilizadores, os mesmos ficam retidos sendo apenas uma simulação de envio, utilizando para isso o mailtrap.
No ficheiro **/etc/zulip/settings.py** é possível configurar o servidor SMTP, para tal devem ser definidas as seguintes variáveis:

- EMAIL_HOST
- EMAIL_HOST_USER
- EMAIL_PORT

Por último no ficheiro **/etc/zulip/zulip-secrets.conf** define-se a password do servidor na variável **email_password**.
Após a configuração é necessário reiniciar o zulip atavés do seguinte comando:
sudo su zulip -c '/home/zulip/deployments/current/scripts/restart-server'
Desta forma todos os serviços e componentes associados ao servidor zulip são reiniciados.

4.2.2 incoming email

O incoming email, permite enviar mensagens para o **zulip** via email, garantindo assim integrar aplicações de terceiros para enviar mensagens ("notificações") para o **zulip** e os utilizadores poderem responder a emails de notificações.
Existem duas formas de configurar o gateway de email:

- **Localmente**: um servidor **Postfix** corre no servidor e direciona as mensagens diretamente para a aplicação.
- **Polling**: uma tarefa agendada(**cron job**) que corre na aplicação que verifica minuto a minuto a caixa de entrada do email de instituição por mails novos.

Localmente

Deve-se criar um **Mail exchanger record** configurando o email para o email da aplicação para ser processado pelo **host** escolhido. Para verificar o resultado:

```
$ dig +short "Nome de domínio completo" -t MX
```

De seguida no ficheiro de configuração (/etc/zulip/zulip.conf) deve se adicionar zulip::postfix_localmail a puppet_classes o resultado deve ser algo assim:

```
puppet_classes = zulip::voyager, zulip::postfix_localmail
```

Caso o nome de domino seja diferente do nome do email escolhido, deve -se adicionar uma secção ao ficheiro de configuração (/etc/zulip/zulip.conf):

```
[postfix]
mailname = emaildomain.example.com
```

Por fim deve-se alterar o **email_gateway_pattern** no ficheiro (/etc/zulip/settings.py) para ("%s@(emaildodominio).com")
Para aplicar as novas alterações deve-se correr o ficheiro (/home/zulip/deployments/current/scripts/zulip-puppet-apply) e responder com y e de seguida reinicar a aplicação com seguinte comando (/home/zulip/deployments/current/scripts/restart-server) de modo poder aplicar as novas alterações ao sistema.

Polling

Deve ser selecionado um endereço de e-mail que será dedicado para as mensagens de gateway, após isto deve-se alterar o **email_gateway_pattern** no ficheiro (/etc/zulip/settings.py) para ("%s@(emaildodominio).com"). De seguida deve-se permitir o protocolo de acesso à mensagem da internet **(IMAP)**, ter acesso ao e-mail, de forma a permitir o acesso deste protocolo à aplicação **zulip** é necessário especificar os detalhes de email no ficheiro de configuração (/etc/zulip/settings.py) e especificar a **password** no ficheiro (/etc/zulip/zulip-secrets.conf) no parâmetro **email_gateway_password**. Por fim é necessário instalar, podendo ser alcançado através dos seguintes comandos:

```
cd /home/zulip/deployments/current/
sudo cp puppet/zulip/files/cron.d/email-mirror /etc/cron.d/
```

4.3 Automatização

Com o propósito da instalção do servidor zulip ser o mais automático possível foram definidas no ínicio do ficheiro Vagrantfile um conjunto de variáveis utilizadas durante a instalação e configuração do servidor. Desta forma o utilizador não tem de, após a instalação, abrir e alterar ficheiros específicos à configuração tornando o processo mais facilitado.

```
# HARDWARE
RAM = "2048"
CPUs = "2"

# NETWORK
IP = "10.0.0.101"
PORTFOWARD_80 = "80"
PORTFOWARD_443 = "443"

# ZULIP

# Installer options
ZULIP_ADMINISTRATOR = "[redacted]@gmail.com"
EXTERNAL_HOST = "10.0.0.101"
SSL_CERT = "--self-signed-cert"

# SMTP
EMAIL_HOST = 'smtp.mailtrap.io'
EMAIL_HOST_USER = '[redacted]'
EMAIL_HOST_PASSWORD = '[redacted]'
EMAIL_PORT = '2525'
```

Figura 4.4: Variáveis de instalação e configuração.

Capítulo 5

Pontos críticos

Analisando a arquitetura da aplicação **zulip**, verifica-se que existem componentes não replicados, sendo que, do ponto de vista do grupo, isso originará pontos críticos.

Deve ser aplicado a grande parte da arquitetura uma replicação, pois não se deve conter num sistema componentes, que em sua falta, causem a falha de todo o sistema. Deste modo, o componente do **servidor da base de dados** e o **aplicacional**, devem ser replicados para tornar a aplicação **zulip** tolerante a faltas, bem como reduzir os congestionamentos destes servidores. Operações que façam acessos à base dados, ou apenas ao servidor aplicacional, estão dependetes destes componentes, sendo crucial a fiablidade dos mesmos.

Deste modo, sugere-se a extensão da base de dados a pelo menos dois servidores **PostgreSQL**, bem como o servidor aplicacional, denominado **Django**.

Do mesmo modo, o componente **NGINX** também deve ser replicado, pois o mesmo é responável pela comunicação entre o **Client** e o restante sistema, logo a sua replicação tornará o sistema tolerante à sua falta, bem como distribuirá os pedidos pelas replicas, reduzindo a sobrecarga num único componente deste tipo.

Capítulo 6

Conclusão

Assim, finalizada a primeira fase do projeto, conclui-se que todos os objetivos inicialmente propostos foram cumpridos, tais como, análise da arquitetura, os padrões de distribuição e comunicação, a configuração da aplicação e por fim os pontos críticos da mesma.

Em relação à arquitetura da aplicação **zulip** a equipa baseou-se na documentação do website oficial da aplicação, onde a mesma é explicada detalhadamente.

FALAR SOBRE OS PADRÕES DE COMUNICAÇÃO

Para uma melhor perceção da configuração da aplicação a equipa decidiu instalar a mesma usando o **Vagrant**, e apresentar todos os passos neste relatório, tal como se pode verificar, com o objetivo de clarificar todos os processos da instalação.

Por fim, e não menos importante a análise dos pontos críticos da aplicação passou por uma análise da arquitetura da mesma e identifcar possíveis problemas. Os principais problemas encontrados foram a não tolerância a faltas do sistema e possíveis congestionamentos de servidores, quer da base de dados, bem como do servidor aplicacional.