
Gestão de Infraestruturas de Computação

# Easy Appointments

Relatório

*Autor:*
84746 Rafael Teixeira

*Professor:*
João Barraca

# Conteúdo


**1 Introdução** **4**

**2 Objetivos** **5**

**3 Produto** **5**
3.1 Cenário Alvo . . . . . 6
3.2 Composição do Produto . . . . . 6

**4 Estratégia** **7**
4.1 SLA Estabelecido . . . . . 7
4.1.1 Serviço . . . . . 7
4.1.2 Componentes Individuais . . . . . 7
4.2 Distribuição de Carga e Redundância . . . . . 7
4.2.1 Nginx . . . . . 8
4.2.2 MySQL InnoDB . . . . . 9
4.2.3 Docker Swarm . . . . . 9
4.3 Monitorização, Alarmística e Recuperação de Falhas . . . . . 10
4.4 Docker Swarm . . . . . 10
4.4.1 TICK Stack . . . . . 10
4.5 ELK Stack . . . . . 11

**5 Instalação** **11**
5.1 Docker Swarm . . . . . 12
5.2 NFS . . . . . 13
5.3 Nginx . . . . . 14
5.4 Aplicação . . . . . 14
5.5 InnoDB . . . . . 14
5.6 Imagens Docker . . . . . 15

**6 Operação** **15**
6.1 Mecanismos de Monitorização e Alarmística . . . . . 15
6.1.1 Telegraf . . . . . 15
6.1.2 SNMP . . . . . 16
6.1.3 InfluxDB, Chronograf e Kapacitor . . . . . 17
6.1.4 Syslog . . . . . 17
6.2 Logstach, Elasticsearch e Kibana . . . . . 17
6.3 Mecanismos de Automação . . . . . 18
6.3.1 Nginx e MySql Router . . . . . 18
6.3.2 MySql Server e Aplicação . . . . . 18

**7 Testes** **19**
7.1 Considerações Iniciais . . . . . 19
7.2 Preparação do Ambiente . . . . . 19
7.3 Workflow do Teste . . . . . 19
7.4 Primeira Fase de Testes . . . . . 20
7.5 Segunda Fase de Testes . . . . . 21

**8 Conclusão** **22**

**9 Referências** **23**

**10 Anexo de Imagens das Dashboards e Alarmes** **24**

# Lista de Figuras


1 Arquitetura da Solução . . . . . 12
2 Chamadas realizadas para simular carga . . . . . 20
3 Resultados do teste com 100 utilizadores . . . . . 21
4 Resultados do teste com 50 utilizadores . . . . . 22
5 Home page Chronograf . . . . . 24
6 Dashboard dos Dados do Sistema . . . . . 24
7 Dashboard dos Dados do Apache . . . . . 25
8 Dashboard dos Dados do MySql . . . . . 25
9 Dashboard dos Dados do Nginx . . . . . 26
10 Configuração do alerta do Nginx . . . . . 26
11 Chamada http para Escalagem Automática . . . . . 26
12 Mensagens de Alerta por Violação do SLA . . . . . 27
13 Mensagens de Downgrade por Inutilização de Recursos . . . . . 27
14 Dashboards do Kibana . . . . . 27
15 Dashboard da Aplicação 1 . . . . . 27
16 Dashboard do Router 2 . . . . . 28
17 Dashboard da Shell . . . . . 28
18 Dashboard do Nginx 1 . . . . . 29

# 1 Introdução

Inserido no plano curricular da disciplina de Gestão de Infraestruturas de Computação, do curso de Mestrado em Engenharia Informática, da Universidade de Aveiro e leccionada pelo professor João Barraca, este relatório é proveniente da implementação da aplicação *Easy!Appointments* como um serviço, estabelecimento de um SLA e monitorização da implementação para verificar o cumprimento do mesmo.

O projeto tem como objetivo demonstrar na prática a operacionalização em produção da aplicação *Easy!Appointments* e o relatório documentar as decisões de implementação, o funcionamento da aplicação, o SLA definido, o tipo de monitorização e alertas e as estratégias de aprovisionamento automático implementadas.

Na Secção 2 são apresentados os objetivos definidos para o projeto, na Secção 3 encontra-se a explicação detalhada do software *Easy!Appointments*, o cenário alvo considerado e o seu funcionamento distribuído. Na Secção 4 é feita a descrição do SLA a ser respeitado e do software que será usado para alcançá-lo e monitorizá-lo. Na Secção 5 é abordada a implementação do diversos componentes de forma distribuída, como comunicam e como alcançam os valores estabelecidos no SLA. Na Secção 6 é analisada a implementação dos mecanismos de monitorização e de automação de aprovisionamento de novos elementos. Na Secção 7 são expostos os testes realizados e os resultados obtidos. Por fim na Secção 8 é feita uma breve nota sobre os resultados obtidos, desafios encontrados e oportunidades não exploradas.

Todo os ficheiros e configurações utilizados podem ser encontrados em https://github.com/rgtzths/GIC_final .

# 2 Objetivos

Uma vez que o o propósito do projeto é a operacionalização e monitorização completa da aplicação os seguintes objetivos foram definidos:

- Definir e configurar imagens *Docker*;
- Gerir e distribuir os containers das imagens utilizando o *Docker swarm*;
- Definir o cenário de uso pretendido e o SLA que o garante;
- Implementar mecanismos de replicação e load balancing para alcançar o SLA;
- Criar um simulador de carga para testar o cenário estabelecido.
- Implementar mecanismos de monitorização, alarmistica e aprovisionamento automático.

# 3 Produto

O produto que vai ser disponibilizado chama-se *Easy!Appointments* e como o nome indica tem como objetivo simplificar o agendamento de marcações.

A aplicação apresenta diversas vantagens para os seus diversos stakeholders. Nomeadamente, para os clientes de uma dada organização, a aplicação permite que estes façam marcações, de forma simples e rápida, acedendo a uma página web. Para a organização esta permite a definição do horário dos funcionários, dos serviços e de quem os fornece. Juntando esta informação para gerar as vagas para marcações automaticamente. Ao funcionário permite a gestão dos clientes e das suas marcações, para uma orientação fluída do seu horário. Para além disto a aplicação inclui ainda sincronismo com o *Google Calendar*.

Para a equipa de desenvolvimento e operacionalização a aplicação oferece ainda uma API. Esta API é relevante uma vez que permite estender a aplicação sem necessidade do conhecimento da implementação da mesma, tornando o software relevante no mercado.

Tendo em conta estas características a aplicação parece trazer valor a qualquer empresa que trabalhe com marcações. As marcações podem ser individuais ou em grupo, alcançando assim, não só áreas como advocacia, ou psicologia, mas também centros de explicações ou terapias de grupo.

## 3.1 Cenário Alvo

O cenário alvo do projeto passa pela disponibilização da aplicação como um serviço à Universidade de Aveiro.

O objetivo da univerisdade é tornar digital todo o processo de preenchimento de vagas para atendimento ao público prestado pelos seus funcionários nos Serviços de Ação Social e nos Serviços Académicos, com possível extensão aos docentes.

Mudando o cenário atual, onde um aluno se dirige presencialmente ao serviço para obter uma senha e esperar pela sua vez. Para um modelo onde um aluno faz a marcação com um dos prestadores de serviço disponíveis à hora que deseja em qualquer lugar.

Para além de eliminar as filas de espera, que os estudantes enfrentam, vai ajudar também os funcionários, uma vez que, com a marcação limitada temporalmente e um número de marcações diárias estabelecidas, a sobrecarga que os funcionários encontram em certos dias, será diluída por vários dias. Permitindo assim uma gestão entre as tarefas de atendimento e de escritório muito mais eficiente.

Com os objetivos de negócio estabelecidos, o cenário alvo considerado consiste em 500 prestadores de serviço, que prestam serviço em um ou mais serviços. Como serviços temos 50 departamentos (representativos da sua secretaria) e 8 serviços representativos das opções de atendimento dos serviços académicos e de ação social. E é esperado, que no máximo, a plataforma tenha 500 utilizadores em simultâneo, agrupando prestadores de serviço e alunos que realizam marcações.

## 3.2 Composição do Produto

O produto base é composto por um servidor web Apache/Nginx e uma base de dados MySql.

A implementação do produto é feita em PHP usando o servidor Apache para receber os pedidos e a base de dados MySql para persistir os dados. A aplicação pode ser implementada com recurso a docker, sendo que uma imagem base é disponibilizada pelo autor. O autor com recurso a docker-compose implementa ainda uma stack base composta pela image do servidor e uma image da base de dados MySql. Sendo que estes são considerados os requisitos mínimos da aplicação.

A documentação oferecida pelos autores da aplicação não menciona a execução distribuída da aplicação nem o suporte da mesma, sendo que a única distribuição possível à partida é a divisão entre o servidor web e o armazenamento.

# 4 Estratégia

Uma vez que o propósito deste projeto é disponibilizar a aplicação *Easy!Appointments* como um serviço, é necessário estabelecer um SLA (Service Level Aggrement) com a universidade e uma estratégia que explica como é que este é alcançado.

## 4.1 SLA Estabelecido

O SLA estabelecido tem como objetivo dar garantias mínimas de disponibilidade e performance do sistema *Easy!Appointments* à organização contratadora, Universidade de Aveiro.

O SLA está dividido em dois componentes, o SLA para o produto em si e os valores para os serviços do mesmo.

### 4.1.1 Serviço

Considerando o âmbito do projeto, os seguintes pontos foram definidos:

- O fornecimento do produto é feito através do acesso a um web browser;
- Qualquer problema de usabilidade da aplicação não é da responsabilidade do fornecedor;
- A disponibilidade esperada do serviço é 100% em atividade regular e de 90% em atividade extrema.
- Os tempos de resposta para pedidos GET ou POST não devem ultrupassar os 5 segundos em atividade regular e os 7 segundos em atividade extrema.

### 4.1.2 Componentes Individuais

Para garantir o cumprimento do SLA são necessários alguns SLO (Service Level Objects)- métricas a serem monitorizadas - representativas do estado da stack e alguns SLIs, que são limites ou funções aplicados aos valores extraídos pelos SLOs.

A Tabela 2 apresenta os SLOs utilizados para monitorizar o sistema e os respetivos SLIs.

## 4.2 Distribuição de Carga e Redundância

Para alcançar os valores estabelecidos no SLA são necessários mecanismos de distribuição de carga, dado que, apenas uma instância da aplicação pode não ser suficiente para a carga prevista e escalar uma aplicação horizontalmente é muito

menos dispendioso que verticalmente. E são necessários ainda mecanismos de redundância, para que seja possível resistir a falhas planeadas, como manutenção dos servidores e não planeadas, como um disco se estragar.

### 4.2.1 Nginx

O Nginx é um servidor leve de HTTP que pode funcionar como um balanceador de carga, sendo um proxy entre os pedidos dos clientes e os diversos servidores disponíveis. Na instalação o Nginx será o ponto de acesso à aplicação web, funcionando como um balanceador de carga e um distribuidor de ficheiros estáticos.

| Categoria | Métrica | Limite |
|---|---|---|
| Hardware (SNMP e Telegraf) | Up Time Sistema | - |
| | Ram Usada | - |
| | Ram Disponível | $> 1G$ |
| | IO Envidada | - |
| | IO Recevida | - |
| | Uso CPU (user) | - |
| | Uso CPU (sistema) | $< 90\%$ |
| | Disco Disponível | - |
| | Disco Usado | $< 90\%$ |
| | Rede Usada (recebida) | - |
| | Rede Usada (envidade) | - |
| NginX | Clientes em Espera | $< 50$ |
| | Erros de Cliente | 0 |
| | Cleintes Connectados | - |
| | Nº de Pedidos | - |
| Apache | Clientes em Espera | $< 100$ |
| | Clientes Ativos | - |
| | Clientes em Leitura | - |
| | Throughput | - |
| MySql | Nº Conexões (Ativas) | $< 1000$ |
| | Nº Conexões (Execução) | $< 10$ |
| | Dados Recebidos | - |
| | Dados Enviados | - |
| | Nº de Quereis/s | - |
| | Queries Lentas | 0 |

Tabela 1: SLIs e SLOs Avaliados

Uma vez que, serão utilizadas duas instâncias do servidor Apache ativas, o Nginx vai permitir balanceamento de carga entre as duas instâncias e ainda redundância, dado que, caso uma falhe os pedidos serão direcionados para a outra.

### 4.2.2 MySQL InnoDB

A InnoDb é um mecanismo de armazenamento para MySQL que tem como objetivo alcançar alto desempenho e disponibilidade. Na instalação a InnoDB é composta por um router, três servidores e um configurador.

O router é o elemento responsável por receber o tráfego e o distribuir pelos três servidores. Os três servidores criam entre si um grupo de replicação na topologia master-slave. Nesta topologia apenas o master recebe comandos de escrita, mas qualquer uma das instâncias pode receber operações de leitura. O configurador é responsável por configurar o grupo de replicação indicando aos servidores onde estão os restantes elementos.

A divisão entre operações de leitura e escrita é feita no router, através de diferentes portos de acesso.

O uso dos três servidores garante redundância e distribuição de carga sendo que caso o master falhe, um dos slaves assume o seu papel e as diversas operações de leitura podem ser distribuídas.

### 4.2.3 Docker Swarm

O docker swarm pertence à plataforma docker. A plataforma docker permite virtualização ao nível do sistema operativo. O docker swarm apresenta-se como um grupo de nós docker distribuídos que executam stacks de containers. Na implementação o docker swarm é utilizado para disttribuir a aplicação, replicar o Nginx e o MySql router e ainda fazer balanceamento de carga dos últimos dois.

Uma vez que, o docker swarm usa várias instâncias distribuídas e coloca nelas os serviços, este vai eliminar o problema de ponto de falha único que seria, colocar todas as instâncias numa máquina.

Na aplicação, o swarm vai replicar os serviços de Nginx e MySql router, pois são os únicos que, para instâncias diferentes não necessitam de configurações diferentes. A replicação inicial é de duas instâncias por serviço, garantindo assim a redundância. Contudo, caso algum destes serviços fique sobrecarregado o aprovisionamento automático de mais instâncias do serviço foi configurado. No aprovisionamento automático, o sistema de balanceamento de carga do swarm vai garantir que a nova instância recebe também a sua parte dos pedidos, de forma transparente para os restantes componentes.

A remoção automática de instâncias também foi criada para evitar utilização desnecessária de recursos.

## 4.3 Monitorização, Alarmística e Recuperação de Falhas

Embora uma estratégia tenha sido delineada para garantir o cumprimento do SLA esta não é suficiente uma vez que falhas podem acontecer ou a estratégia pode não conseguir alcançá-lo. Nesse sentido são necessários mecanismos que monitorizam a aplicação que em caso de falha do serviço criem alarmes que notificam a equipa de operacionalização e caso seja possível recuperem das mesmas.

## 4.4 Docker Swarm

Uma vez que quando uma falha ocorre, esta deve ser tratada o mais rápido possível, é importante que mecanismos de recuperação automática do sistema sejam implementados. O docker swarm possui uma característica que permite tratar desse problema. Essa característica é o reinicio automático de um componente caso este ou a máquina onde ele está alojado falhe. Uma vez que o swarm contém um serviço de DNS a recuperação é transparente aos restantes componentes, que apenas usam o nome do componente para comunicar.

### 4.4.1 TICK Stack

A TICK Stack é uma stack de monitorização que permite obter métricas de diversos sistemas. A stack é composta pelo Telegraf, que é reponsável por obter as métricas nas máquinas a monitorizar. Pela InfluxDB, que armazena os dados vindos do telegraf numa base de dados *time series*. Pelo Chronograf, que é uma aplicação web que consome os dados armazenados na InfluxDB, os apresenta graficamente e permite definir alarmes no Kapacitor. E pelo Kapacitor que, consumindo os dados da InfluxDB desencadeia alarmes caso os valores definidos excedam um dado limite.

Na implementação a TICK Stack vai recolher e tratar os dados de funcionamento de todas os containers docker da stack, nomeadamente, dados sobre o uso de cpu, ram, swap e rede.

Para além destes dados, de acordo com o tipo de imagem, Nginx, Aplicação ou MySql Server, vai recolher métricas sobre Nginx, Apache Server e MySql Server respetivamente.

E uma vez que a aplicação pode não estar sobrecarregada mas a máquina sim, os dados sobre as máquinas onde os containers estão a correr são recolhidos através de SNMP, para analisarmos a carga da máquina onde o container está a correr.

Os dados coletados serão utilizados para construir dashboards para visualização gráfica da informação, alarmes na forma de mensagens no slack, caso os valores do SLA sejam violados e alarmes na forma de triggers de aprovisionamento automático, caso os alarmes se refiram a sobrecarga do Nginx e do MySql router.

### 4.5 ELK Stack

A obtenção de métricas é um bom indicador do desempenho de uma aplicação, contudo se a aplicação falhar é importante saber porque é que o mesmo aconteceu.

Para que seja possível ter um histórico do que aconteceu é importante mantermos os logs que os diversos componentes emitem durante a sua execução.

Para gerar este histórico e para também obtermos informação extra dos mesmos podemos usar a ELK Stack que é composta pelo Logstash que recebe os logs num formato pré-definido, os processa, obtendo os campos relevantes para análise e os insere no Elasticsearch. O Elastisearch é um outro componente da stack que armazena os logs e permite a consulta dos mesmos de forma eficiente. Por fim temos o Kibana que é uma aplicação web que consome os dados do elastic search, permite definir visualizações dos mesmos para análise e definir alarmes costumizados.

Na implementação os logs são obtidos com recurso à driver de logging do docker que obtém os logs em formato syslog. Os dados obtidos em syslog são enviados para o Logstash que faz o tratamento dos mesmos e os envia para o Elasticsearch. Por fim no Kibana é possível analisar os logs nas diferentes dashboards.

## 5 Instalação

Uma vez definido o SLA e a estratégia de cumprimento do mesmo passamos à instalação. A instalação planeada é composta por dois proxies Nginx, que funcionam como balanceadores de carga, dois servidores apache que correm a aplicação web, dois MySql routers que fazem a distribuição de pedidos, um MySql Shell para configurar as BDs e por fim três MySql Servers para armazenar os dados. Para comunicação entre componentes foram definidas três redes, estas não só garantem isolamento dos componentes, como permitem entender, de uma forma clara, a interação dos componentes.

As redes definidas são:

- Frontend: Entre os servidores Apache e os proxies Nginx;
- Backend: Entre os sevidores Apache e os MySql routers;

- Database: Entre os MySql routers, servers e Shell.

A arquitetura da solução pode ser vista na Figura 1.

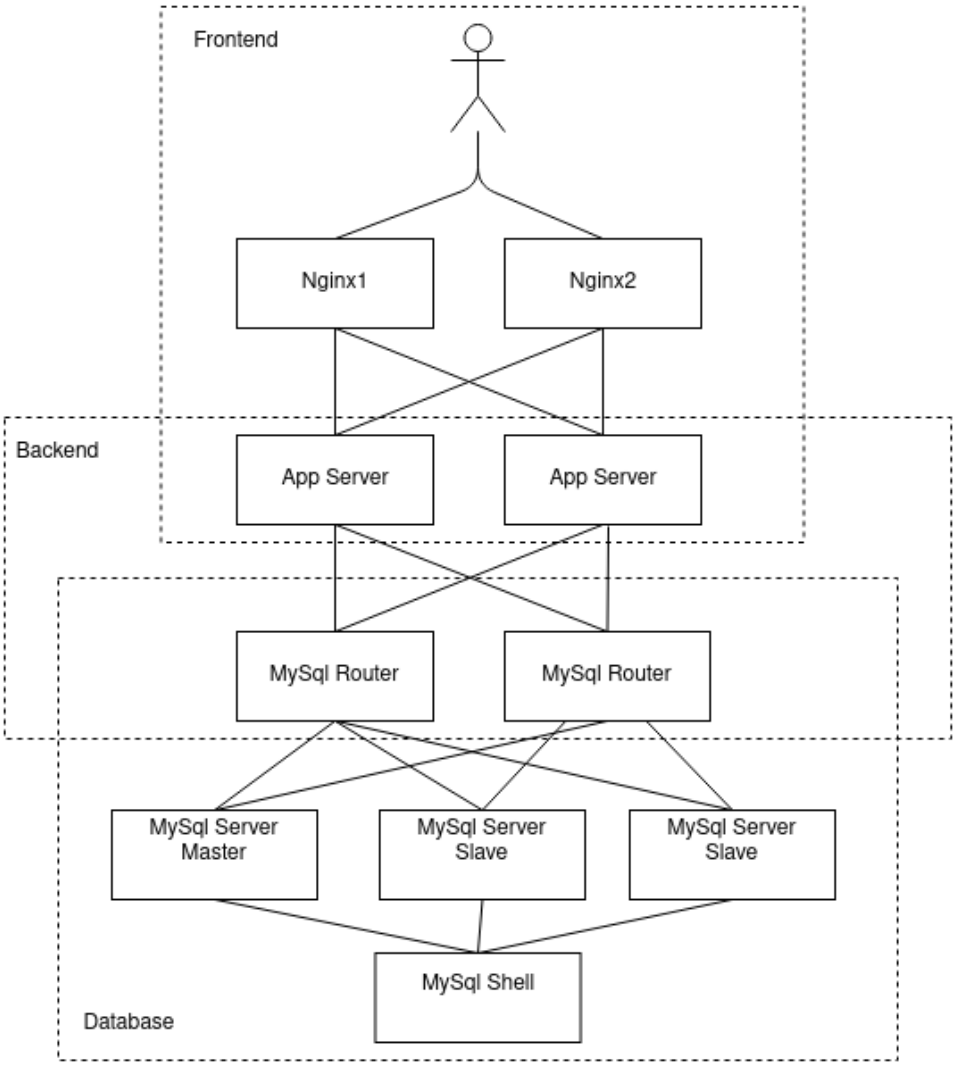


Figura 1: Arquitetura da Solução

## 5.1 Docker Swarm

Como mencionamos anteriormente na Subsecção 4.4, a nossa instalação terá como base o docker swarm.

Este swarm é composto por um cluster com um total de 58 nós, 115 cores, 5.6T de disco e 235.6G de ram. Numa primeira análise, este swarm aparenta ser

até demasiado poderoso para o cenário de utilização esperado, contudo o swarm não será utilizado exclusivamente para este projeto, pelo que a partilha destes recursos é necessária. Nesse sentido, consideramos que a aplicação poderá consumir perto de 5% dos recursos totais disponíveis.

As redes estabelecidas na instalação são virtualizadas pelo docker e a conexão entre os componentes é feita através do nome a aplicação com recurso ao DNS do swarm. Por este motivo é indiferente para os componentes a localização dos restantes serviços nos diversos nós do cluster.

Uma outra vantagem do swarm é a possibilidade da utilização de configs e secrets.

As configs são ficheiros de configuração, que podemos criar no swarm e podem ser aplicados a uma qualquer imagem. Estes ficheiros são úteis, uma vez que, só são inseridos quando inicializamos uma stack. Isto permite alterar as configurações sem ter de reconstruir as imagens. Na instalação as configs são utilizadas para passar as diversas configurações do telegraf e a configuração do nginx.

Os secrets funcionam de forma semelhante às configs contudo estes protegem informações sensíveis como passwords, não as deixando visíveis. Estes ficheiros evitam que os dados sensíveis sejam escritos em texto nas configurações e acabem por ficar disponíveis publicamente. Na instalação os secrets foram utilizados para guardar os utilizadores da base de dados, as suas passwords e a base de dados usada pela aplicação.

## 5.2 NFS

Uma vez que, o swarm coloca um componente em qualquer um dos nós e em caso de falha, o swarm pode realocar o componente num outro nó, a persistência dos dados não é garantida.

Para garantir a persistência dos dados foi necessário utilizar o NFS disponibilizado.

O NFS é um protocolo de sistema de ficheiros distribuído e permite que consigamos utilizar um sistema de ficheiros remoto à máquina onde o componente está a ser executado. Uma vez que todos os nós do cluster conseguem aceder ao NFS, os containers utilizam o NFS como repositório de dados persistente.

Para a instalação foi criada uma pasta que engloba todos os componentes. Dentro desta pasta temos uma pasta para cada MySql Server, uma pasta para cada aplicação, uma pasta para os ficheiros estáticos disponibilizados pelo nginx, uma

pasta para os dados do MySql router e uma pasta que guarda os scripts do MySql Shell.

## 5.3 Nginx

Como mencionamos anteriormente na Subsecção 4.2.1 o Nginx funciona como um balanceador de carga entre as duas aplicações que vamos disponibilizar.

Isto é conseguido graças à definição de um upstream, que balanceia a carga entre os servidores definidos no mesmo, com base na hash do seu IP.

O Nginx é necessário como balanceador de carga, uma vez que, a aplicação se apresenta como Statefull, ou seja, os dados sobre as conexões ativas são mantidas no servidor. Para garantir o funcionamento correto da aplicação é necessário um balanceamento de carga que mantenha as sessões, algo que o Swarm não é capaz de fazer.

Para além do balanceamento de carga, o Nginx é ainda responsável por distribuir os ficheiros estáticos da aplicação. Desta forma, evitamos que algumas conexões desnecessárias sejam feitas à aplicação, fornecendo mais recursos para responder aos restantes pedidos.

## 5.4 Aplicação

A aplicação implementada como foi referido no sub capítulo anterior é statefull. Por este motivo não podemos utilizar o sistema de replicação fornecido pelo docker swarm, pois este não garante que uma mesma conexão vá sempre para a mesma aplicação.

Sendo assim duas instâncias foram definidas manualmente para que o nginx realiza-se o balanceamento de carga entre as duas e a redundância mínima fosse alcançada.

## 5.5 InnoDB

A implementação da InnoDB foi sem dúvida a parte mais desafiante do projeto.

Embora as imagens docker que configuram uma InnoDB sejam disponibilizadas, as mesmas tinham alguns problemas, levando a falhas frequentes durante a configuração da mesma. Contudo, após algumas alterações a taxa de sucesso de configuração melhorou bastante.

O maior problema desta implementação é não ser possível a replicação automática das bases de dados, uma vez que estas necessitam ser adicionadas ao clus-

ter através de um Id único e o MySql Shell tem de conhecer o número de instâncias a configurar e o seu nome. Contudo não consideramos que esta implementação seja um entrave uma vez que o uso de três bases de dados garante a redundância mínima necessária e uma base de dados apenas é capaz de aguentar milhares de escritas por segundo.

Sendo assim, após garantirmos a redundância mínima do MySql router através da réplica do docker swarm, a InnoDB tem não só alto desempenho, como alta disponibilidade.

## 5.6 Imagens Docker

Embora todos os componentes tivessem imagens base disponíveis, todas as imagens tiveram de ser alteradas.

A imagem da aplicação teve de ser alterada para instalar o telegraf e suportar o uso de secrets. O mysql router foi alterado para instalar o telegraf, suportar secrets e criar mecanismos de espera pelos restantes componentes. O mysql shell foi alterado para suportar secrets e criar mecanismos de espera pelos restantes componentes. Os MySql servers foram alterados para instalar o telegraf e suportar secrets. E o nginx foi alterado para instalar o telegraf e esperar pelos restantes componentes.

# 6 Operação

Depois de instalado o projeto é necessário monitorizá-lo. Como referido na Subsecção 4.3 isto pode ser feito com recurso à monitorização de métricas do sistema, que será feito pela TICK Stack e com recurso à monitorização dos logs que será feito pela ELK Stack.

Com base na análise de métricas foram ainda criados alarmes e mecanismos de aprovisionamento/desaprovisionamento automático de instâncias.

## 6.1 Mecanismos de Monitorização e Alarmística

Nesta subsecção vamos abordar as métricas extraídas dos diferentes componentes, como os obtivemos e os alarmes criados para responder aos limites estabelecidos pelo SLA.

### 6.1.1 Telegraf

O telegraf é a aplicação responsável por obter as métricas das diferentes aplicações e dos recursos gastos pelas instâncias docker. Para isso, o telegraf foi

instalado em cada imagem docker e com recurso aos seus plugins definimos várias métricas a ser extraídas. A Tabela 2 indica que métricas foram extraídas dependendo do tipo de imagem onde foi instalado.

| | System | CPU | NET | RAM | DISK | DiskIO | Processes | SWAP | Apache | Nginx | MySql |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Aplicação | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - | - |
| Nginx | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - | ✓ | - |
| Server | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - | - | ✓ |
| Router | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - | - | - |

Tabela 2: Métricas extraídas em cada componente.

As métricas são enviadas para uma VM onde estão instalados os restantes componentes da stack, sendo que os dados são enviados por upd de 1 em 1 minuto para evitar congestionamento extra da rede, mas ao mesmo tempo, conseguir criar alarmes relevantes e atempadamente.

### 6.1.2 SNMP

A extração das métricas snmp foi ela também desafiante, uma vez que não sabemos em que máquina vai estar a executar um dos componentes da stack e não faz sentido extrair informação de todas as possíveis máquinas.

Então, para extrairmos informação apenas das máquinas que são relevantes, começamos por criar um servidor em cherrypy, que aceitava conexões e recebia o nome da aplicação como argumento. Através dessa conexão o servidor criava um mapeamento serviço - IP fonte da comunicação. Assim obtínhamos o IP no qual estava a ser executado o componente.

Depois de termos os IPs dos diferentes serviços, com recurso à biblioteca pysnmp obtivemos as métricas snmp das diferentes máquinas. Nomeadamente, o uptime do servidor, uso de CPU, memória usada e disponível, uso de I/O e uso de disco.

Agora que os dados foram extraídos é necessário inseri-los na base de dados. Nas restantes métricas o telegraf faz isso automaticamente, mas neste caso foi necessário o uso da biblioteca da influxDb.

A recolha e inserção dos dados na base de dados é feito numa thread cíclica que insere os dados de 1 em 1 minuto. Esta thread é independente do servidor cherrypy, sendo que ambos partilham apenas um dicionário com os serviços e os respetivos IPs.

Os dados coletactados são apresentados na dashboard do sistema no Chronograf.

### 6.1.3 InfluxDB, Chronograf e Kapacitor

A InfluxDB, o Chronograf e o Kapacitor foram instalados numa máquina virtual separada da instalação. Na InfluxDB foi criada uma base de dados para receber todas as métricas extraídas. No Chronograf foram criadas várias dashboards. Uma para dados Nginx, uma para dados Apache, uma para os dados do Sistema e uma para dados MySql. No Kapacitor foram criados alarmes para notificação da equipa de operação e para aprovisionamento automático.

As dashboards criadas são importantes pois oferencem-nos um histórico das medidas avaliadas pelo SLA permitindo adaptar a estrutura à realidade da carga experienciada.

Os alarmes são necessários, pois em casos de emergência o sistema tem de ser capaz de alertar as equipas. Estes alarmes consistem em mensagens enviadas para um canal de slack especifico e notificam quando o sistema entra em estado crítico e quando entra em estado normal. Os alarmes são referentes às medidas estabelecidas no SLA visto na Subsecção 4.1.

Os dashboards, alarmes e notificações geradas podem ser vistas nas Figuras presentes em 10.

### 6.1.4 Syslog

O syslog é um padrão para transmisão de mensagens log e é usado na comunicação das mensagens dos componentes do projeto para o logstach. A recolha das mensagens de log é feita pela drive de logging que o docker disponibiliza e todas os compenentes a usam. Para evitar congestionamento as mensagens são enviadas em udp.

## 6.2 Logstach, Elasticsearch e Kibana

A ELK Stack está instalada na mesma VM que os componentes da TICK stack.

O logstach ao receber as mensagens no formato syslog, retira várias informações como, quando foi enviada, gravidade da mensagem, a prioridade, o programa que a enviou, entre outras.

Este processamento é feito através de filtros, começando por usar o grok para obter os campos da mensagem, de seguida matches para os alterar e no final os campos que não são necessários são removidos.

Estas informações são inseridas no elasticsearch num índice, o índice em que o

logstach insere os dados corresponde ao dia em que os dados são enviadas. Criando assim um índice por dia.

No Kibana temos acesso aos logs após todo este processamento. Para simplificar a visualização dos mesmos foi criada uma dashboard para cada componente do sistema. Onde apenas os logs desse componente aparecem e se podem fazer filtros temporais aos mesmos.

As dashboards criadas podem ser vistas nas Figuras presentes em 10

## 6.3 Mecanismos de Automação

Para além de alertar a equipa de operação de possíveis problemas com a stack é importante que o sistema tenha autonomia para realizar atividades de aprovisionamento para responder a picos de utilização e desaprovisionamento em casos de uso desnecessário de recursos.

### 6.3.1 Nginx e MySql Router

O Nginx e o MySql router, como vimos anteriormente foram replicados com recurso ao docker swarm. Esta característica permite o aprovisionamento/desaprovisionamento automático destes componentes.

Para o realizar, o servidor utilizado para monitorizar o SNMP foi extendido, adicionando uma nova chamada ao cherrypy, que recebe como argumento qual a aplicação que necessita de ser escalada ou reduzida e através da API do docker para python executa o aprovisionamento/desaprovisionamento da mesma.

Para chamar este método, foram utilizados alarmes no Kapacitor que fazem a chamada http indicando o serviço e se necessita mais ou menos instâncias. Os alarmes ocorrem quando o Nginx tem mais de 50 utilizadores à espera e quando o MySql Router tem um uso de CPU do sistema superior a 90 %.

### 6.3.2 MySql Server e Aplicação

Dado as características de instalação do MySql Server e da Aplicação, estas não podem ser escaladas automaticamente uma vez que têm de ser configuradas de forma única.

Para escalar os serviços a stack tem de ser reiniciada pelo que quando colocada em produção os valores utilizados terão de ser suficientes para aguentar a carga prevista e eventuais picos.

# 7 Testes

## 7.1 Considerações Iniciais

Uma vez que, o balanceamento de carga é feito através de uma hash do IP e como apenas um computador foi usado para simular a carga, todos os pedidos enviados incidiram numa mesma instância da aplicação.

Sendo assim, os resultados obtidos referem-se a apenas uma aplicação. Criando previsões para o número de réplicas para os restantes cenários com base na capacidade de uma aplicação.

Outras considerações a ter são o congestionamento da rede do swarm, uma vez que outras aplicações são executadas no mesmo e o atraso causado pela comunicação através das duas vpns, que são necessárias para aceder ao swarm. Estas duas limitações vão criar atrasos na comunicação, não só com o cliente mas também internamente entre os componentes.

Uma vez que estas limitações não podem ser ignoradas, um primeiro teste foi realizado com apenas um cliente para obter o valor mínimo possível ideal do tempo de resposta à implementação. Este valor é aproximadamente 2.5 segundos para realizar todas as comunicações presentes no workflow.

## 7.2 Preparação do Ambiente

A preparação do ambiente foi realizada com recurso a selenium uma vez que é suposto ser uma tarefa que ocorre apenas uma vez.

A preparação teve em conta o cenário alvo considerado na Subsecção 3.1, ou seja, foram criados 500 prestadores de serviço, 50 serviços representativos dos departamentos e 8 serviços representativos das vagas dos Serviços Académicos e de Ação Social.

## 7.3 Workflow do Teste

O workflow criado pretende representar a maioria dos acessos que vão ser realizados à aplicação. Uma vez que a universidade conta com dez mil alunos e apenas 500 funcionários foram considerados como prestadores de serviço, maioria dos pedidos realziados, serão de alunos a tentar encontrar uma vaga que encaixe no seu horário.

Nesse sentido, o workflow criado consiste na abertura da página inicial, escolha de um serviço e prestador de serviço, escolha de um dia e um horário (pode ocorrer

várias vezes) e por fim na submissão da marcação.

Uma vez que, ferramentas como selenium, embora práticas, consomem demasiados recursos para simular o número de utilizadores desejado, uma outra ferramenta teve de ser escolhida. A ferramenta escolhida é o locust, que permite definir chamadas http, definir tarefas a executar, o número de utilizadores esperado e o hatch rate dos mesmos.

Para usar o locust foi necessário fazer a coleção de todos os pedidos realizados por um browser e o mapeamento dos mesmos para pedidos get e post. Depois de realizado o resultado pode ser visto na Figura 2.

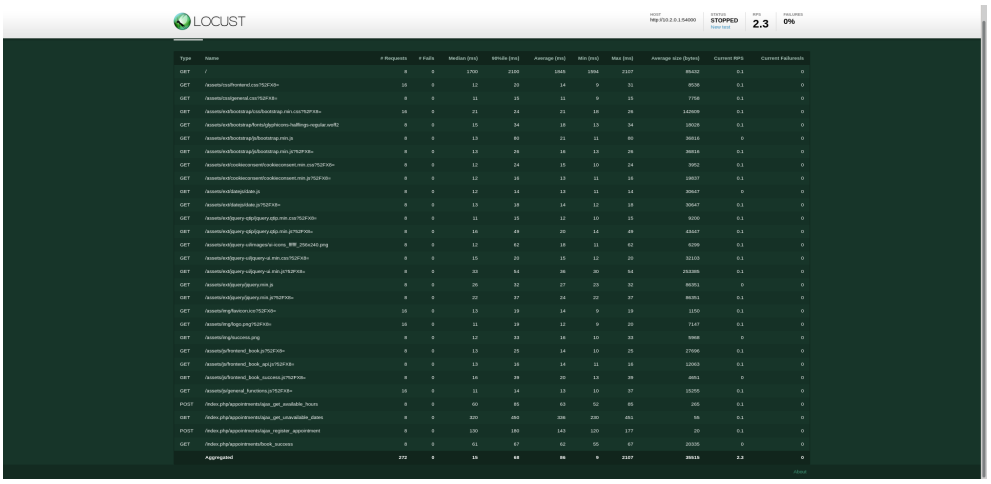


Figura 2: Chamadas realizadas para simular carga

## 7.4 Primeira Fase de Testes

Para evitar a sobrecarga repentina do sistema, o primeiro teste foi realizado com 100 utilizadores com um hatch rate de 5 por segundo. Após todos os utilizadores estarem em funcionamento, constatamos que o tempo de resposta máximo foi de 26 segundos para o pedido base da página e ainda que os pedidos de um workflow demoram aproximadamente em méida 10 segundos. Sendo que o pedido base em média demora 7 segundos a ser respondido.

Uma vez que 100 utilizadores seria um valor abaixo do esperado para esta instância, que é 250 e existem pedidos que já violam o SLA estabelecido. Foi necessário analisar o bottleneck do sistema, este encontra-se na aplicação uma vez que esta era a que encontra maior uso de recursos e clientes em espera. Por este motivo foi necessário diminuir o número de utilizadores para encontrar qual é o valor máximo que um servidor aguenta, respeitando o SLA.

Os resultados obtidos podem ser vistos na Figura 3.

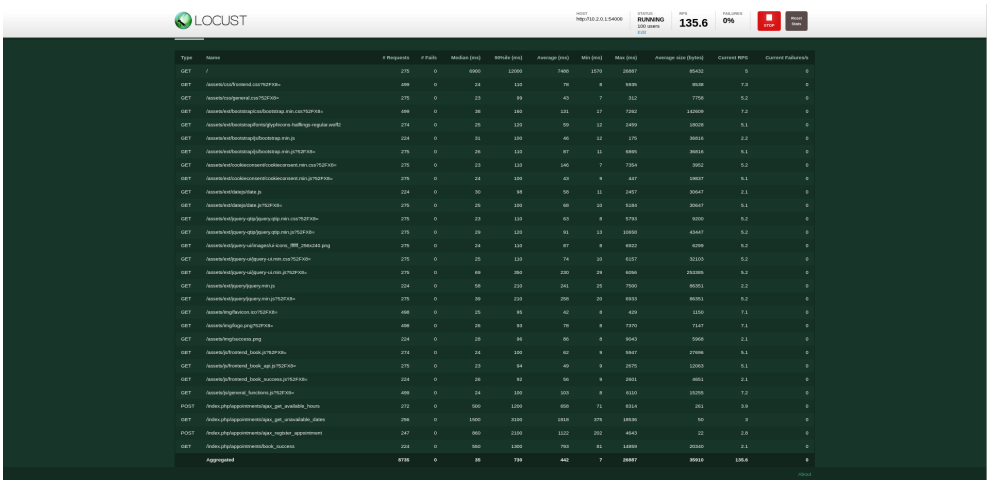


Figura 3: Resultados do teste com 100 utilizadores

## 7.5 Segunda Fase de Testes

Devido aos resultados obtidos no primeiro teste, o valor de utilizadores foi reduzido para 50, mantendo o hatch rate.

Nestas condições o tempo máximo de resposta foi de 4.2 segundos, sendo que um workflow demora em média 4 segundos.

Após obter estes resultados a capacidade máxima de um servidor da aplicação fica estimado ser 50 utilizadores de forma a cumprir os requisitos do SLA.

Os resultados obtidos podem ser vistos na Figura 4.

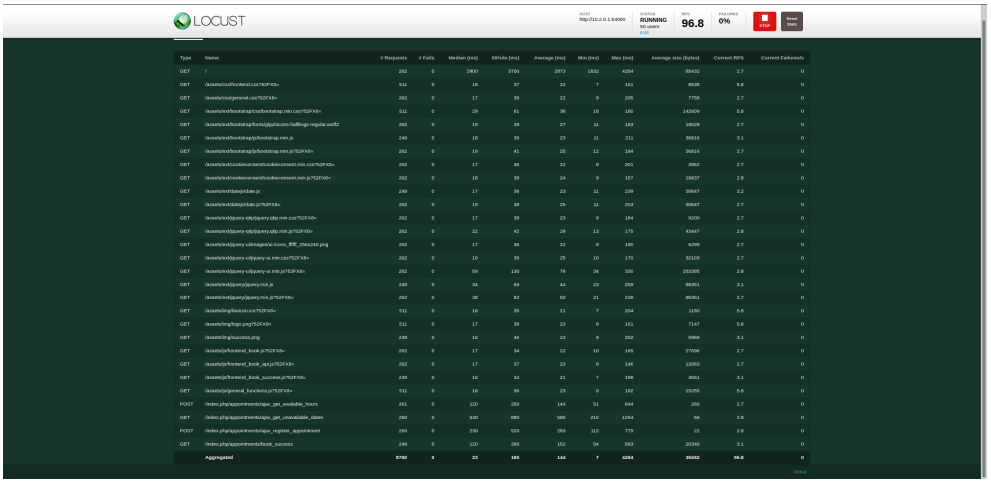


Figura 4: Resultados do teste com 50 utilizadores

# 8 Conclusão

Depois dos testes, falta apenas uma reflexão sobre os mesmos.

A primeira reflexão que sobressai é que a implementação realizada não é suficiente para respeitar o SLA definido, uma vez que, uma instância do servidor aplicacional apenas consegue lidar com 50 utilizadores. Para combater isso um aumento no número de instâncias seria necessário, ficando com 12 instâncias do servidor web e 8 instâncias Nginx para fazer o balanceamento de carga.

A segunda é que embora o desempenho do servidor da aplicação tenha ficado aquém do esperado, tanto os mecanismos de monitorização como os de aprovisionamento automático funcionaram, mostrando que estão corretamente implementados, permitem avisar as equipas de eventuais picos de utilização e tratar até certo ponto automaticamente dos mesmos. E os restantes componentes da solução mostraram eles também cargas que permitiam a escalagem para os 500 utilizadores e até mais.

A última reflexão, incide sobre o que poderia ter feito. A aplicação poderia ter sido alterada para um modo stateless, embora a falta de conhecimento sobre a implementação da mesma o impedisse. O balanceamento de carga, caso fosse usado o Nginx Pro, poderia ter sido baseado em cookies e não em ip_hash, o que daria uma maior veracidade aos testes realizados, uma vez que se assim fosse ambas as instâncias do servidor da aplicação estariam a ser utilizadas. E por fim, a aplicação poderia ter sido testada num ambiente sem tanta entropia, contudo, não foi possível por limitações nos recursos.

# 9 Referências


- Easy!Appointments -
- NginX - https://easyappointments.org/
- InnoDB - https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html
- Pysnmp - http://snmplabs.com/pysnmp/index.html
- CherryPy - https://cherrypy.org/
- ELK Stack - https://www.elastic.co/pt/what-is/elk-stack
- TICK Stack - https://www.influxdata.com/time-series-platform/
- Docker - https://www.docker.com/
- Docker Swarm - https://docs.docker.com/engine/swarm/

# 10 Anexo de Imagens das Dashboards e Alarmes

Neste anexo encontram-se as imagens dos dashboards e alertas da TICK STACK e das dashboards da ELK Stack.

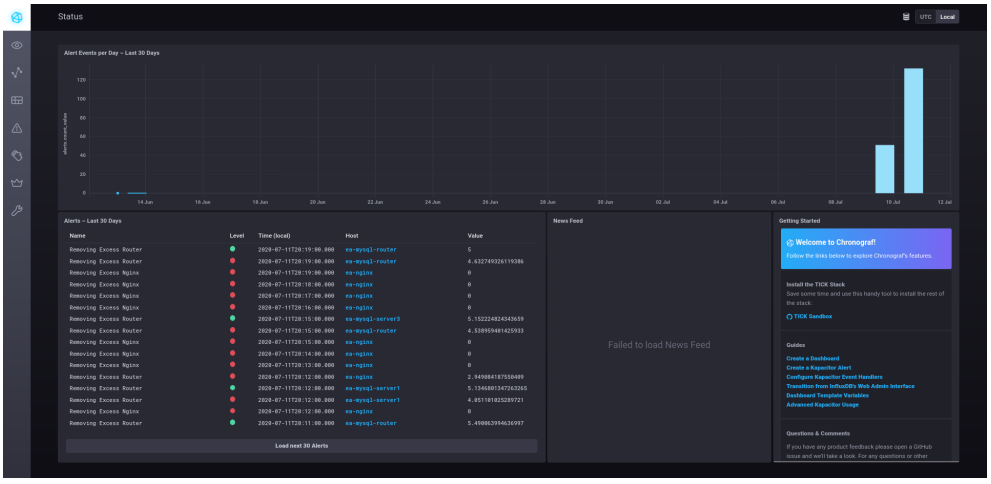


Figura 5: Home page Chronograf

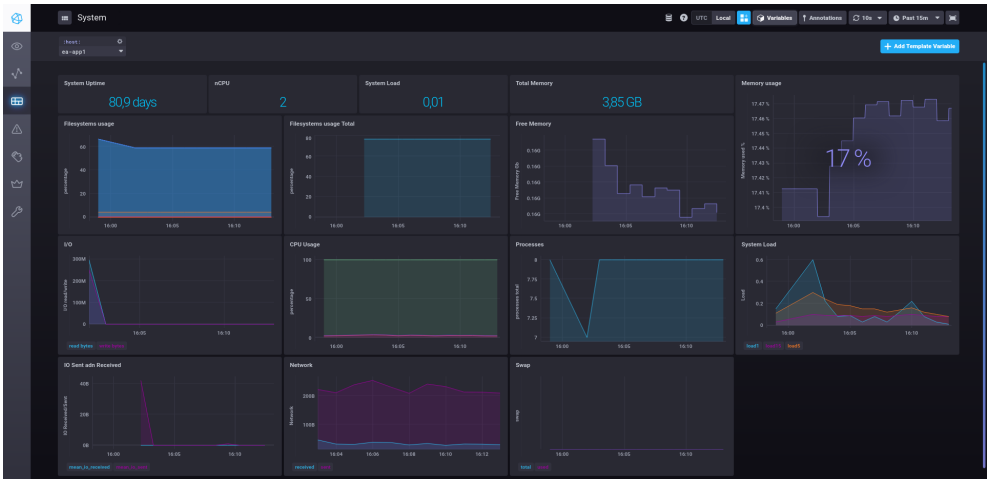


Figura 6: Dashboard dos Dados do Sistema

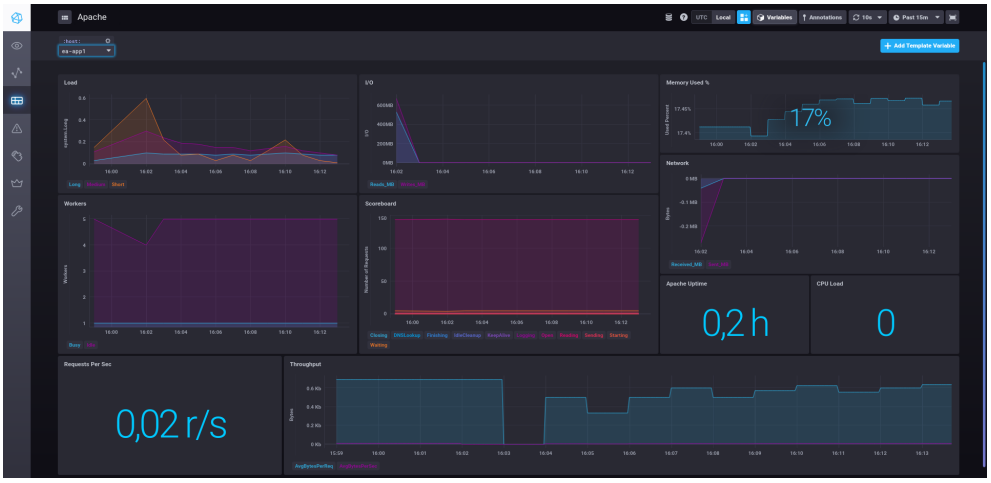


Figura 7: Dashboard dos Dados do Apache

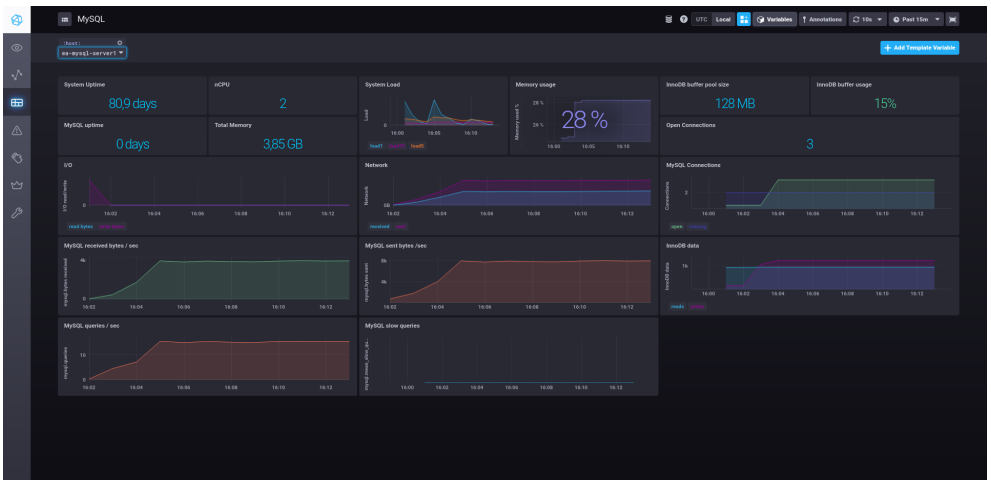


Figura 8: Dashboard dos Dados do MySql

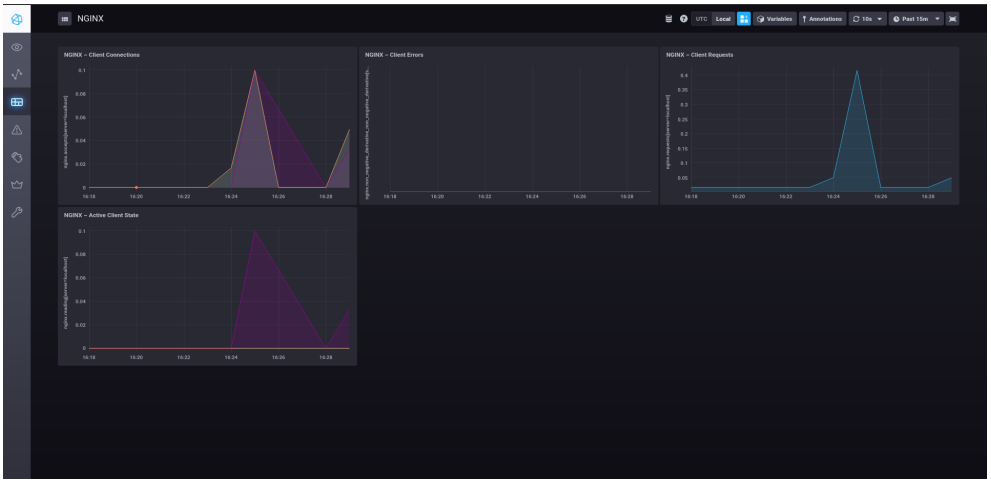


Figura 9: Dashboard dos Dados do Nginx

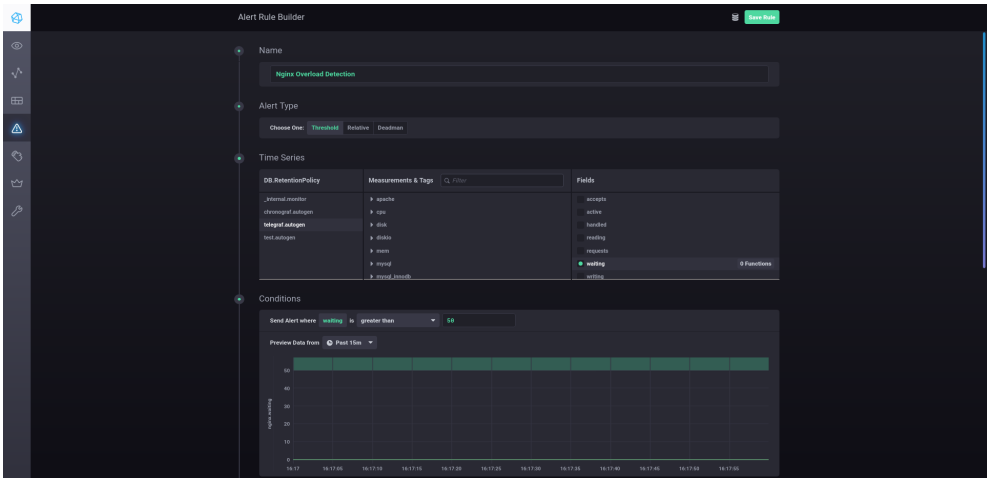


Figura 10: Configuração do alerta do Nginx

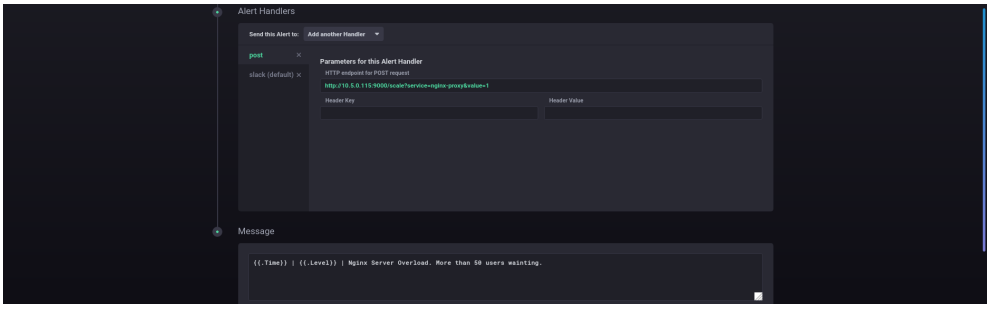


Figura 11: Chamada http para Escalagem Automática

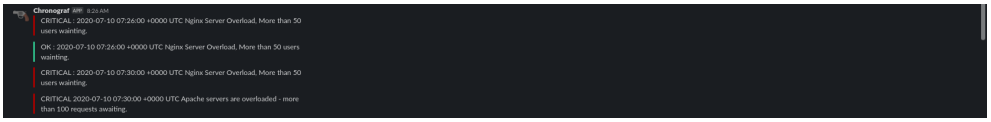


Figura 12: Mensagens de Alerta por Violação do SLA

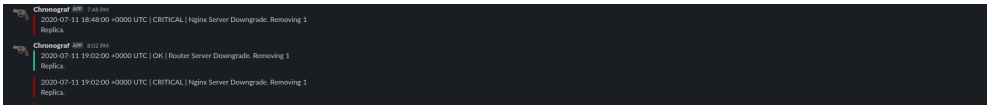


Figura 13: Mensagens de Downgrade por Inutilização de Recursos

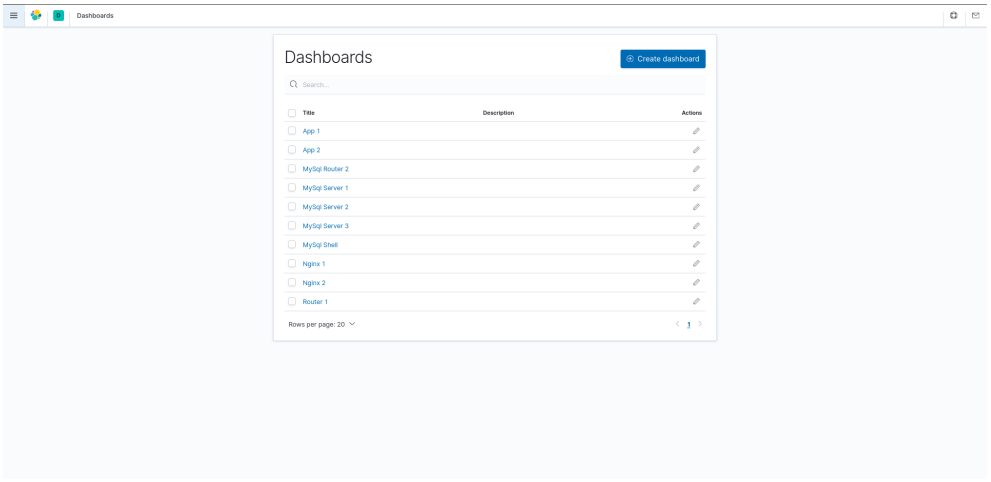


Figura 14: Dashboards do Kibana

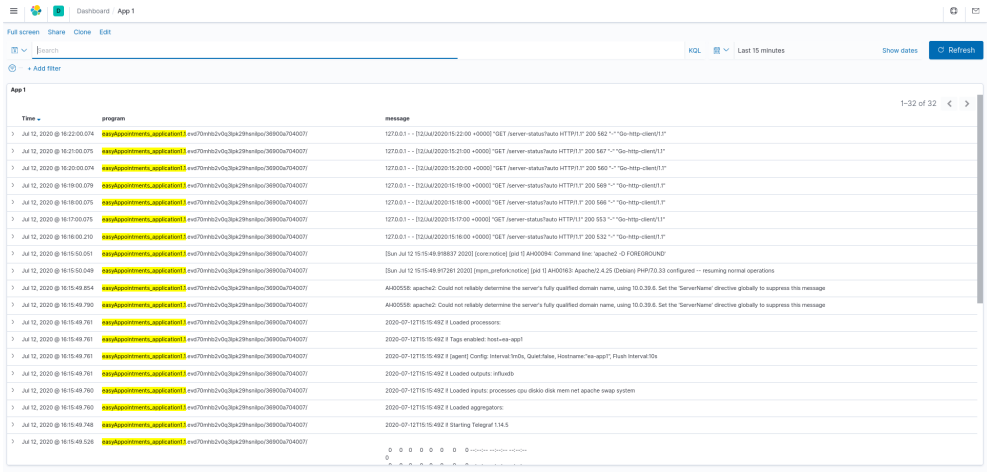


Figura 15: Dashboard da Aplicação 1

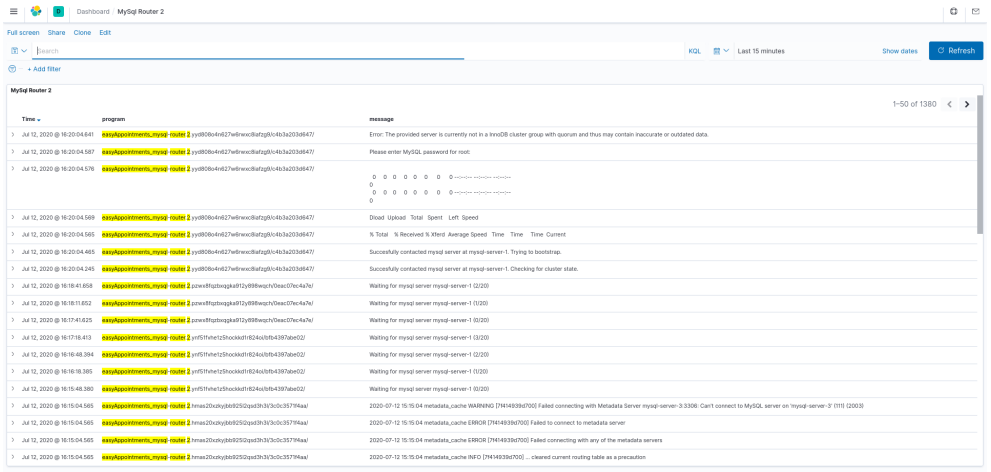


Figura 16: Dashboard do Router 2

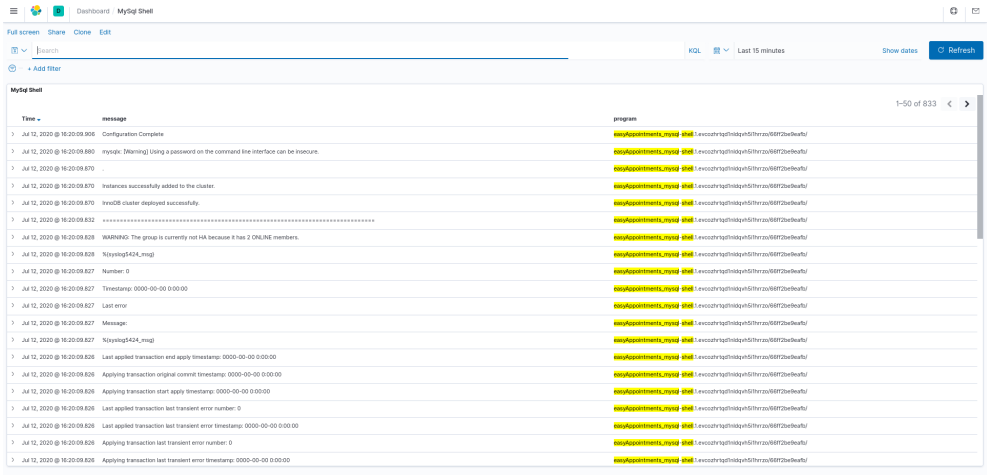


Figura 17: Dashboard da Shell

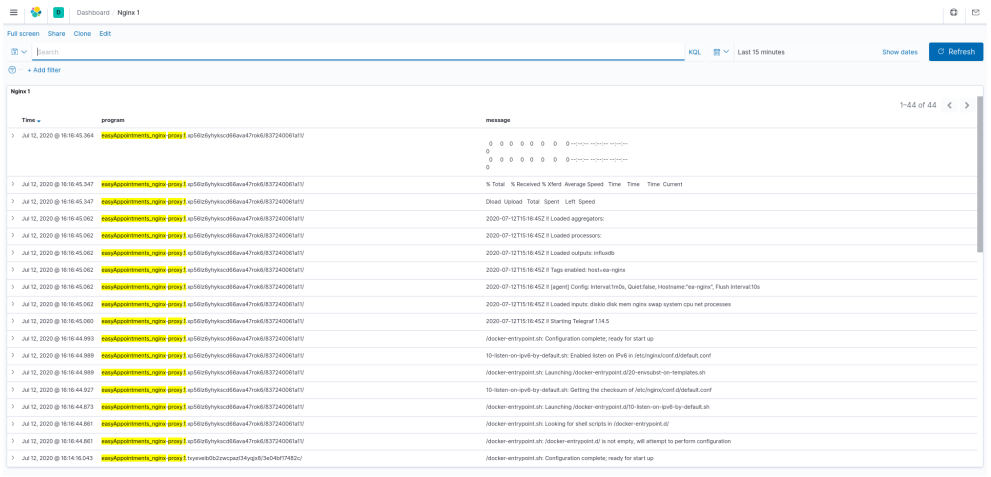


Figura 18: Dashboard do Nginx 1