Escola de Engenharia
Universidade do Minho

1º Semestre 2020/21
(Mestrado em Engenharia Informática)

System Deployment and Benchmarking

Trabalho Prático - Fase 1

Número	**Nome Completo**
PG42816	Cândido Filipe Lima do Vale
A89982	Joel Alexandre Dias Ferreira
A85700	Pedro Miguel Araújo Costa
A80789	Rui Filipe Brito Azevedo

20 de Novembro de 2020

Conteúdo

1 Introdução

O presente relatório tem como objectivo caracterizar e analisar a aplicação *open-source Mattermost*, uma plataforma de *chat* que agiliza toda a comunicação numa organização.

Numa primeira instância será feita a caracterização do sistema referido, definindo a sua arquitectura e componentes, as suas formas de comunicação, bem como os seus pontos de configuração. Posteriormente, será feita uma análise às operações com desempenho crítico e possíveis pontos de falha.

O objectivo desta fase de caracterização e análise é, posteriormente, tornar o sistema mais escalável e disponível para os utilizadores, assim como automatizar a instalação do mesmo.

2 Mattermost

O *Mattermost* é um serviço *open-source*, *self-hostable* de comunicação, partilha e procura de ficheiros, contendo outras integrações. É maioritariamente usado em organizações para comunicações internas servindo como alternativa a outras aplicações do mesmo género como o *Slack* e o *Microsoft Teams*.

3 Arquitectura e Componentes

O *Mattermost* é composto, essencialmente, por três camadas lógicas, nomeadamente, uma *interface* para o utilizador final, um servidor de aplicação e uma camada de persistência de dados de todos os utilizadores. Esta última é dividida em dois componentes, um sistema de base de dados usado para guardar e devolver dados do sistema assim como fazer procuras *full-text*, e um sistema de ficheiros onde serão armazenados ficheiros e imagens partilhadas pelos utilizadores.

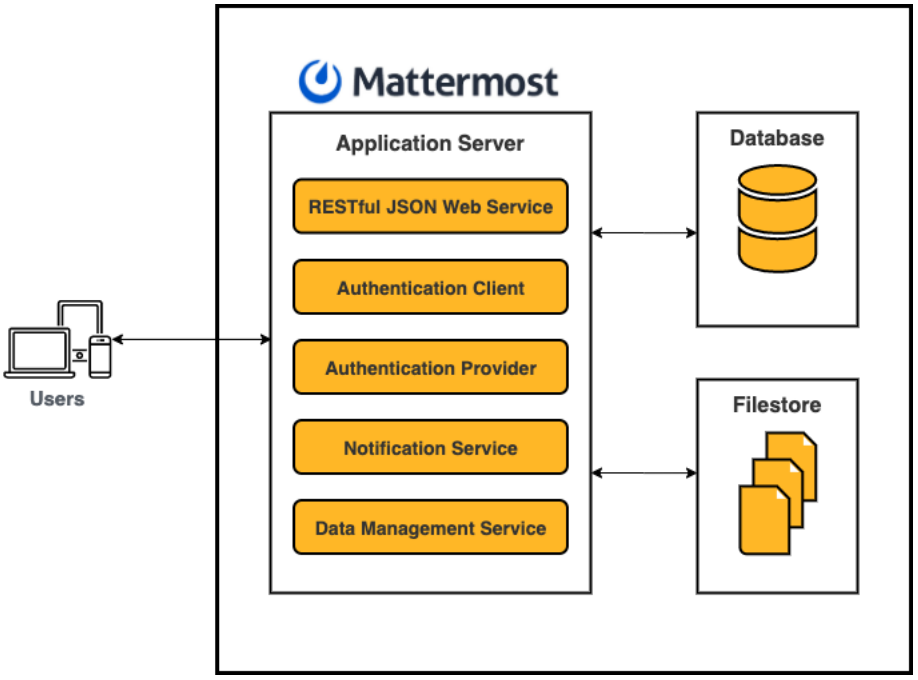

Figura 1: Arquitectura *single-machine*

O servidor de aplicação (*middleware*) é responsável por receber os pedidos dos utilizadores e tratá-los, garantindo o acesso à base de dados e ao sistema de ficheiros. Neste sistema, o servidor de aplicação está dividido nos seguintes componentes:

- ***RESTful API Web Service***: usado pelos clientes da aplicação, bem como aplicações de terceiros, para interagir com o servidor.
- ***Authentication Client***: autentica utilizadores através de *email* e *password*.

- ***Authentication Provider***: permite que o servidor *Mattermost* se autentique com outros serviços através da interface da autenticação do cliente.
- ***Notification Service***: envia notificações via *SMTP* (*Simple Email Transfer Protocol*).
- ***Data Management Service***: faz a conexão com o sistema de base de dados assim como o sistema de ficheiros, e faz a gestão dos dados que são escritos e lidos dos mesmos.

O *deployment* desta aplicação tem como objectivos disponibilizar a mesma para um número elevado de clientes, fazendo com que o sistema seja altamente disponível, escalável e, de preferência, com nenhum ponto único de falha. Dado isto, serão usados, para além dos já mencionados, três componentes que permitem cobrir os objectivos da instalação, particularmente:

- ***NGINX***: servidor de *reverse proxy* usado como *interface* de acesso ao serviço e responsável por reencaminhar o tráfego para outros servidores. Para além disso, actua também como balanceador de carga sobre os servidores aplicacionais do sistema.
- ***Prometheus & Grafana***: componentes que irão fazer parte de uma camada de monitorização do sistema. O *Prometheus* é usado para para rastrear a saúde do *deployment* através de análises de *performance*. Por sua vez, o *Grafana* permite visualizar as métricas colectadas pelo *Prometheus*.
- ***Elasticsearh*** : motor de busca distribuído que permite fazer buscas de *full-text*. Permite também, tornar as procuras na base de dados mais eficientes num sistema *cluster-based*.

Nas secções posteriores, iremos abordar mais detalhadamente estes três últimos componentes.

4 Formas de Comunicação

De modo a assegurar a comunicação entre o utilizador, o servidor Mattermost e os restantes componentes da sua arquitectura, são utilizados três tipos de conexões:

- **WSS - WebSocket Secure:** A comunicação entre o cliente e o servidor Mattermost pode ser feita através de WSS ou HTTPS no entanto, é feita preferencialmente usando WebSecure Sockets pois estes permitem actualizações e notificações em tempo real enquanto que, numa conexão HTTPS é necessário actualizar a página de forma constante para obter as actualizações mais recentes. Isto deve-se ao facto de a conexão WSS ser permanente e bidireccional, o que significa que o servidor pode enviar mensagens directamente para o cliente

enquanto que uma conexão HTTPS é intermitente, ou seja, o servidor só envia informação ao cliente quando este faz um pedido.

- **HTTPS - Hyper Text Transfer Protocol Secure:** Quando não for possível estabelecer uma conexão WSS, o sistema tentará estabelecer conexões HTTPS. Nestas ligações, é possível ao cliente fazer a renderização das páginas e aceder às principais funcionalidades da aplicação mas não será possível obter actualizações ou notificações em tempo real. Caso não seja possível estabelecer conexões WSS ou HTTPS o serviço Mattermost não irá funcionar.
- **TCP/IP**: A comunicação entre o Servidor Mattermost e as Bases de Dados existentes é efectuada através de conexões TCP/IP nas quais existe a possibilidade de se utilizar TLS para estabelecer ligações encriptadas. De acordo com a documentação, num modelo Single Machine é também possível que ligações com bases de dados PostgreSQL sejam feitas através de Unix Sockets.

O protocolo de comunicação utilizado entre o servidor Mattermost e a componente de *file storage* será HTTP(S) para implementações em que se utilize *MinIO* ou *S3* mas pode variar caso se utilize NAS (Network-Attached Storage).

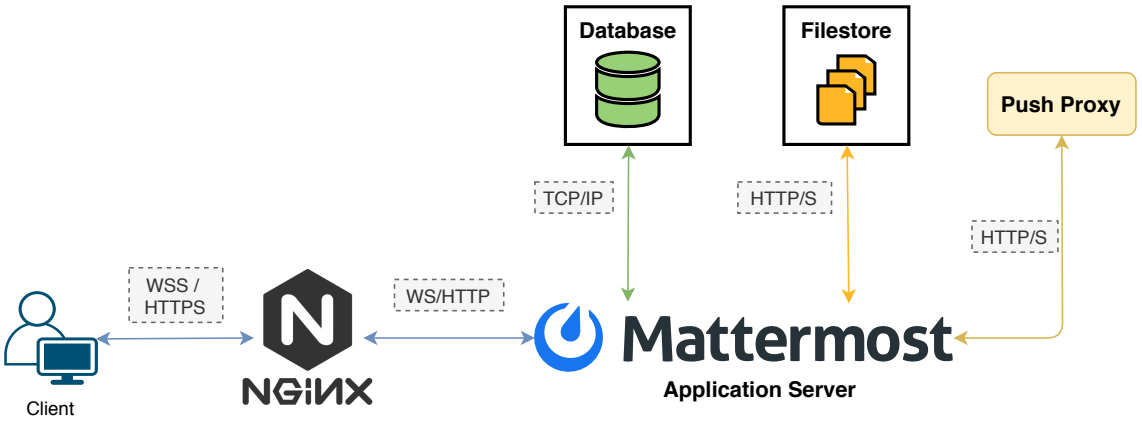

Figura 2: Comunicação entre Componentes - Mattermost

5 Viabilidade e Escalabilidade

É imperativo que um sistema distribuído seja capaz de suportar um grande número de utilizadores e garantir que todos os pedidos feitos pelos mesmos tenham resposta em tempo útil. Para além disso, é de elevada importância que, mesmo que um ou vários componentes do sistema falhe, isto seja completamente transparente aos utilizadores.

Dado isto, é perceptível que o modelo *single-machine* não é o ideal para uma solução escalável e viável pois apresenta vários aspectos que tornam o sistema instável.

5.1 Pontos únicos de falha

Os pontos únicos de falha (SPOFs) são partes do sistema que, caso falhem, todo o sistema irá parar de funcionar. Tendo como base a arquitectura apresentada inicialmente, os pontos únicos de falha são os seguintes:

- **Servidor Aplicacional**: a falha do servidor aplicacional compromete o funcionamento do sistema uma vez que os utilizadores não conseguiriam aceder aos serviços do mesmo. A solução para este problema passa por ter várias instâncias de servidores aplicacionais conectados a um servidor *proxy* capaz de fazer o balanceamento de carga e tornar transparente a falha de um ou vários servidores de aplicação.
- **Servidor *Proxy***: esta camada actua como um balanceador de carga sendo um ponto onde passa todo o tráfego dirigido ao servidor aplicacional. É portanto, de extrema importância que este esteja sempre funcional caso contrário toda a aplicação irá deixar de funcionar. Para solucionar este problema deve ser implementada redundância na camada de Proxy, de forma a que, mesmo quando exposta a um elevado número de pedidos este não seja um ponto de falha e a disponibilidade do serviço possa ser garantida.
- **Base de Dados**: da mesma maneira, a base de dados também é um ponto único de falha. Uma vez que todos os dados do sistema são guardados nesta camada, esta irá ser acedida em praticamente todas as operações efectuadas e portanto, estar sujeita a um elevado número de pedidos. Isto significa que uma falha da mesma irá implicar uma pior performance ou, no pior cenário, perda de acesso à informação impedindo a comunicação entre utilizadores. Mais uma vez, a solução passa por replicar os dados persistidos por vários servidores.

5.2 Operações com desempenho crítico

Uma operação com desempenho crítico trata-se de qualquer operação que necessite de ser otimizada ao máximo para funcionar com *delay* mínimo.

Consideram-se operações com desempenho crítico as seguintes:

- ***Push Proxy Layer***: Tendo em conta que o objectivo do serviço *Mattermost* é, essencialmente, agilizar e facilitar a comunicação entre uma equipa, reconhecemos que uma das operações mais críticas é o envio de notificações para os vários utilizadores. Esta é uma feature crucial para o uso eficiente do sistema que depende maioritariamente de investimento em hardware.
- **Pesquisa**: Embora a pesquisa pareça um problema trivial, numa quantidade enorme de dados, pode ser um grande problema. Quando temos uma aplicação como o Mattermost onde são enviadas milhares de mensagens e ficheiros em canais distintos diariamente, é crucial que o sistema de pesquisa seja rápido o

suficiente para encontrar uma mensagem rapidamente. Esta é também, para nós, uma operação com desempenho crítico, uma vez que nenhum utilizador gosta de ficar muito tempo à espera de uma resposta à sua pesquisa. Uma forma de solucionar este problema é recorrer à divisão da carga de trabalho pelos diferentes servidores de pesquisa, garantindo assim uma resposta mais rápida ao utilizador.

- ***File Storage***: esta componente não foi considerada um *SPOF* pois, mesmo que por algum motivo não seja possível aceder ao sistema de ficheiros, a aplicação contínua activa. No entanto, não é desejável que esta componente pare, daí acrescentar redundância ao sistema de ficheiros ser algo a ter em consideração. Com isto, para além de reduzir os *downtimes* provocados pela falha deste serviço, aumentaria-se a disponibilidade do mesmo tornando mais eficiente o acesso aos ficheiros e imagens.

6 Padrões de Distribuição

No Mattermost existem essencialmente 3 tipos de padrões de distribuição. São eles, os padrões Single-Machine, Multi-Machine e Cluster-Based. Em cada um destes padrões o número de recursos varia, uma vez que a aplicação à medida que vai crescendo tem de manter alguns standards de performance.

A tabela seguinte, relaciona o tipo de arquitectura a adoptar consoante o número de utilizadores do serviço.

Users	**Deployment**
<2 000	Single-Machine
2 000 - 10 000	Multi-Machine
>10 000	Cluster-based

Tabela 1: Tipos de instalação

6.1 Single-Machine

No caso do padrão de distribuição Single-Machine, o número de utilizadores terá de ser inferior a 2000 utilizadores. Neste caso não há a necessidade de se ter uma arquitetura de servidor de proxy, bastando apenas ter um servidor aplicacional Mattermost que recebe os pedidos dos clientes e os reencaminha para as outras camadas.

Relativamente ao serviço *job* e de monitorização (serviço que se encontra em comunicação constante com o servidor aplicacional para obter reports do estado do sistema) optou-se por não replicar o mesmo, uma vez que não são operações com desempenho crítico.

A camada *elasticsearch*, cujo objetivo é fornecer um mecanismo de pesquisa de texto completo e distribuído, bastante útil para a pesquisa de mensagens, *file storage* e *push proxy* têm apenas também um único servidor, uma vez que, dado o número reduzido de utilizadores existe pouca probabilidade da indisponibilidade do serviço.

A camada de base de dados, tem uma base de dados primária e outra secundária onde os dados da base de dados primária estão a ser constantemente replicados na secundária e em caso de falha existe a possibilidade da secundária assumir o cargo de primária.

Na figura seguinte consegue visualizar-se um exemplo do padrão de distribuição *Single Machine*.

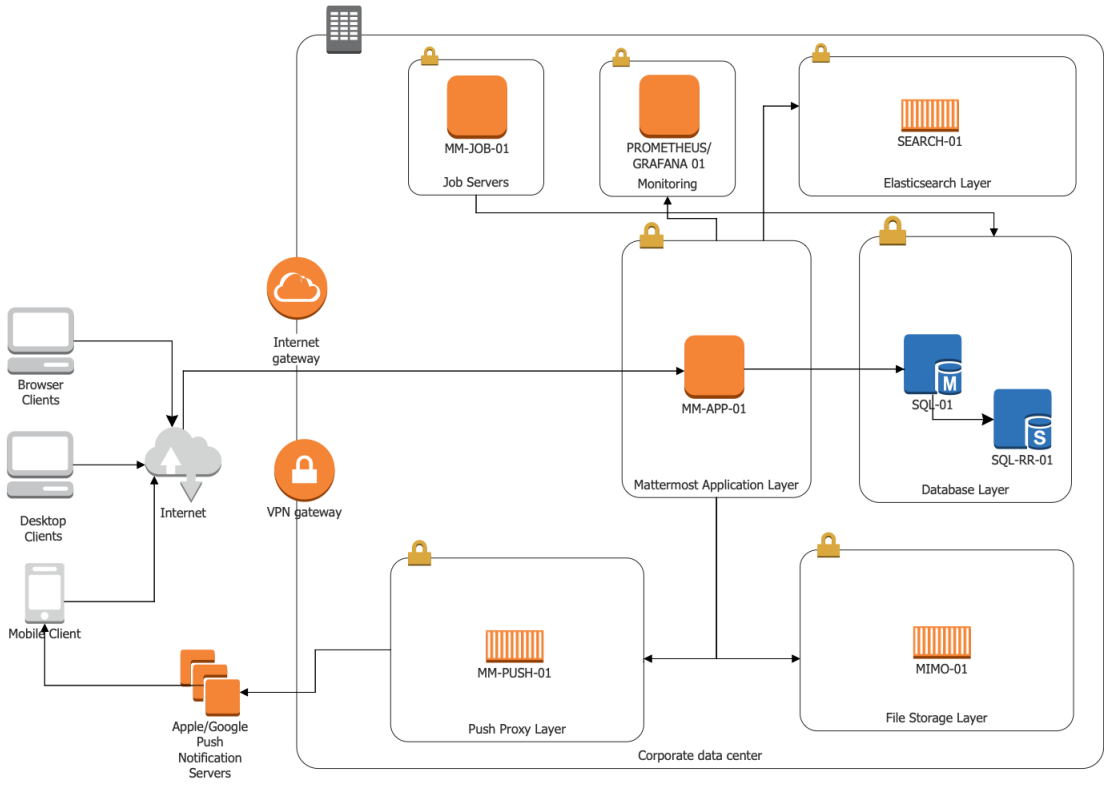

Figura 3: Padrão de distribuição Single Machine.

6.2 Multi-Machine

Para o padrão Multi-Machine, estima-se um número de utilizadores entre 2000 e 10000 e consequentemente teremos de ter em conta este fator no nosso padrão de distribuição. Vão ser necessários recursos adicionais para suportar a carga de pedidos feitos pelos utilizadores à aplicação.

No caso anterior vimos que não existia a necessidade de ter uma camada de proxy, uma vez que o número de utilizadores era reduzido, no entanto neste

padrão é necessário ter-se pelo menos dois servidores de proxy (NGINX) que fazem o balanceamento de carga pelos nossos servidores aplicacionais, temos então uma arquitetura servidor de proxy.

Relativamente à camada de *Job* e de monitorização não há a necessidade de replicar estes serviços, uma vez que não são serviços com desempenho crítico. No entanto, o mesmo não acontece com a camada de base de dados, camada de armazenamento de ficheiros, *push proxy* e *elasticsearch*, havendo a necessidade de replicar alguns serviços dentro destas camadas para garantir alta disponibilidade e escalabilidade da aplicação.

Na camada de base de dados optou-se por ter uma réplica adicional da base de dados master, garantindo assim que em caso de falha da master temos uma das duas bases de dados secundárias pronta a assumir o papel de master.

Nas camadas de *Elasticsearch*, *Push Proxy* e de armazenamento de ficheiros, optou-se por incrementar em cada uma destas o número de servidores em uma unidade. Para além disto, foi adicionado a cada uma destas camadas um balanceador de carga que distribui de uma maneira justa os pedidos dos clientes pelos seus servidores. Na imagem seguinte está representado o respectivo padrão de distribuição.

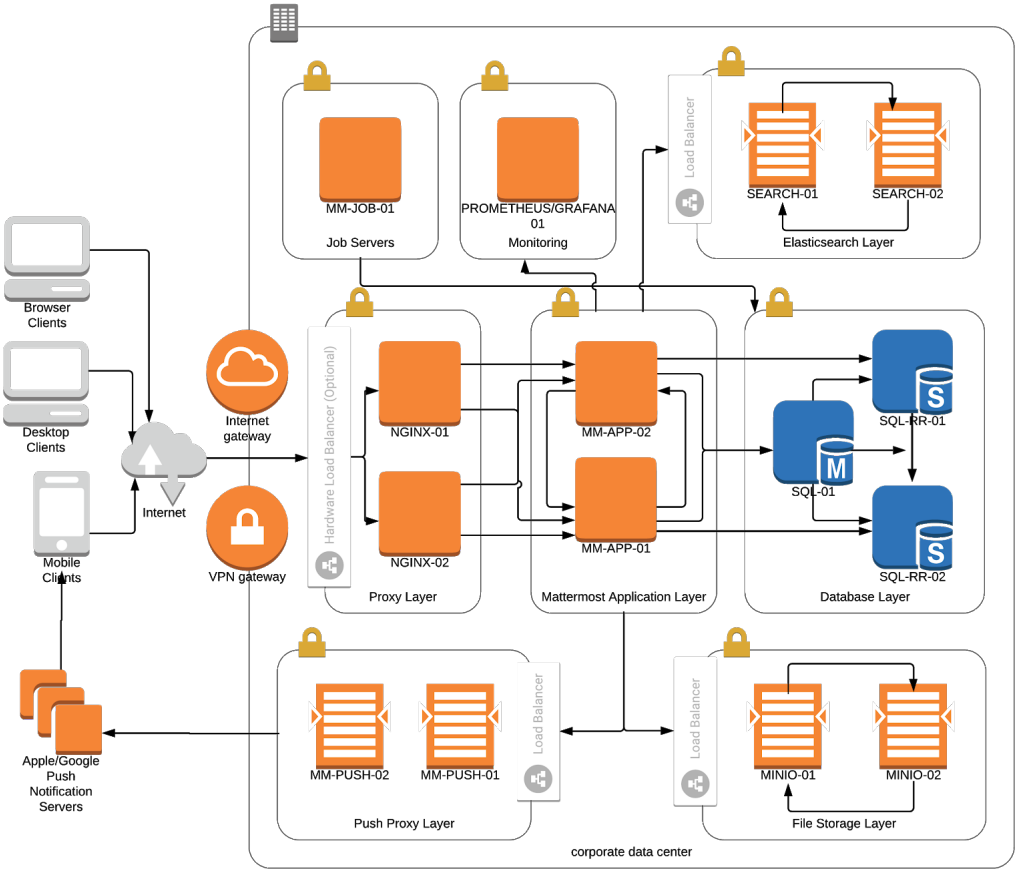

Figura 4: Padrão de distribuição Multi Machine.

6.3 Cluster-Based

Quando o número de utilizadores a utilizar a aplicação aumenta drasticamente (mais do que 10000 utilizadores), há a necessidade de adoptar estratégias de gestão e de garantias de disponibilidade de todos os serviços presentes na aplicação. Uma dessas soluções é a utilização de um cluster. Através da utilização de um cluster, conseguimos garantir que em caso de falhas, a nossa aplicação continua a funcionar correctamente, através do recurso a infraestruturas redundantes.

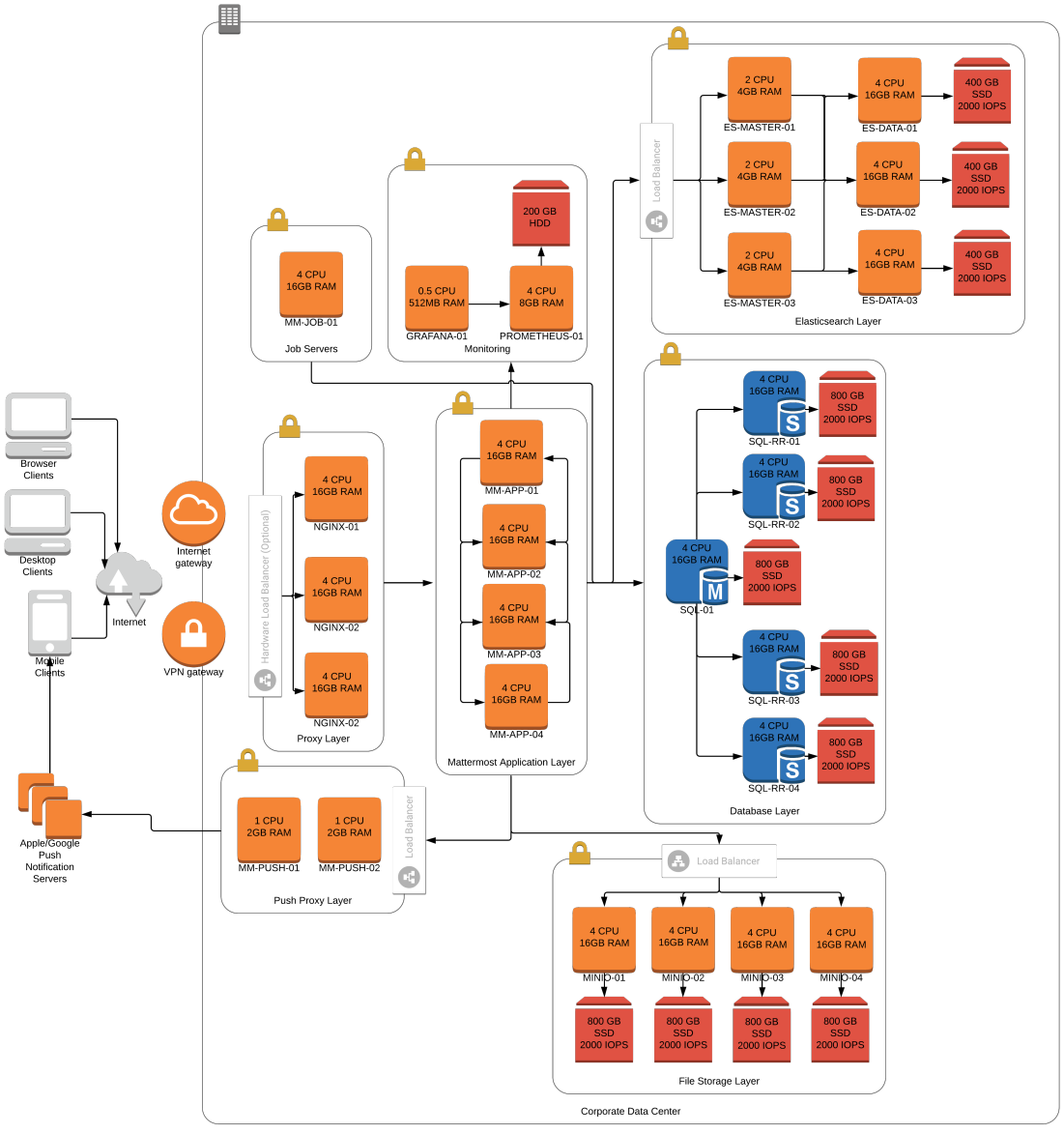

Figura 5: Padrão de distribuição Cluster Based.

A alta disponibilidade do Mattermost num número tão elevado de utilizadores é conseguida através da utilização de servidores aplicacionais redundantes, bases de dados redundantes e balanceadores de carga redundantes. A falha de qualquer um destes componentes não interfere com o bom funcionamento de todo o sistema. Após a falha de um dos componentes os outros componentes redundantes devem assumir a responsabilidade do bom funcionamento do sistema. Num cluster os nós gestores comunicam com os nós trabalhadores e distribuem a carga de trabalho entre todos eles.

Uma vez que o número de clientes duplicou face ao exemplo anterior (Multi-Machine), foi também necessário incrementar o número de réplicas dos serviços críticos para suportar a carga elevada de pedidos. No diagrama seguinte são visíveis essas alterações.

6.4 Padrão de Distribuição com Docker e Kubernetes

Numa aplicação com várias camadas, cada uma delas com vários serviços, torna-se complicado garantir a organização de tudo isto. Posto isto, nos diferentes serviços que dispomos, existe a necessidade de os isolar para garantir uma gestão eficiente dos mesmos, uma solução para este problema é o recurso à utilização de *containers*. Para o *deployment* da aplicação consideramos o *Docker*.

Tal como vimos nas sub-secções anteriores, temos casos em que para a aplicação se tornar escalável e confiável existe a necessidade de se replicarem alguns serviços. No entanto com a replicação, existe a necessidade da comunicação constante entres as várias réplicas para fazerem a distribuição da carga de trabalho. Como solução para este problema recorremos ao *Kubernetes*, um *cluster* que agrega vários nós *master* que distribuem a carga de trabalho pelos diversos *workers*. Além disso a utilização de um *cluster*, como o *Kubernetes*, permite-nos aumentar e diminuir o número de réplicas bastante facilmente.

Na figura seguinte, conseguimos visualizar um diagrama, com os diversos serviços onde serão utilizados *containers* e onde iremos utilizar um *cluster* para a gestão dos diversos serviços. Como podemos visualizar a camada de Proxy está fora do *cluster*, isto deve-se ao facto do serviço NGINX estar a funcionar como controlador de pedidos (*ingresser controller*) para o *cluster*.

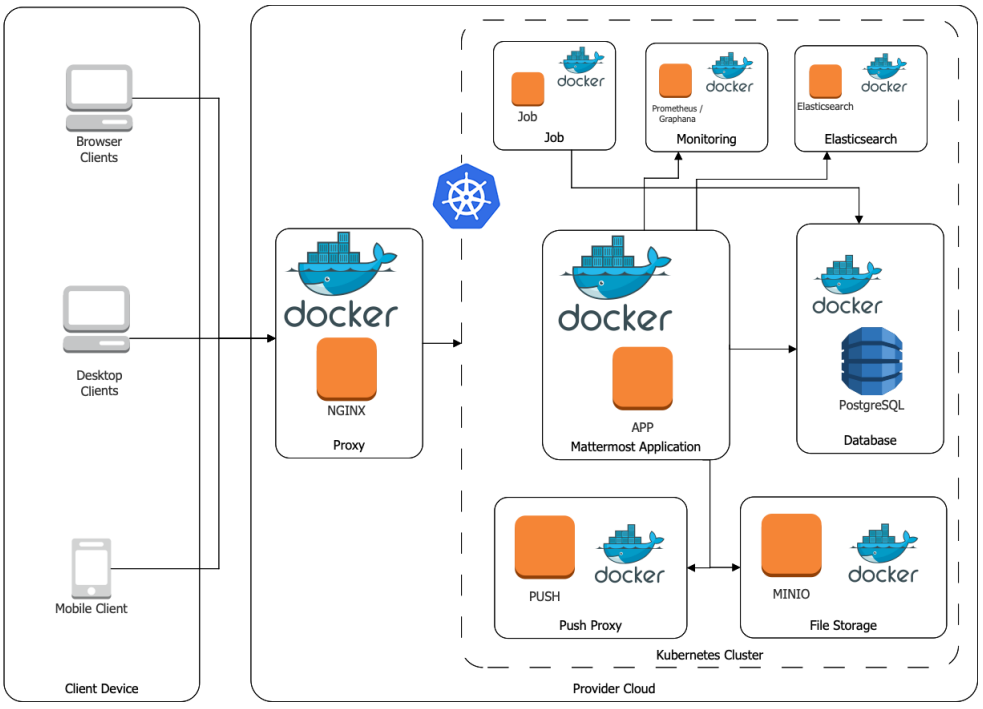

Figura 6: Padrão de distribuição com Docker e Kubernetes.

7 Pontos de Configuração

Após feita a instalação do Mattermost com sucesso, temos a possibilidade de o configurar de acordo com as nossas necessidades, manipulando praticamente todos os serviços fornecidos. No entanto, vamos também abordar alguns aspetos de configuração feitos durante a instalação.

Para esse fim, temos ao nosso dispor o ficheiro localizado em *mattermost/-config/config.json*, ainda que, a partir da versão 5.10 seja possível manter esta configuração na base de dados, permitindo assim a utilização direta da System Console para alterar qualquer feature.

7.1 Configurações gerais

Neste ficheiro temos centenas de opções para explorar, dizendo estas respeito a todos os pontos da aplicação. Podemos, por exemplo, customizar o processo de criação de contas forçando a verificação do email utilizado ou até limitar os domínios de email aceitados. A esmagadora maioria destas configurações são apenas flags ou listas de algum tipo de valores que estejamos a especificar algo sobre.

7.2 Base de Dados

O Mattermost disponibiliza a possibilidade de optar pelo serviço de base de dados que queremos utilizar, limitando as opções a MySQL ou a PostgreSQL. Nós

optamos por utilizar MySQL pelo que vamos abordar a configuração feita para esse serviço.

Foi apenas necessário criar a base de dados no MySQL console e de seguida criar um utilizador, com o IP da máquina que contém a base dados, com todas as permissões. De resto tivemos apenas de comentar a linha *bind-adress = 127.0.0.1* no ficheiro de configuração do MySQL (/etc/mysql/mysql.conf.d/mysqld.cnf).

7.3 Sistema de gestão de ficheiros

Como foi indicado anteriormente na secção 3, tencionamos utilizar MinIO para este fim, sendo este um componente S3. O Mattermost dá-nos a opção de escolher entre um sistema de ficheiros local ou um sistema S3 compatível. O standard é o Amazon S3 mas também há compatibilidade com MinIO e Digital Ocean Spaces.

Basta alterar o campo **DriverName** no config consoante a decisão.

7.4 Sevidor proxy NGINX com SSL e HTTP/2

Ainda que a decisão não seja final, iremos muito provavelmente utilizar o serviço NGINX como proxy.

Contrariamente a TLS, utilizar NGINX permite-nos melhorar algumas configurações, sendo estas:

- Desativar forwarding do tráfego inseguro no port 80 para o port 443.
- Utilizar *client_max_body_size* como tamanho máximo de ficheiros anexados.

Toda a configuração necessária após instalar o proxy é feita no ficheiro /etc/nginx/sites-available/mattermost. Basta alterar a variável **server_name** para o endereço correto ao longo do ficheiro de configuração.

8 Conclusão

O desenvolvimento deste trabalho prático permitiu-nos adquirir bastante informação relativamente ao *deployment* de uma aplicação num mundo real, onde existe a necessidade de garantir extrema fiabilidade dos serviços e garantir que uma aplicação é escalável.

Conseguimos essencialmente perceber o processo incremental para o *deployment* de uma aplicação, desde a avaliação da arquitetura geral até à parte dos padrões de distribuição onde existe a necessidade de garantir a facilidade de gestão e manutenção de toda a aplicação, quer seja com o recurso a *clusters* ou com o recurso ao isolamento dos serviços através de *containers*.

Ao fazermos uma análise como um todo da aplicação percebemos que um dos pontos extremamente importantes antes de pôr em prática todo o processo de *deployment* é identificar quais serão os pontos únicos de falha da nossa aplicação e quais serão as operações com desempenho crítico, após reflexão sobre estes assuntos conseguimos ter uma visão ampla sobre como devemos distribuir a nossa aplicação e como a devemos escalar.

Foi sem dúvida uma extrema mais valia poder fazer esta análise prévia da aplicação e perceber como poderíamos solucionar alguns dos problemas encontrados, mesmo antes de iniciar o processo de *deployment*, fazendo com que o nosso *mindset* esteja em tornar a aplicação escalável, confiável, disponível e fácil de gerir.

Por fim e após a vasta pesquisa efetuada, ficamos com uma percepção completamente diferente relativamente às vastas configurações necessárias para o *deployment* da aplicação como um todo, reconhecendo assim a necessidade de arranjar alguma ferramenta que nos auxilie neste processo, como é o caso do Ansible.

Referências

[1] Documentação Mattermost. Disponível em: https://docs.mattermost.com.

[2] NGINX Ingresser Controller. Disponível em:
https://www.nginx.com/products/nginx-ingress-controller/

[3] Slides acerca de Containers e Clusters da UC de System Deployment & Benchmarking.

[4] Amazon S3 REST API. Disponível em:
https://docs.aws.amazon.com/pt_br/AmazonS3/latest/API/Welcome.html

[5] Mais informação sobre como instalar o serviço mattermost.
https://wiki.archlinux.org/index.php/Mattermost