Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Nuno Miguel Gameiro Dias Trindade Cardoso

Estágio PTISP | Sistemas de monitorização e alarmística

Relatório de estágio

Orientado por:

IPT: Professor Doutor Ricardo Campos
Entidade de Estágio: PTISP

Relatório de Estágio apresentado ao Instituto Politécnico de Tomar
para cumprimento dos requisitos necessários
à obtenção do grau de Mestre em Engenharia Informática

AGRADECIMENTOS

A elaboração do presente Relatório de Estágio não seria possível sem o apoio de algumas pessoas. Assim sendo, gostaria de agradecer a todos os que sempre me apoiaram e contribuíram para a realização e concretização desta etapa final da minha formação, o Mestrado de Engenharia Informática em Internet das Coisas.

Em particular, à minha família, pois tudo isto foi possível graças ao esforço e dedicação que sempre tiveram.

Ao meu orientador, Professor Ricardo Campos pela sua disponibilidade e compreensão, orientando e guiando o desenrolar do meu trabalho, manifestando sempre as suas opiniões para a elaboração deste Relatório de Estágio.

Aos colegas e diretores da PTISP empresa onde decorreu o estágio, por toda a atenção, compreensão e conhecimento transmitidos durante este meu percurso na empresa.

A todos os professores do Mestrado de Engenharia Informática em Internet das Coisas que contribuíram para o meu enriquecimento na área.

Por fim, mas não menos importante, quero também agradecer à minha namorada, que esteve sempre presente ao longo de todo este percurso e que sempre me apoiou de forma compreensiva e motivadora.

A todos o meu muito obrigado!

RESUMO

Ao longo dos últimos anos, os sistemas de monitorização e alarmística têm assumido um papel importante no seio das organizações. A sua existência permite a um administrador de sistemas ter conhecimento, em tempo real, do estado do parque informático permitindo-lhe ter uma noção mais concreta da necessidade de operar eventuais intervenções nas máquinas em questão. Garantir um sistema fiável, robusto e de alta disponibilidade, obriga, no entanto, a uma monitorização constante de hardware e de software. Os resultados dessa monitorização são informação essencial não só para qualquer administrador de sistemas ou prestador de serviços, mas também para o utilizador final, que desta forma passa a ter uma melhor perceção do estado dos serviços que utiliza.

Neste relatório, iremos abordar a implementação de raiz de um sistema de monitorização de domínio publico e *open source* na PTISP. Em particular, focamo-nos nas suas funcionalidades, vantagens e desvantagens. As principais vantagens deste sistema são: (1) os dados são guardados num modelo de dados multidimensional numa base de dados temporal (*time-series database*), uma vantagem comparativamente a outros sistemas de alarmística que não dispõem de qualquer base de dados; (2) possui sistema de alertas incorporado sem necessitar de outro software; (3) é facilmente escalável; (4) a coleta das métricas é feita por meio de um PULL em HTTP; (5) a implementação em sistemas de suporte gráfico é simples e é totalmente *open source* com uma boa aceitação pela comunidade. Em sentido inverso, as principais desvantagens prendem-se com (1) a dificuldade na configuração da "*High availability*" (HA); (2) bem como na exportação de dados históricos.

O trabalho aqui apresentado foi desenvolvido no âmbito da unidade curricular de Estágio do 2º ano de Mestrado em Engenharia Informática – Internet das Coisas, tendo como objetivo principal apresentar e descrever o trabalho realizado durante o estágio de 7 meses [de Outubro de 2019 a Junho de 2020] realizado na PTISP. A PTISP, Hosting Provider de referência 100% Português presente no mercado desde 2001 e sediada em Constância, está presente em três data centers, dois em Portugal e um em Espanha.

ABSTRACT

Over the last few years, monitoring and alarm systems have assumed an important role within organizations. They allow a system administrator to know, for example, the status of the computer equipment, enabling him/her to be aware of eventual interventions to be taken in the servers. Ensure a reliable, robust, high availability system, requires, however, constant monitoring of hardware and software. The results of this monitoring constitute vital information not only for any system administrator or service provider, but also for the end user, who gets a better understanding of the status of the services he/she is using.

In this report, we address the implementation from scratch of a public domain, open source monitoring system at PTISP. In particular, we focus on its features, advantages and disadvantages. The main advantages of this system are: (1) data is stored using a multidimensional data model, in a time series database, which presents advantages compared to other alarm systems, without any database; (2) it has a built-in alert system with no need for any other software; (3) it is easily scalable; (4) metrics are collected through PULL HTTP requests; (5) the implementation in graphic support systems is simple and fully open source, with wide community approval. Conversely, the main disadvantages are (1) the difficulty in "High Availability" (HA) configuration; (2) as well as in exporting historical data.

The present work was developed in the context of the Internship course of the 2nd year of the Master in Computer Engineering - Internet of Things, with the main goal of presenting and describing the work done during a 9-month internship [from October 2019 to June 2020] held at PTISP. PTISP is a reference, 100% Portuguese Hosting Provider, in the market since 2001. Headquartered in Constância, the company is present in three data centers, two in Portugal and one in Spain.

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

1. Introdução

O presente relatório tem como objetivo apresentar o projeto de implementação de sistemas de monitorização e alarmística desenvolvido na empresa PTISP durante o período de Estágio do Mestrado em Engenharia Informática – Internet das Coisas.

Este relatório descreve do ponto de vista técnico toda a implementação de um sistema de monitorização e alarmística, o que por si só não demonstra quaisquer avanços ou contributos científicos nas áreas em causa.

Neste capítulo é dada uma visão geral do estágio. Na Seção 1.1 é feito um enquadramento do estágio realizado. Na seção 1.2 é apresentada uma breve motivação acerca dos sistemas de monitorização e alarmística. Na seção 1.3 apresentamos os objetivos delineados para a realização deste estágio. Finalmente, na seção 1.4 procedemos a uma descrição da estrutura do relatório.

1.1.Enquadramento

O presente relatório resulta do estágio curricular realizado no âmbito do Mestrado em Engenharia Informática – Internet das Coisas do Instituto Politécnico de Tomar que decorreu no período compreendido entre Outubro de 2019 a Junho de 2020, na PTISP.

A PTISP é uma empresa presente no mercado, desde 2001, Hosting Provider de referência 100% Português, sediada em Constância e distribuída em três data centers, dois em Portugal um em Espanha. Durante o estágio assumi as funções de Administrador de Sistemas com a responsabilidade de implementar um sistema de monitorização de sistemas informáticos e alarmística.

1.2.Motivação

Nos dias de hoje, qualquer plataforma ou serviço necessita de sistemas informáticos.

Para garantir que estes funcionam corretamente é necessária uma constante monitorização. Ao longo dos tempos os sistemas de monitorização foram evoluindo. Inicialmente os dados eram controlados máquina a máquina, mas com a evolução dos tempos passaram a ser enviados para máquinas centralizadas e consultados a partir dessas máquinas.

Do funcionamento destes sistemas dependem muitas organizações, mas também inúmeras empresas. Qualquer pequena falha neste tipo de sistemas, poderá assim causar longas paragens e eventuais perturbações nos serviços aí disponibilizados, gerando elevados prejuízos. Evitar a ocorrência destes erros obriga a uma monitorização em tempo real. O objetivo consiste não só na eliminação de falhas, mas também na previsão e eventual antecipação, evitando dessa forma desfechos graves para as empresas e para os seus utilizadores. As áreas de sistemas e monitorização de sistemas, são assim áreas bastantes importantes e sujeitas a constante desenvolvimento. A realização do estágio aqui descrito, descreve a experiência decorrente da implementação e desenvolvimento de soluções de monitorização.

1.3. Objetivos

A realização do estágio teve com principal objetivo a integração no mundo do trabalho, de forma a pôr em prática alguns dos conhecimentos adquiridos durante o percurso académico.

Os objetivos delineados foram os seguintes:

(1) **Implementação de um sistema de monitorização** – configuração e instalação de um sistema que armazena e faz a recolha das métricas dos serviços de cada servidor monitorizado;

(2) **Desenvolvimento e implementação de scripts para recolha das métricas** – vista a expor os dados dos serviços dos servidores de maneira a que estes possam ser consultados pelo sistema de monitorização;

(3) **Implementação de um sistema de alarmística** – configuração e instalação de um sistema de tratamento de dados para fins de alarmística;

(4) **Implementação de um sistema de visualização gráfica dos dados recolhidos pela monitorização** – configuração e instalação de um sistema de visualização dos dados recolhidos pela plataforma de monitorização.

1.4.Estrutura do Relatório

O presente relatório é dividido em sete capítulos ao qual acresce uma seção dedicada a anexos. No capítulo 2 é apresentada uma breve descrição da empresa e dos principais serviços que a mesma presta aos clientes. No capítulo 3 apresentamos a implementação de um sistema de monitorização. No capítulo 4 é feita uma descrição do desenvolvimento e implementação dos scripts para a recolha das métricas dos serviços dos servidores, que serão consumidos pelo sistema de monitorização. No capítulo 5 detalhamos a implementação de um sistema de alarmística. No capítulo 6 apresentamos a implementação de um sistema de visualização gráfica dos dados recolhidos pela monitorização. Finalmente, no capítulo 7 concluímos este relatório.

2. Empresa

Neste capítulo é dada uma visão geral da PTISP empresa onde decorreu o estágio. Na seção 2.1 é feita uma breve apresentação da empresa. Na seção 2.2 apresentamos os principais serviços prestados pela empresa.

2.1.PTISP

A PTISP, empresa de Hosting Provider de referência 100% Português, com sede em Constância, encontra-se presente no mercado desde 2001.

Disponibiliza atualmente cerca de 35,000 serviços através de três data centers (dois em Portugal um em Espanha) e mais de 1500 servidores. Conta com mais de 60,000 domínios registados e 14,000 clientes ativos, entre empresas e particulares.

Integra uma equipa experiente e multidisciplinar, distribuída por três escritórios (Leiria, Lisboa e Constância) com a missão de ser o que o cliente precisa e quando precisa.

Foi eleita por três anos consecutivos, o Melhor Serviço de Alojamento pelos leitores da PCGuia, tendo obtido, pelo 7º ano consecutivo, o estatuto de Cliente aplauso do Millennium bcp. Adicionalmente conta também com os estatutos PME Excelência e PME líder.

2.2.Serviços prestados

Nas subsecções seguintes são apresentados os principais serviços prestados pela PTISP aos seus clientes.

2.2.1. Alojamentos

O serviço de alojamentos, é um serviço de contas de alojamento partilhado que disponibiliza um espaço privado num servidor físico. Este servidor encontra-se devidamente capacitado e configurado para alojar conteúdos (páginas, bases de dados, emails, etc.), 24 sobre 24 horas por dia, 365 dias por ano. A PTISP disponibiliza alojamento partilhado em Linux/PHP e Windows/ASP.NET.

2.2.2. Revendas

Os planos de revenda destinam-se a Agências Web, *Developers*, Freelancers ou outros que façam gestão de várias contas de alojamento e o queiram fazer de forma

centralizada e simples. Com um baixo custo permite alojamento de vários sites, numa infraestrutura segura devidamente monitorizada e com suporte 24x7x365. Os planos de revenda, além do painel de controle de gestão do revendedor, incluem ainda acessos diferenciados para as várias contas de alojamento, garantindo que cada utilizador terá o seu próprio acesso FTP e Painel de Controlo. Ser cliente de revenda é ainda sinónimo de descontos nos serviços associados aos alojamentos como domínios e certificados SSL.

2.2.3. Email

A PTISP disponibiliza diversas soluções para alojamento de email. Dependendo das exigências e funcionalidades, pode optar por um serviço de email tradicional com POP3, IMAP, SMTP e Webmail até serviços mais avançados e que incluem, para lá do email, ferramentas de produtividade e de colaboração.

2.2.4. Domínios

O recurso ao serviço de registo de domínios permite acesso a vários tipos de TLD (top-level domain), como gTLD (extensões genéricas como .COM), ccTLD (extensões de países como .PT) e ngTLD (extensões de nova geração como .shop).

2.2.5. Servidores Virtuais

A PTISP disponibiliza também duas tipologias de servidores virtuais, as tipologias são VPS e Cloud descritas em maior detalhe nas seguintes secções.

2.2.5.1. VPS

A VPS ou Servidor Privado Virtual é um serviço com recursos (RAM, CPU e HDD) dedicados e com o seu próprio sistema operativo. A VPS corre em servidores físicos de elevada capacidade e redundância que são depois virtualmente divididos em máquinas mais pequenas. Com recurso à virtualização e/ou a sistema de containers, cada VPS é totalmente isolada das demais, tendo o seu próprio acesso *root* e endereço IP.

2.2.5.2. Cloud

Os servidores cloud são servidores virtuais criados sobre uma infraestrutura de alta-disponibilidade. Estes planos incluem recursos dedicados e sistema operativo Linux ou Windows. Ajuste de recursos em função das necessidades, são ideais para alojamento de plataformas críticas e que necessitam de recursos dinâmicos. Configurações geridas e monitorizadas ou não geridas.

2.2.6. Servidores Dedicados

A PTISP disponibiliza também servidores dedicados baseados em hardware *enterprise*. Neste tipo de serviço, o cliente tem alocação total dos recursos e hardware do servidor, através do sistema operativo Linux, Windows, virtualização ou outro. Esta é uma solução indicada para sites ou plataformas que necessitem de muitos recursos e elevada performance. É fornecida com configurações geridas e monitorizadas ou não geridas.

3. Sistema de monitorização

Neste capítulo é dada uma visão geral do sistema de monitorização *Prometheus* e da sua implementação na PTISP, com vista à substituição do software *checkmk* anteriormente usado para a monitorização das máquinas. Assim na seção 3.1 é feita uma breve apresentação do sistema de monitorização *Prometheus*, da sua arquitetura baseada em (Brebner, 2019), das vantagens e desvantagens segundo (Ashok, 2020). Na seção 3.2 apresentamos e detalhamos a sua implementação.

3.1. Prometheus

O *Prometheus* (ver Figura 1) é um sistema de monitorização para serviços e aplicações. Faz, entre outras coisas, a recolha de métricas dos serviços dos servidores em determinados intervalos, avalia regras, mostra resultados e permite adicionar condições que fazem despoletar alertas.

Figura 1 – Logo *Prometheus*

Foi originalmente desenvolvido em GO (Golang) pela *SoundCloud*. Em 2016 tornou-se código aberto e foi aceite como projeto na *Cloud Computing Foundation* (CNCF[1]). É um sistema que tem um modelo de dados multidimensional em uma base de dados temporal (*time-series database*). Adicionalmente, possui uma linguagem própria (*PromQL*) para realização de *queries* aos dados armazenados. É totalmente autónomo a nível de armazenamento. A recolha de métricas é feita por meio de um PULL via HTTP. Por outro lado, a definição dos serviços a serem monitorizados pode ser feita através de um ficheiro de configuração ou de descoberta automática (*service discovery*) isto é, com recursos a uma zona DNS (*Domain Name System*). É uma ferramenta muito popular no processo de monitorização e alertas de aplicações e servidores, sendo usada, entre outros por empresas líderes de sistemas na *Cloud*, tais como *DigitalOcean*, *Red Hat*, *Suse* e *Weaveworks*.

[1] https://www.cncf.io/

3.1.1. Arquitetura

Os principais componentes do *Prometheus* (ver Figura 2) incluem o servidor *Prometheus* (que é o core, lida com o *service discovery*, recuperação e armazenamento de métricas, e análise dos dados da time-series com recurso à linguagem *PromQL*), um modelo de métricas e dados, um simples *built-in GUI* e tem suporte nativo para *Grafana*. Componentes adicionais do *prometheus*, incluem um gestor de alertas (no qual os alertas são definidos na linguagem de consulta *PromQL*) e um *gateway* de envio para monitorização de aplicações de curta duração (aplicações provisionadas em *Docker* ou *Kubernetes*, plataformas *open source* de desenvolvimento, provisionamento e execução de aplicações de forma simples, têm como objetivos principais a criação e gestão de ambientes isolados com recurso à tecnologia de containers).

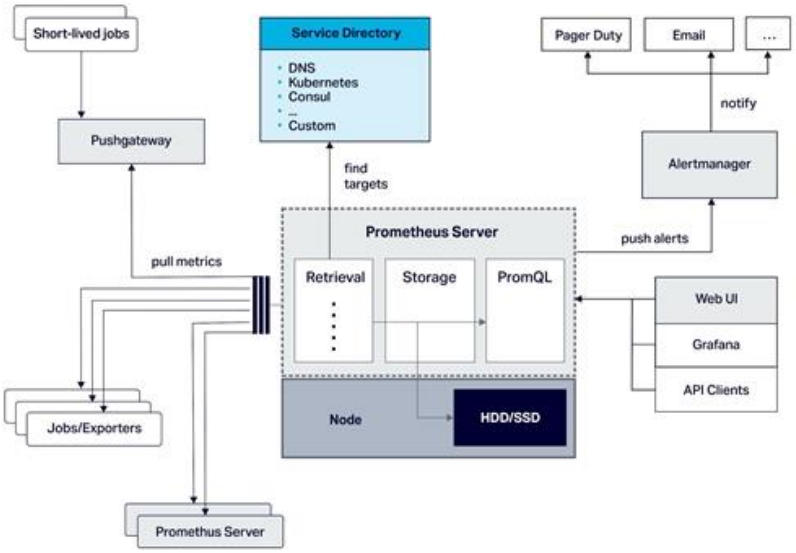

Figura 2 – Diagrama da arquitetura do *Prometheus*

Geralmente as ferramentas de monitorização recorrem a um dos três métodos abaixo indicados para fazer a recolha das métricas:

- ***Instrumentation***: adicionando código ao código fonte da aplicação que está a ser monitorizada;
- ***Agents***: criação de um agente que externamente à aplicação irá executar comandos e capturar métricas de forma automática para serem recolhidas;
- ***Espionage***: usa *traps* ou intercetores de rede para monitorizar pedidos ou fluxo de dados entre sistemas.

O *Prometheus* aconselha o uso de uma combinação de *Instrumentation* e *Agents* denominada "*exporters*". Para o *Prometheus* a linguagem de programação usada nos *exporters* é indiferente. O *prometheus* apenas guarda uma estrutura de dados que é consumida por http, o que permite que os *exporters* sejam feitos em qualquer linguagem, necessitando apenas serem seguidas as normas de construção das métricas.

Muitos dos *exporters* desenvolvidos pelas marcas que produzem software estão disponíveis para download, alguns implementados diretamente no código fonte das aplicações outros de forma isolada.

O *node_exporter* um dos *exporters* mais conhecidos do *Prometheus* é usado para recolher as métricas do hardware de sistemas *linux*, como por exemplo: uso de CPU, uso de RAM, uso do espaço em disco, métricas da *kernel*, etc…

3.1.2. Vantagens e Desvantagens

Neste capítulo apresentamos as vantagens e as desvantagens do uso do *Prometheus* face a outros serviços similares de monitorização de sistemas.

3.1.2.1. Vantagens

O *Prometheus* é um TSDB, *Time Series DataBase*, otimizado para guardar dados com o registo de data e hora. É baseado em PULL, uma diferença quando comparado com os outros sistemas de monitorização. Num sistema de monitorização comum a máquina envia as métricas para o servidor de monitorização. Ao invés, o *prometheus* vai buscar às máquinas as métricas via http. Ao ser o *prometheus* a requisitar os dados, as máquinas ficam livres dessa responsabilidade evitando assim a carga adicional decorrente da necessidade do envio das métricas para o servidor de monitorização.

Adicionalmente, o funcionamento do *prometheus* por via de um PULL permite um controlo centralizado de todas as máquinas a serem monitorizadas. Devido aos pedidos partirem todos do mesmo local, possibilita uma facilidade de instalação nas máquinas nas quais apenas é necessário proceder à instalação dos módulos de exposição de métricas.

O *prometheus* dispõe de um sistema de alertas incluído no seu "*core*", fazendo com que a necessidade de instalação de software adicional para gerar alertas não seja necessária. Os alertas são configurados em ficheiro por meio de condições lógicas.

A facilidade no *prometheus* em alterar as configurações passa pela alteração de apenas um ficheiro, enquanto num software como o *Nagios*, as alterações de configurações são efetuadas em diversos locais.

O *Service discovery* do *prometheus* pode ser efetuado por recurso a um ficheiro com a lista de todas as máquinas a serem monitorizadas e/ou com recurso a uma zona dns onde são verificadas as máquinas a serem monitorizadas, o que o torna bastante mais versátil quando comparado a um sistema de monitorização comum onde todas as máquinas teriam de ser adicionadas numa base de dados.

3.1.2.2. Desvantagens

No *prometheus*, a alta disponibilidade é algo difícil de implementar devido ao formato usado pelo *prometheus* para recolher métricas. A exportação de dados históricos de uma forma estruturada é complexa devido ao *prometheus* ser uma base de dados temporal não relacional. Usando o *prometheus* para fazer uma monitorização de vários data *centers* de forma diferenciada também se torna uma tarefa complexa. Para o conseguir fazer é necessária a instalação de um *prometheus* em cada data *center* e a instalação de um *prometheus* mestre para fazer a coleta dos dados de cada data *center*.

3.2.Implementação

Neste capítulo, iremos detalhar todos os passos decorridos na implementação e instalação do *prometheus* num servidor para produção com base em (Documentação Prometheus) e (Configure a Prometheus Monitoring Server with a Grafana Dashboard).

3.2.1. Instalação do servidor de Prometheus

Para a instalação do *prometheus*, um dos objetivos deste estágio, foi inicialmente provisionado um servidor dedicado em *data center* com a seguinte configuração: 8 cores, 32 gigas de memoria ram e 5tb de disco. O servidor dedicado é destinado ao *prometheus* e ao *alertmanager*. O *alertmanager* irá ser detalhado no Capítulo 5 quando o sistema de alarmística for descrito.

Após o servidor provisionado em *data center* procedeu-se à instalação do sistema operativo centos 7, foi feito o download dos ficheiros de instalação do *prometheus* do

repositório oficial no github[2] e por fim extraiu-se os ficheiros. De seguida foi necessário criar um utilizador no sistema operativo para o *prometheus* e a estrutura de pastas para onde os ficheiros irão ser copiados. Aquando das pastas criadas foram copiados os ficheiros para os seus devidos lugares com todas as permissões corrigidas.

Por fim, para o *prometheus* ser executado no background com todas as configurações efetuadas foi criado um serviço de *systemd* para o *prometheus*. Neste serviço definimos que a retenção de dados do *prometheus* é de 365 dias, o que significa que o *prometheus* vai manter os dados durante um ano antes de os começar a reescrever. Foi também definido onde se localizam todos os ficheiros do *prometeheus* e onde vão ficar guardados os dados recolhidos. Por último definimos a interface web como ativa.

Após as configurações acima e o arranque do serviço, o *prometheus* passa a estar disponível através da página web (ver Figura 3) no seguinte endereço http://IP:9090.

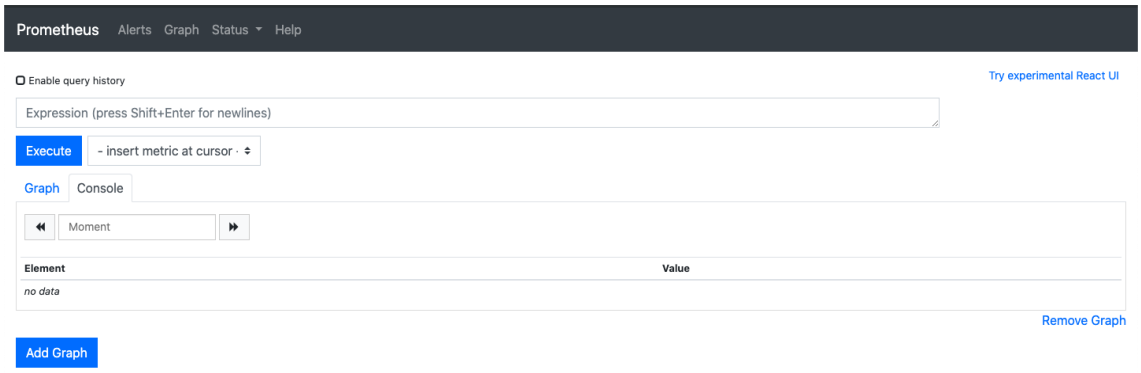

Figura 3 – Interface web *Prometheus*

Com a instalação concluída procedemos à configuração do sistema abaixo detalhada.

3.2.2. Configuração do servidor de Prometheus

Para configurar o *prometheus* basta ir ao ficheiro *prometheus.yml* e fazer as alterações de acordo com as necessidades. No anexo A poderemos ver o ficheiro de configuração criado no estágio. Neste ficheiro ficam registados os serviços que o *prometheus* vai

[2] https://github.com/prometheus/prometheus

monitorizar, os targets, isto é, os ficheiros ou a zona dns onde o *prometheus* vai consultar os ip's dos servidores aos quais tem de ir recolher as métricas, de quanto em quanto tempo o *prometheus* irá recolher as métricas e algumas configurações adicionais que possam vir a ser necessárias por serviço.

Começamos pela *tag global* que regista as gerais do *prometheus*. A **Erro! A origem da referência não foi encontrada.** mostra um exemplo de configuração:

Tabela 1 - Tag global do ficheiro de configuração do *prometheus*

```
global:
  scrape_interval:     60s
  evaluation_interval: 60s
  scrape_timeout: 30s
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - localhost:9093
```

O *scrape_interval* é a periodicidade com que o *prometheus* irá recolher as métricas dos servidores configurados. O *evaluation_interval* é a periodicidade com que o *prometheus* irá fazer a verificação das regras dos alertas e o *scrape_timeout* é o tempo que o *prometheus* tem para conseguir fazer a recolha das métricas. Caso demore mais do que 30 segundos, desiste e tenta numa próxima oportunidade.

O *alerting* é onde ficam configurados todos os sistemas de alertas/alarmística. Mais abaixo neste relatório iremos abordar o *AlertManager* que é referido como sistema de alertas no exemplo de configuração acima.

Por sua vez, a tag *scrape_configs* é a *tag* responsável por verificar que serviços estão configurados para recolha de métricas no *prometheus*. A estes *exporters* dá se o nome de *job_name* que irá ser, no fundo, o nome do serviço a ser recolhido.

Nesta configuração iremos configurar os seguintes serviços a serem monitorizados:

- *prometheus*: monitoriza o seu próprio funcionamento (auto-monitorização);

- *node*: diz respeito ao *node_exporter*. É o *exporter* responsável pelas métricas de hardware e *kernel* dos dispositivos *Linux*;
- *wmi*: diz respeito ao *wmi_exporter*. É o *exporter* responsável pelas métricas de *hardware*, *MSSQL*, *IIS* e *kernel* dos dispositivos *Windows*;
- *mysqld*: diz respeito ao *mysqld_exporter*. É o *exporter* responsável pelas métricas dos servidores de base de dados *MySQL*;
- *nginx*: diz respeito ao *nginx_exporter*. É o *exporter* responsável pelas métricas dos servidores web *NGINX*;
- *elasticsearch*: diz respeito ao *elasticsearch_exporter*. É o *exporter* responsável pelas métricas dos servidores de *ElasticSearch*;
- *blackbox*: diz respeito ao *blackbox_exporter*. É o *exporter* responsável pelas métricas vindas de verificações externas como por exemplo se um site está em baixo, controlar *pings*, verificar se o servidor de email está em baixo, etc…
- redis: diz respeito ao *redis_exporter*. É o *exporter* responsável pelas métricas do serviço de Redis;
- *postgres*: diz respeito ao *postgres_exporter*. É o *exporter* responsável pelas métricas dos serviços de base de dados *PostGreSQL*;
- *minio*: diz respeito ao *exporter* nativo do *MinIO* para *prometheus*. Responsável pelas métricas dos serviços *MinIO*;

3.2.3. Instalação dos exporters

Para a recolha das métricas ser efetuada, é necessário que em cada máquina a ser monitorizada exista um software para expor as mesmas. A esse software dá-se o nome de *exporters*. Os *exporters* consultam os dados de um determinado serviço. Após essa consulta formata-os e disponibiliza-os para serem consumidas pelo *prometheus* (por exemplo o *mysqld_exporter* expõe métricas da base de dados MYSQL).

A instalação destes *exporters* não é complexa, mas pode se tornar bastante trabalhosa e demorada dependendo do número de máquinas a serem instaladas, dado ser necessário aceder máquina a máquina e copiar um ou vários binários consoante os serviços a serem monitorizados e com estes, todos os ficheiros de configuração. Com vista a ultrapassar estas limitações decidimos implementar uma solução para facilitar este processo.

Com vista a esse objetivo, desenvolvemos um script que permite a instalação de forma simples dos *exporters* numa ou em várias máquinas (no Anexo B poderemos ver

o script de instalação), sendo apenas necessário mencionar o ip da máquina e o serviço a ser monitorizado. O script fica encarregue de copiar todos os ficheiros necessários para a instalação e configuração do *exporters*. A Seção 3.2.6 aborda em maior detalhe o script de instalação dos *exporters* nos servidores.

Recorremos também a outra solução para fazer a instalação dos *exporters*, que a PTISP já tinha implementada, essa solução chama-se *PUPPET*. O *PUPPET* é um serviço que permite automatizar a instalação e configuração de software em máquinas, categorizando-as com um nome, com esse nome podemos definir/criar manifestos para grupos de máquina. Os manifestos são ficheiros de configuração onde se define todo o software a ser instalado e as configurações a serem feitas.

3.2.4. Configuração dos exporters

A configuração dos *exporters* é necessária para os mesmos funcionarem corretamente. Esta configuração varia consoante o serviço que o *exporter* é responsável por consultar/monitorizar. De uma forma geral a configuração dos *exporters* tem como intuito definir que métricas são consultadas/expostas, a configuração de credenciais para o *exporter* aceder aos serviços que necessitem de credenciais, definir *timeouts*, entre outras configurações que sejam necessárias.

No caso em específico do *node_exporter* (*exporter* que expõe métricas do sistema) a configuração é mais complexa, devido a este *exporter* ter também a responsabilidade de recolher métricas de ficheiros, ficheiros estes que resultam de scripts customizados para serviços ou aplicações mais especificas. O Capítulo 4 aborda estes scripts em maior detalhe.

3.2.5. Implementação CI/CD e testes no GIT

Para garantir a integridade, rastreabilidade e testes de todas as configurações do *prometheus* e de todos os scripts, foi criado um repositório num *gitlab* interno da PTISP (no Anexo C poderemos ver a estrutura do repositório), onde foram implementadas soluções de CI/CD.

A cada alteração feita e adicionada ao repositório são corridos testes, testes estes que garantem que todas as alterações nas configurações ou ao código dos scripts são validas (no Anexo D poderemos ver o código de CI/CD implementado no estágio). Em caso negativo a alteração é revertida, é recebido um alerta e as alterações não entram em

produção. Em caso positivo a alteração é concretizada, é recebido um alerta de que todas as alterações são validas e é feita a implementação automática de todas as alterações no *PUPPET*. Em seguida o *PUPPET* encarrega-se de distribuir as configurações e os *exporters*, por todas as máquinas que estão no grupo que tem o *prometheus* como software a ser instalado.

Estas soluções de CI/CD foram fundamentais na tarefa de alteração de configurações e na garantia de que todas são validas e distribuídas de forma uniforme por todas as máquinas contempladas a recebê-la. De outra forma seria necessário efetuar alterações máquina a máquina, levando à geração de potenciais erros ou esquecimentos.

3.2.6. Script de instalação de exporters nos servidores

O *script* de instalação dos *exporters*, foi desenvolvido com a finalidade de permitir a instalação dos *exporters* nas máquinas a serem monitorizadas. Este script não menos é mais do que a execução de uma ordem sequencial de comandos. Assim inserindo qual o *exporter* e qual o ip da máquina, o script corre na máquina a sequência de comandos necessários à instalação o *exporter*.

Por exemplo, no caso do *exporter* do *mysql* o *script* copia todos os ficheiros necessários à instalação do *exporter* tal como o serviço para o executar. Depois do *exporter* instalado, é necessário criar na base de dados *mysql* um utilizador para que o *prometheus* possa obter métricas, a criação desse utilizador também é assegurada pelo script. É também ainda assegurado pelo script a configuração da firewall nas máquinas.

4. Scripts para recolha de métricas

Neste capítulo são abordados os scripts para recolha de métricas de serviços não padronizados. Os *scripts* para recolha de métricas de serviços não padronizados direciona-se para serviços que diferem de máquina para máquina. Estes scripts são bastante importantes porque são serviços críticos que de forma alguma podem ser padronizados. Ao invés terão de ser vistos caso a caso.

Para cada caso são instalados diferentes *softwares* e/ou diferentes configurações de hardware, como por exemplo monitorizar as controladoras de *RAID* ou os sistemas de backups. A existência de várias controladoras *RAID* de várias marcas ou vários tipos de sistemas de *backups*, torna o processo de monitorização com um único *exporter* difícil de conseguir. Para isso foram criados scripts que permitam de uma forma indireta contemplar todas essas soluções. Na seção 4.1 é feita uma breve descrição do seu desenvolvimento. Na seção 4.2 apresentamos a sua implementação.

4.1.Desenvolvimento

O desenvolvimento destes *scripts* foi motivado pela necessidade de recolher métricas de serviços ou configurações de hardware com características que não podem ser padronizadas. O *exporter* que aqui será retratado é o *node_exporter*, que é o *exporter* que devolve as métricas do sistema operativo e *hardware*.

Todas as máquinas têm necessariamente *hardware* e sistema operativo, o que fará com que por defeito todas terão este *exporter* instalado, mas nem todas por exemplo dispõem da mesma controladora *RAID*. Ao existirem configurações de *hardware/software* diferentes é muito difícil o *node_exporter* contemplar todas estas combinações para poder exportar as métricas.

Para resolver este problema optámos por criar um *script* principal (igual em todas as máquinas) que irá executar *scripts* responsáveis por exportar as métricas dos serviços customizados para ficheiros de texto, ficheiros de texto esses que posteriormente são processados pelo *node_exporter*. Com esta solução passa a ser possível, que consoante a configuração do *hardware/software* sejam executados os *scripts* correspondentes.

4.1.1. Script Principal (runner)

O *script* principal consiste num conjunto de scripts desenvolvidos em *python* e *bash* que são responsáveis por executar todos os scripts que processam as métricas dos serviços customizados (poderemos ver script *runner* no Anexo E-1). Estes *scripts* têm com objetivo executar em simultâneo todos os *scripts* customizados existentes numa pasta e escrever o *output* da execução de cada script numa outra pasta em ficheiros de texto separados.

Inicialmente era para ser criado apenas um *script* desenvolvido em *python*, mas ao aparecerem algumas dificuldades em paralelizar os processos em diferentes *threads* para que fossem todos executados em simultâneo, optámos por adotar dois *scripts*. O primeiro *script*, *bash* chamado de *runner* é desenvolvido em *bash* e fica encarregue de verificar quais os *scripts* existentes na pasta de *scripts* customizados e enviá-los para execução no *runner_proc* num processo individual. O *runner_proc* é o *script* criado em *python* que tem como função verificar se o *output* da execução é valido (poderemos ver script *runner* no Anexo E-2). Se o *output* não for válido este é rejeitado e não é escrito para o ficheiro de texto, caso o *output* seja válido este é escrito no ficheiro de texto. Ver abaixo Figura 4 – Diagrama do *script* principal.

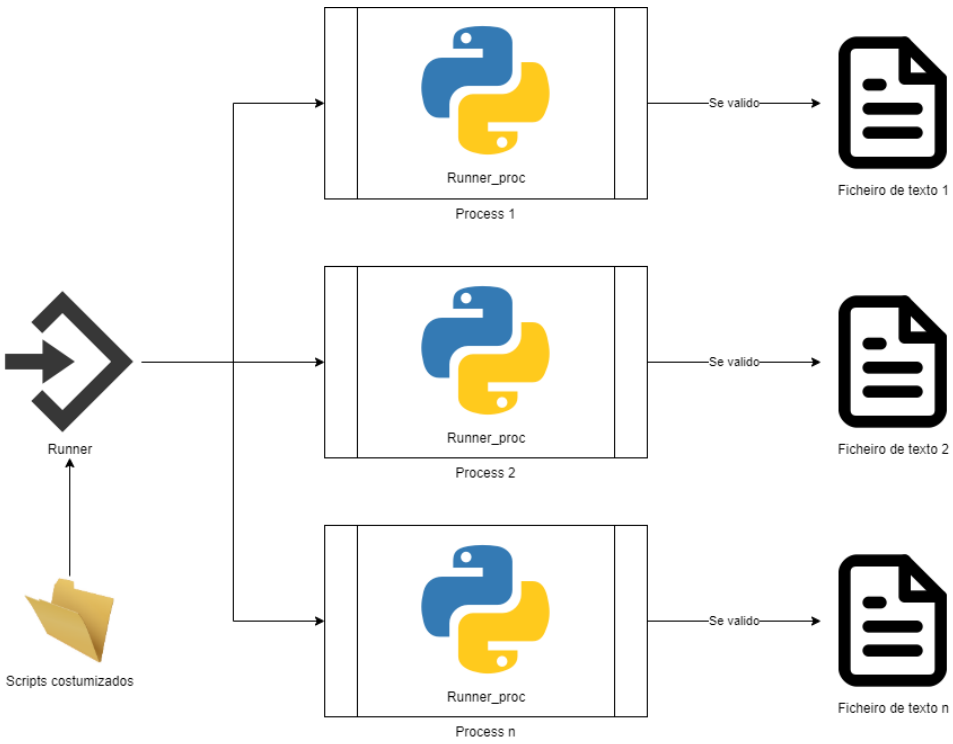

Figura 4 – Diagrama do *script* principal

Desta forma podemos adicionar em todas as máquinas todos os *scripts* customizados para todas as configurações de *hardware* e *software* utilizadas, e o *runner* tem a capacidade de apenas escrever o output para ficheiro caso este seja valido.

4.1.2. Scripts customizados

Os *scripts* customizados consistem de vários *scripts*. Para expor as métricas de *softwares*/serviços customizados, criámos vários *scripts* em *bash*, *php*, *perl* e *python*. Esses *scripts* têm como objetivo executar comandos específicos a cada serviço e formatar o *output* de forma a que o *node_exporter* o consiga entender e expor as métricas ao *prometheus* (no Anexo F poderemos ver o exemplo de um *script* customizado). Os *scripts* ao executarem os comandos a cada serviço tem de ser garantido que o *script* irá despoletar um erro caso o serviço não exista, erro esse que é essencial para o *runner_proc* entender que o serviço não existe e por isso não criar o ficheiro de texto.

Por exemplo no caso das controladoras *RAID* criámos um *script* para cada uma das marcas e modelos que irá expor as métricas do estado da controladora, do estado dos discos, da temperatura da controladora e quantos discos tem em funcionamento. Quando as máquinas têm painel de controlo criámos um *script* para cada um dos painéis de controlo que irá expor quantas contas existem criadas e quantas contas estão suspensas. Criámos também um *script*, no caso de a máquina ter serviço de *backups* instalado para cada uma das tipologias de *backups* que irá expor quantos *backups* estão em falta, quantos *backups* falharam, quantos *backups* foram bem-sucedidos e quantos *backups* estão em atraso.

Para todos os tipos de serviços não padronizados foram criados *scripts* para expor todas as métricas necessárias para a monitorização correta desses serviços.

4.2.Implementação

Neste capítulo iremos detalhar a implementação do script principal(*runner*) e dos *scripts* customizados num servidor a ser monitorizado.

4.2.1. Script Principal (runner)

A implementação do "*script* principal" é uma tarefa relativamente simples e distribui-se em três pontos. Instalação do *node_exporter*, cópia dos *scripts* para uma pasta existente

em sistema e a criação de um "*cronjob*" (tarefa executada automaticamente de x em x tempo) para executar o *script*. De seguida iremos explicar mais detalhadamente todo o processo de implementação.

Para instalação do *node_exporter* no servidor a ser monitorizado, foi feito o *download* do *node_exporter*, copiou-se para a pasta onde o *node_exporter* irá ficar e foi criado o serviço que o vai deixar a executar em *backgroud* no sistema. Nesse serviço foi configurado o caminho da pasta onde irão se encontrar os ficheiros de texto dos serviços customizados que ele irá ter de consultar para expor ao *prometheus*.

Após a instalação do *node_exporter* para dentro da sua pasta, foram copiados os ficheiros do "*script* principal" (*runner* e *runner_proc*) e foram criadas duas pastas. Uma de nome *scripts*, pasta onde se irão encontrar todos os scripts customizados e outra de nome *text*, pasta onde irão ficar todos os ficheiros de texto com o *output* valido dos *scripts* customizados. Para todo este processo funcionar fica a faltar a execução do "*script* principal", que foi por opção fazê-la criando um "*cronjob*", "*cronjob*" esse que foi configurado para executar o *runner* de 30 em 30 segundos. Toda esta implementação, posteriormente foi configurada num manifesto para ser distribuída pelo *PUPPET* por todas as máquinas, passando assim a ser um processo automático.

4.2.2. Scripts customizados

A implementação dos *scripts* customizados é uma tarefa ainda mais simples que a do *script* principal. Neste caso apenas é necessário copiar os ficheiros para dentro de uma pasta, pasta essa que depois é definida no "*script* principal". Mas mais uma vez nesta implementação, foi posteriormente também configurado um manifesto para a distribuição pelas máquinas ser feita pelo *PUPPET*.

5. Sistema de alarmística

Neste capítulo é dada uma visão geral do sistema de alarmística *AlertManager* e retratada a implementação com o *Prometheus*.

Os sistemas de monitorização abordados no capítulo acima ajudam a prever, descobrir ou a dar detalhes sobre o potencial problema nos sistemas informáticos através da recolha de métricas, já os sistemas de alarmística ou de alertas, tendo como base as métricas identificam o problema alertando o utilizador, e permitindo que este identifique o problema e o resolva no momento. Na secção 5.1 fazemos a apresentação do serviço utilizado para gestão de alertas e envio de notificações e também da sua arquitetura baseado em (Chiboub, 2020). Na secção 5.2 detalhamos a sua implementação com base em (Mitesh, 2018).

5.1. AlertManager

O *AlertManager* é um serviço que gere os eventos enviados pelo servidor de *Prometheus*, enviando-os assim para o utilizador. O *Prometheus* mesmo sendo uma excelente ferramenta para monitorização de eventos não tem a capacidade de enviar um alerta quando algo inesperado acontece, para resolver essa lacuna foi criado o *AlertManager*, que ao receber os eventos do *Prometheus* permite fazer uma gestão dos mesmos e enviá-los em formato de notificação/alerta. Estes alertas podem ser enviados por meio de um e-mail, *slack* (Marshall, 2017), sms ou qualquer outro serviço de mensagens.

O *AlertManager* foi construído como uma ferramenta separada o que faz bastante sentido quando examinamos o funcionamento do *Prometheus* de perto. O *Prometheus* faz a monitorização e recolha das métricas em tempo real, se apenas fosse usado o *Prometheus* iriamos ficar sobrecarregados com a quantidade de dados que ele fornece, com o *AlertManager* os dados transformam se em alertas que apenas serão enviados quando algo importante acontece. Uma vez que é o utilizador que decide o que é considerado “importante” não irá ter de lidar com toda imensidade de dados fornecida pelo *Prometheus*.

Uma das grandes funcionalidades do *AlertManager* é que o mesmo pode processar eventos e notificar criando alertas de uma forma que faça sentido. Por exemplo, se um

evento acionar 100 alertas no *Prometheus*, o *AlertManager* irá identificar que eles são essencialmente o mesmo evento e irá agrupa los em uma única notificação/alerta, em vez de enviar 100 mensagens.

O *AlertManager* age como um segurança num bar. Ele decide quem entra e quem não pode entrar. A ferramenta usa três recursos essenciais para o fazer:

- *Grouping* – Recebe eventos semelhantes e agrupa-os em uma única mensagem para que o utilizador não seja sobrecarregado. É como se fosse o segurança do bar a dizer: "OK, todos podem entrar, mas apenas uma pessoa se pode deslocar até ao bar para fazer os pedidos".
- *Inhibition* – Suprime os alertas de baixo nível quando já estiverem a ser enviados alertas de eventos mais críticos. É como se fosse o segurança do bar a dizer: "Desculpe, mas temos uma festa privada para clientes VIP esta noite. Volte num outro dia se não for cliente VIP, se for cliente VIP pode entrar."
- *Silences* – Impede alerta de ser enviado por um determinado tempo. É basicamente o segurança a dizer: "Estamos fechados durante uma hora, volte depois."

5.1.1. Arquitetura

Na figura abaixo (Figura 5) podemos ver o quanto a arquitetura do *AlertManager* com o *Prometheus* é básica.

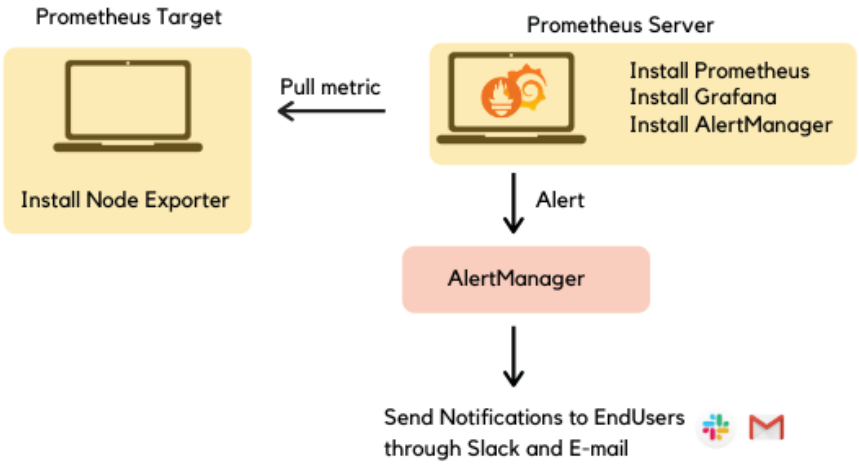

Figura 5 - Arquitetura do *AlertManager* com *Prometheus*

As regras dos alertas são definidas no *Prometheus*. O *Prometheus* ao fazer PULL das métricas dos clientes recebidas de um *exporter*, analisa-as e se alguma regra de alerta for acionada esta é enviada para o *AlertManager*.

Quando recebida pelo *AlertManager* este alerta passa pelo processo de seleção e é enviado por meio uma notificação para canal de mensagens selecionado consoante as regras definidas.

5.2. Implementação

Neste capítulo iremos abordar e demonstrar todos os passos decorridos para a instalação e configuração do *AlertManager* no servidor de *Prometheus* num ambiente de produção.

5.2.1. Instalação do AlertManger

Na instalação do *AlertManger*, foi utilizado o servidor dedicado em *data center* onde tinha já sido anteriormente instalado o *Prometheus*.

Foi efetuado o download dos ficheiros de instalação do *AlertManager* do repositório oficial no github[3] e extraíram-se os ficheiros. De seguida foi necessário criar um utilizador no sistema operativo para o *AlertManager* e a estrutura de pastas para onde os ficheiros irão ser copiados. Aquando das pastas criadas foram copiados os ficheiros para os seus devidos lugares com todas as permissões corrigidas.

Por fim, para o *AlertManger* ser executado no background com todas as configurações efetuadas foi criado um serviço de *systemd* para o *AlertManger*. Neste serviço definimos qual o caminho do ficheiro de configuração, que é o ficheiro de nome "*alertmanager.yml*" onde ficam armazenadas todas as indicações para configuração. Por último definimos a interface web como ativa e definimos qual e o ip e porta a serem utilizados.

Após as configurações acima e o arranque do serviço, o *AlertManager* passa a estar disponível através da página web (ver Figura 6) no seguinte endereço http://IP:9093.

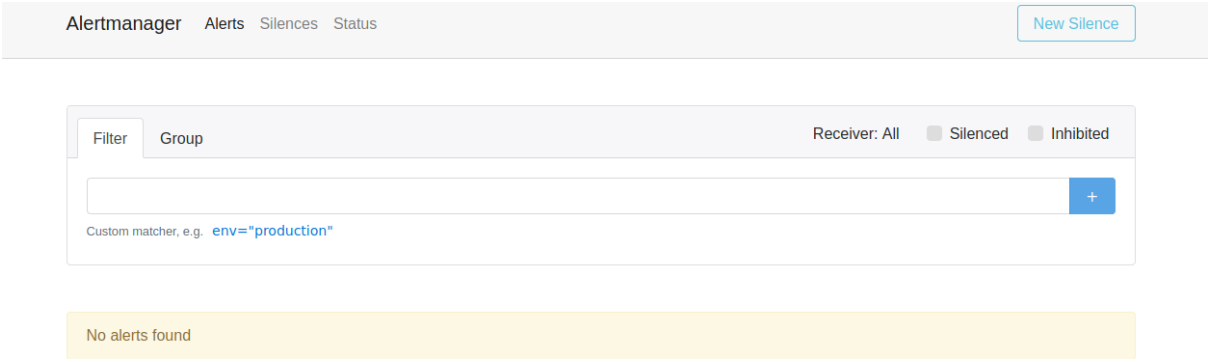

Figura 6 - Página web *AlertManger*

Após a instalação concluída procedemos à configuração do *AlertManager* abaixo detalhada.

[3] https://github.com/prometheus/alertmanager

5.2.2. Configuração do AlertManager

A configuração do *AlertManger* passa por editar o ficheiro *alertmanager.yml* e fazer as alterações de acordo com as necessidades. No anexo G encontra-se o ficheiro de configuração do *AlertManager* criado no estágio. Neste ficheiro ficam registadas todas as configurações relativas a tempos de espera, plataformas para onde irão ser enviados os alertas a fim de gerar a notificação, espaço temporal entre notificações e regras de seleção e de agrupamento de alertas. Abaixo, na Tabela 2, é mostrada uma configuração por defeito do *AlertManager* e uma breve descrição da mesma e todas as variáveis usando (Documentação AlertManager).

Tabela 2 – Exemplo de ficheiro de configuração do *AlertManager*

```
route:
  group_by: ['alertname', 'cluster', 'service']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 3h
  receiver: default
  routes:
  - match_re:
      service: ^(foo1|foo2|baz)$
    receiver: example1
  routes:
    - match:
        severity: critical
      receiver: example
inhibit_rules:
- source_match:
    severity: 'critical'
  target_match:
    severity: 'warning'
  equal: ['alertname', 'cluster', 'service']
receivers:
- name: default
- name: 'example'
```

```
- name: 'example1'
```

Dentro do campo *route* iremos encontrar o *group_by* que é o parâmetro responsável por agrupar todos os alertas, no caso da configuração acima irão ser agrupados por nome, cluster e serviço com isto todos os alertas que preencherem os requisitos acima descritos irão ser agrupados e enviados numa só notificação.

O *group_wait* define quando irão surgir novos alertas agrupados no exemplo definido a cada 30s, desta forma é garantido que múltiplos alertas que estão a ser despoletados são todos agrupados na primeira notificação a sair.

O *group_interval* é responsável por definir quanto tempo demora a sair uma segunda notificação dos mesmos alertas agrupados inicialmente, no caso do *repeat_interval* se as notificações foram enviadas com sucesso este define o tempo necessário para sair uma segunda remessa das mesmas notificações para o caso de os alertas não terem sido resolvidos ou vistos e por último todos os alertas serão enviados para o canal definido no *receiver*.

Temos ainda também disponível uma forma de criar rotas secundarias que preencham certos requisitos, tais como por exemplo, igualdade de um "*label*", uma expressão "*regex*" ou qualquer outro tipo de regra que possa ser definida neste âmbito. No exemplo acima temos mencionado o *match_re* que é uma regra por "*regex*" e temos o *match* que irá ser uma regra simples, podendo considerá-la como algo igual a, caso o "*label*" do alerta preencha os requisitos da regra o alerta será enviado pelo canal definido no *receiver* correspondente à regra no *match_re* é o *receiver* "*example1*" e no *match* é o *receiver* "*example*".

Dentro de *inhibit_rules* temos a possibilidade de silenciar as notificações vindas de alertas por regra caso o alerta seja o mesmo, mas apenas com a diferença de ter mais gravidade como podemos ver no exemplo em *source_match* e *target_match*, se o alerta que disparou e que tem a notificação pendente estiver em *warning* e no mesmo momento dispare o mesmo alerta mas em "*critical*" o *AlertManager* em vez de enviar duas notificações irá inibir a notificação de "*warning*" e apenas irá enviar a de "*critical*", visto que a informação contida em ambos os alertas iria ser a mesma e apenas iria diferir a gravidade.

Por último definimos em *receivers* o nome do canal para onde serão enviadas as notificações configuradas acima e toda a configuração relativa a este.

5.2.3. Definição de regras de alertas

As regras de alertas como já descrito acima são definidas no *Prometheus* para depois serem processadas pelo *AlertManager* e serem enviadas as notificações.

Abaixo na Tabela 3 iremos dar um exemplo de como são definidas estas regras que despoletam os alertas.

Tabela 3 – Exemplo de uma regra para um alerta no *Prometheus*

```
groups:
- name: example
  rules:
  - alert: HighRequestLatency
    expr: request_latency_seconds:mean5m{job="myjob"} > 0.5
    for: 10m
    labels:
      severity: critical
    annotations:
      summary: High request latency
```

A *tag* "*name*" define o nome do grupo de regras, abaixo temos a *tag* "*rules*" que é onde irá ser definidas as regras para os alertas. Neste caso temos o alerta de nome "*HighRequestLatency*", este alerta pelo que esta definido na *tag* "*expr*" significa que quando a média de 5 minutos da latência dos pedidos em segundos for maior que 0.5 durante 10 minutos este alerta irá disparar, e porque durante 10 minutos? porque na *tag* "*for*" está definido 10m, nesta *tag* é definido durante quanto tempo a regra tem de ser positiva antes de gerar o alerta.

As *tags* "*labels*" e "*annotations*" é a informação que o alerta irá levar para quando for processado pelo *AlertManger*.

Com base na estrutura da Tabela 3 podemos ter vários grupos de alertas com nomes diferentes para diferenciar por exemplo categorias de alertas e em cada grupo podemos ter várias regras para gerar os alertas.

6. Sistema de visualização gráfica dos dados recolhidos

Neste último capítulo é mostrado todo o processo de configuração dos sistemas de visualização gráfica dos dados/métricas.

Os sistemas de visualização gráfica dos dados são sistemas que permitem visualizar e analisar os dados recolhidos utilizando gráficos. Nestes sistemas para alem de se poder criar gráficos é ainda possível criar *dashboards* dinâmicos. Para este projeto o sistema de visualização de dados recolhidos foi o *Grafana*.

No capítulo 6.1 apresentamos o *Grafana*, os seus componentes e as suas funcionalidades com base em (HSU, 2019) e (McGinn, 2019), no capítulo 6.2 abordamos a sua implementação no âmbito do projeto desenvolvido no estágio.

6.1. Grafana

O *Grafana* (ver Figura 7) é uma plataforma web *open source* que permite visualizar e analisar métricas por meio de *dashboards* dinâmicos compostos por gráficos.

Figura 7 – Logo *Grafana*

6.1.1. O que é o Grafana e para que é usado?

O *Grafana* tem suporte para diversos tipos de bases de dados como por exemplo, *Graphite*, *Prometheus*, *InfluxDB*, *ElasticSearch*, *MySQL*, *PostgreSQL* etc.

O *Grafana* sendo uma solução *open source* também nos permite desenvolver plug-ins do zero para integração com as várias bases de dados diferentes, que podem ser usadas pela ferramenta, esta ferramenta também nos ajuda a monitorizar e analisar dados ao longo de um período de tempo selecionável, a este processo dá-se o nome de analise de serie temporal. O *Grafana* ajuda-nos a analisar o comportamento de um sistema ou de uma aplicação num período temporal ou em tempo real.

Uma das grandes vantagens do *Grafana* é que pode ser *self-hosted*, isto significa que caso não queiramos que os nossos dados sejam armazenados numa plataforma

externa/online, não o precisamos de fazer, podemos simplesmente instalá-la num servidor dentro da nossa infraestrutura o que garante a maior segurança dos dados.

Com o tempo, esta plataforma ganhou muita popularidade na indústria e foi implementada por grandes empresas tais como *PayPal*, *eBay*, *Intel* entre outras.

6.1.2. Dashboards do Grafana

Na Figura 8 podemos ver um *dashboard* do *node_exporter* no *Grafana*, neste *dashboard* podemos verificar vários painéis entre os quais mostram a utilização de cpu, a utilização de memoria ram, a utilização de rede e a utilização de disco (no Anexo H podem ser encontrados alguns *dashboards* criados durante o estágio).

Figura 8 -Exemplo de um *dashboard* do *Grafana* do *node_exporter*

Os *dashboards* contêm vários painéis individuais que são organizados em grelha, cada painel possui funcionalidades distintas.

Os *dashboards* podem ser criados pelo utilizador ou podem ser também descarregados no repositório oficial de *dashboards* do *Grafana*, este repositório tem vários *dashboards* oficiais, ou seja, criados pelos criadores do *grafana* como também tem vários *dashboards* disponibilizados pela comunidade.

Os painéis extraem dados das bases de dados configuradas por meio de *queries* e contem uma vasta gama de opções de visualização de dados, como mapas geográficos, *heatmaps*, histogramas, uma grande variedade de tabelas e gráficos de linhas e barras.

Na Figura 9 podemos ver um *dashboard* demo apenas.

Figura 9 – *Dashboard* demo

No ponto 1 temos um seletor que nos permite selecionar e navegar num espaço temporal definido com a seleção de pontos no gráfico.

No ponto 2 temos já uma vasta gama de intervalos de tempo com por exemplo os últimos 5 minutos, as últimas 3 horas, as últimas 6 horas e outros, que poderemos selecionar para facilitar os filtros temporais.

No ponto 3 temos o botão que nos permite atualizar os dados que estamos a visualizar e permite também definir um intervalo de tempo para que o *dashboard* atualize os dados de forma automática.

No ponto 4 temos o gráfico que neste caso é um gráfico de linhas, com o tempo no eixo dos X e as métricas no eixo dos Y.

No ponto 5 temos as legendas dos dados selecionados para aparecer no gráfico, ao clicar nestas legendas podemos mostrar ou esconder no gráfico os dados.

6.2. Implementação

Nesta secção iremos abordar e demonstrar todos os passos decorridos para a instalação e configuração do *Grafana* e dos vários *dashboards*, com base em (Grafana Support for Prometheus).

6.2.1. Instalação e configuração Grafana

A instalação do *Grafana* é um processo muito simples, a instalação é feita por um instalador que facilita todo o processo e configura tudo a nível de sistema operativo. Para proceder basta aceder ao site do *Grafana*[4] e seguir os passos para proceder à instalação.

No fim da instalação do *Grafana* e de estar a funcionar corretamente ao aceder ao link http://IP:3000/ irá existir uma página como a da Figura 10. Em seguida é necessário proceder à configuração da "*data source*", isto é, o sítio que irá ser a origem dos dados que neste caso irá ser o *Prometheus*, que é uma das "*data sources*" disponíveis de entre varias lá indicadas.

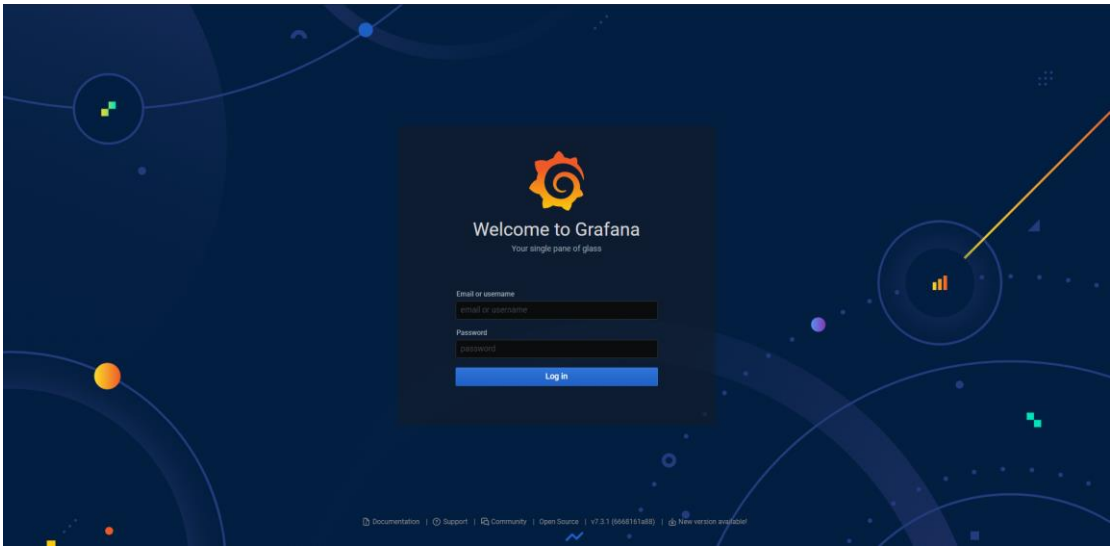

Figura 10 – Página de login do Grafana

[4] https://grafana.com/docs/grafana/latest/installation/

Para adicionar a "*data source*" basta fazer *login* no *Grafana* como administrador, ir a configurações e carregar em "*add data source*" e selecionar o *Prometheus* como se pode observar na Figura 11:

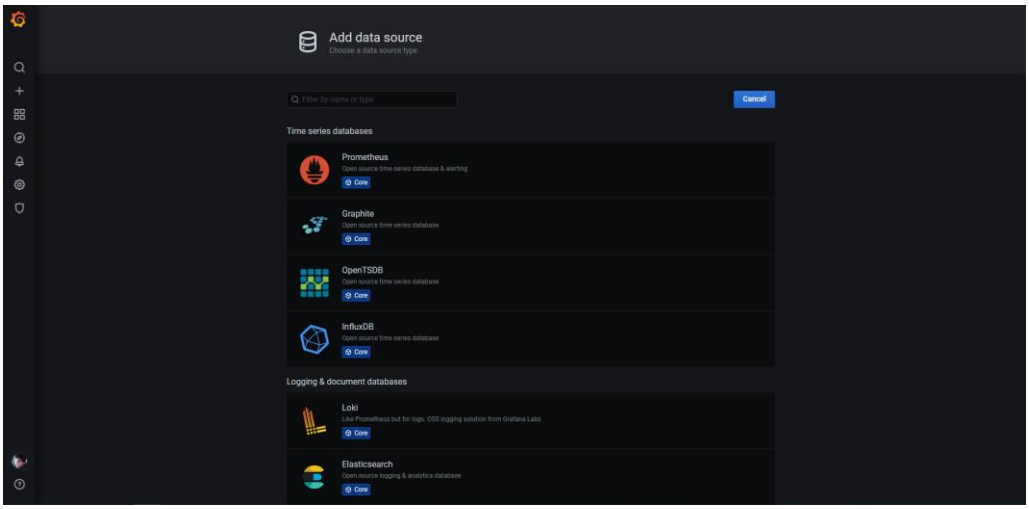

Figura 11 – Adicionar *data source Grafana*

Na página que irá aparecer no fim de carregar em *Prometheus* basta preencher o campo do URL com o url do *Prometheus* e carregar em "*Save & Test*" após este procedimento o *Grafana* já consegue ir consultar todos os dados que estão guardados e disponíveis no *Prometheus*.

O *Grafana* além de permitir criar *dashboards* como iremos ver no capítulo seguinte, permite também fazer *queries* diretamente ao *Prometheus* com uma ferramenta chamada de Explore que ajuda a fazer as *queries* completando o que esta a ser inserido como poderemos ver na Figura 12.

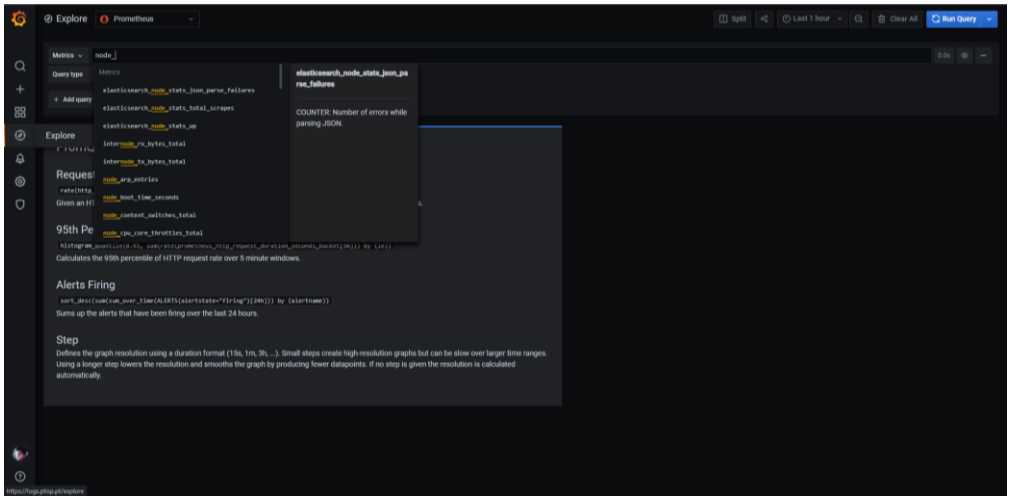

Figura 12 – Ferramenta *Explore* no *Grafana*

6.2.2. Criação/importação de dashboards

A criação ou importação de um *dashboard* no *Grafana* é um processo que se encontra bastante facilitado porque o *Grafana* nos oferece toda as ferramentas e ajudas necessárias no processo.

Começando pela importação, para a efetuar basta carregar no ícone + que pode ser visto na Figura 11 e Figura 12 e lá existe uma opção que é "*import*". Ao selecionar esta opção é requisitado um url ou id para importar via grafana.com ou um *dashboard* em formato JSON como podemos ver na Figura 13. O ID ou o JSON são adquiridos no site do *Grafana* na secção dos *dashboards* da comunidade[5].

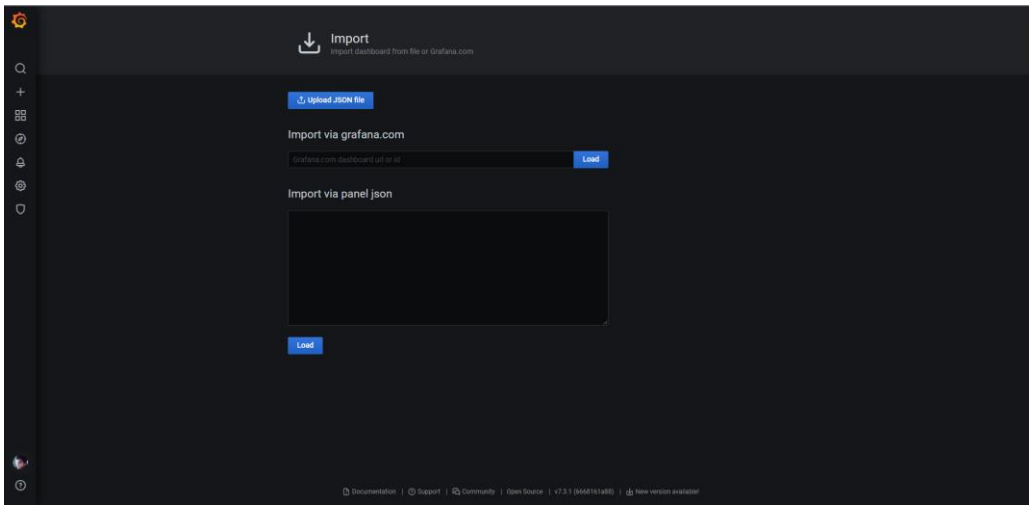

Figura 13 – Importar *dashboard* no *Grafana*

A opção ID é para copiar o *dashboard* exatamente como ele existe na plataforma dos *dashboards* do *Grafana*, ao inserir o ID e clicar em "*load*" ele importa o *dashboard* e o mesmo fica logo funcional e permite fazer todas as alterações necessárias dependendo do que se pretende.

A opção *JSON* é para o caso de querermos alterar o *dashboard* antes de o adicionar à plataforma, desta forma podemos fazer todas as alterações necessárias ao *JSON* e só depois importá-lo, a duplicação de *dashboards* é feita desta forma, exportamos o *JSON* e depois importamos de novo com outro nome ou outra configuração.

Na criação é um processo um pouco mais trabalhoso, mas fácil porque na ferramenta existem todos os tipos de gráficos basta escolher o mais conveniente para o tipo de

[5] https://grafana.com/grafana/dashboards

dados e populá-lo com dados. Passo a explicar brevemente a criação de um *dashboard* com um painel com um gráfico.

Novamente no *Grafana* carregar no ícone + e escolher a opção *dashboard* e iremos ficar com a seguinte pagino como mostrado na Figura 14.

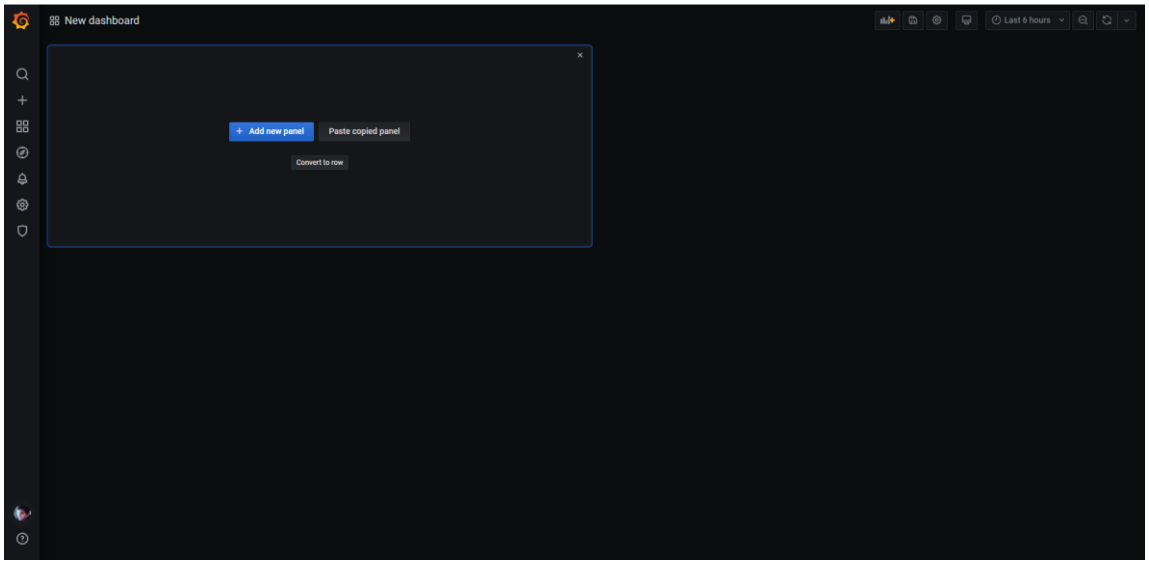

Figura 14 – Novo dashboard no Grafana

Clicamos em "*Add new panel*" e irá aparecer uma página para criação do painel, nesta página escolhemos a "*data source*" que será selecionado o *Prometheus*, e depois basta configurar uma *query* e o gráfico é desenhado com base nos dados retornados dessa *query*. Na Figura 15 abaixo foi feita uma *query* que retorna a temperatura do cpu de uma máquina.

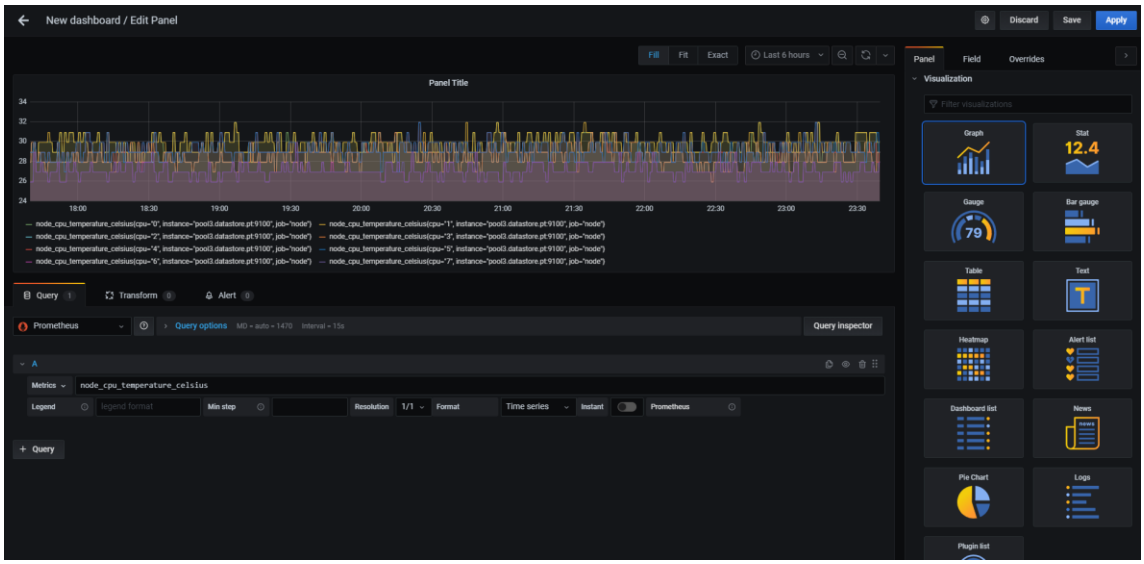

Figura 15 – Criação de um painel de um *dashboard* no *Grafana*

Apos inserir a *query* o gráfico é logo preenchido com os dados quem são retornados da *query* de uma forma automática. Na Figura 15 podemos também verificar que ao lado esquerdo da imagem existe uma aba chamada de "*Panel*" que permite editar o tipo do gráfico, é também nessa aba que ao navegar para cima e para baixo podemos alterar o valor dos eixos de x e y, alterar a forma como os dados são apresentados, alterar as legendas, no fundo alterar tudo o que diz respeito a este painel e aos dados nele contidos.

Ao carregar em "*Apply*" no canto superior esquerdo, o painel irá ser gravado e passará a fazer parte do *dashboard* inicialmente criado. Apos isto basta criar vários painéis com a informação de vários componentes do sistema operativo e hardware e irá ficar um *dashboard* como o mostrado na Figura 16 abaixo.

Figura 16 – Dashboard do Grafana com vários painéis

Na Figura 16 podemos ver um *dashboard* que já tem vários painéis com vários tipos de elementos 3 gráficos e um elemento de texto. No elemento de texto temos o número de cores de cpu da máquina, no segundo painel vemos a utilização de cpu, no terceiro painel vemos a utilização da memoria ram da máquina e no último painel vemos o *load* da máquina.

Tudo o que foi descrito acima pode agora ser aplicado a todos os serviços ou métricas existentes no *Prometheus*, com isto poderemos ter vários gráficos em vários *dashboards* sobre tudo o que necessitamos.

7. Conclusão

Este documento resumiu todo o desenvolvimento, pesquisa e implementação realizada durante os 7 meses de estágio na PTISP, apresentando todo o projeto de implementação feita a nível de sistemas de monitorização e alarmística em que assumi algumas das responsabilidades na conceção e implementação.

Em concreto, começamos por explicar como funciona a PTISP e em que áreas opera e o porquê de ser tão critica a utilização destes sistemas.

De seguida, foram apresentados os conceitos, técnicas, e tecnologias utilizadas no projeto descrito neste documento, nomeadamente: o *Prometheus* como sistema de monitorização e todos os *exporters* para armazenar e recolher todas as métricas dos sistemas informáticos; os scripts para recolha de métricas de serviços não padronizados e todo o desenvolvimento necessário; o *AlertManager* como sistema de alarmística para envio de alertas e notificações para os utilizadores; e por último o *Grafana* como sistema de visualização de métricas e dados recolhidos pelo sistema de monitorização.

Foi explicado com algum detalhe toda a implementação dos mecanismos descritos acima. A implementação destes mecanismos para monitorização de sistemas informáticos traz muitos benefícios, desde redução de erros humanos, aumento da capacidade de resposta a eventos problemáticos a nível de software e hardware e uma melhor perceção da PTISP face ao estado dos serviços que disponibiliza aos clientes. De todos os benefícios o que considero mais importante é um aumento da satisfação do utilizador final ficando com um sistema mais robusto e de alta disponibilidade.

Todo o software utilizado neste projeto, exceto o desenvolvido, é de código aberto e disponível na internet para acesso de todos. Isto permite uma redução nos custos de um sistema de monitorização e uma maior versatilidade para eventuais customizações.

O estágio proporcionou-me uma visão prática que as aulas não conseguem abranger, sobre como o dia-a-dia no mundo do trabalho pode ser complexo e desafiador. Alem de que todos os desafios que tive de enfrentar no desenvolvimento deste projeto me irão ajudar bastante, deram me uma visão essencial e competências a ter no mundo do trabalho.

Bibliografia

Ashok, Arunlal. 2020. Advantages of Prometheus monitoring tool. *CRYBIT*. [Online] 22 de Março de 2020. https://www.crybit.com/advantages-of-prometheus/.

Brebner, Paul. 2019. Como usar a solução de código aberto Prometheus para monitorar aplicações em escala. *infoq.* [Online] 16 de Agosto de 2019. https://www.infoq.com/br/articles/prometheus-monitor-applications-at-scale/.

Chiboub, Sylia. 2020. Prometheus Alerting with AlertManager. *medium.* [Online] 26 de Maio de 2020. https://medium.com/devops-dudes/prometheus-alerting-with-alertmanager-e1bbba8e6a8e.

Configure a Prometheus Monitoring Server with a Grafana Dashboard. *scaleway.* [Online] https://www.scaleway.com/en/docs/configure-prometheus-monitoring-with-grafana/.

Documentação AlertManager. *Prometheus.* [Online] https://prometheus.io/docs/alerting/latest/alertmanager/.

Documentação Prometheus. *Overview Prometheus.* [Online] https://prometheus.io/docs/introduction/overview/.

GRAFANA SUPPORT FOR PROMETHEUS. *Prometheus.* [Online] https://prometheus.io/docs/visualization/grafana/.

HSU, EJ. 2019. Build A Monitoring Dashboard by Prometheus + Grafana. *medium.* [Online] 7 de Janeiro de 2019. https://medium.com/htc-research-engineering-blog/build-a-monitoring-dashboard-by-prometheus-grafana-741a7d949ec2.

Marshall, Ed. 2017. Slack notifications & Alertmanager. *infinityworks.* [Online] 5 de Junho de 2017. https://infinityworks.com/insights/slack-prometheus-alertmanager/.

McGinn, Matthew. 2019. Monitoring the edge with Prometheus pt. 1. *balena.* [Online] 16 de Maio de 2019. https://www.balena.io/blog/monitoring-the-edge-with-prometheus-pt-1/.

Mitesh. 2018. Prometheus With AlertManager. *itnext.* [Online] 19 de Novembro de 2018. https://itnext.io/prometheus-with-alertmanager-f2a1f7efabd6.

Schkn. Monitoring Linux Processes using Prometheus and Grafana. *devconnected.* [Online] https://devconnected.com/monitoring-linux-processes-using-prometheus-and-grafana/.

Turnbull, James. 2018. *Monitoring With Prometheus.* 2018.

Anexos e Apêndices

Anexo A – Ficheiro de Configuração do Prometheus

Tabela 4 – Anexo A – Ficheiro de configuração prometheus.yml

```
global:
  scrape_interval:     60s
  evaluation_interval: 60s
  scrape_timeout: 30s

alerting:
  alertmanagers:
  - static_configs:
    - targets:
       - localhost:9093

rule_files:
  - alerts/*.yml

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['prometheus.sub.tld:9090']

  - job_name: 'node'
    file_sd_configs:
      - files:
        - node_targets/*.yml
    dns_sd_configs:
      - names:
        - _node._tcp.prometheus.sub.tld
    static_configs:
      - targets: ['prometheus.sub.tld:9100']

  - job_name: 'wmi'
    dns_sd_configs:
      - names:
        - _wmi._tcp.prometheus.sub.tld

  - job_name: 'mysqld'
    file_sd_configs:
```

```
    - files:
      - mysqld_targets/*.yml
  dns_sd_configs:
    - names:
      - _mysqld._tcp.prometheus.sub.tld

- job_name: 'nginx'
  file_sd_configs:
    - files:
      - nginx_targets/*.yml
  dns_sd_configs:
    - names:
      - _nginx._tcp.prometheus.sub.tld
- job_name: 'elasticsearch'
  file_sd_configs:
    - files:
      - elasticsearch_targets/*.yml
  dns_sd_configs:
    - names:
      - _elasticsearch._tcp.prometheus.sub.tld

- job_name: 'blackbox'
  file_sd_configs:
    - files:
      - blackbox_targets/*.yml
      - blackbox_clients/*.yml
  dns_sd_configs:
    - names:
      - _blackbox._tcp.prometheus.sub.tld
  metrics_path: /probe
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [module]
      target_label: __param_module
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: IP:9115

- job_name: 'blackbox_us'
  file_sd_configs:
    - files:
```

```
      - blackbox_targets/*.yml
      - blackbox_clients/*.yml
  dns_sd_configs:
    - names:
      - _blackbox._tcp.prometheus.sub.tld
  metrics_path: /probe
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [module]
      target_label: __param_module
    - source_labels: [__param_target]
      target_label: instance
    - target_label: job
      replacement: blackbox_us
    - target_label: __address__
      replacement: IP:9115

- job_name: 'redis_exporter'
  file_sd_configs:
    - files:
      - redis_targets/*.yml
  dns_sd_configs:
    - names:
      - _redis._tcp.prometheus.sub.tld

- job_name: 'postgres_exporter'
  file_sd_configs:
    - files:
      - postgres_targets/*.yml
  dns_sd_configs:
    - names:
      - _postgres._tcp.prometheus.sub.tld

- job_name: 'minio'
  metrics_path: /minio/prometheus/metrics
  scheme: https
  tls_config:
      insecure_skip_verify: true
  file_sd_configs:
    - files:
      - minio_targets/*.yml
  dns_sd_configs:
```

```
    - names:
      - _minio._tcp.prometheus.sub.tld

 - job_name: 'wg'
   dns_sd_configs:
    - names:
      - _wg._tcp.prometheus.sub.tld

 - job_name: 'unbound'
   dns_sd_configs:
    - names:
      - _unbound._tcp.prometheus.sub.tld

 - job_name: 'drand'
   static_configs:
   - targets: [‘IP:9999']

 - job_name: 'haproxy'
   dns_sd_configs:
    - names:
      - _hproxy._tcp.prometheus.sub.tld
```

Anexo B – Script de instalação de exporters

Tabela 5 – Anexo B - Script de instalação exporters

```
#!/bin/bash

source ../.api

echo "Installing $1 in $2"

case "$1" in
  "node")
    scp etc/cron.d/node_exporter root@$2:/etc/cron.d/node_exporter
    scp -r etc/node_exporter root@$2:/etc/
    scp etc/systemd/system/node_exporter.service root@$2:/etc/systemd/system/node_exporter.service
    scp usr/bin/node_exporter root@$2:/usr/bin/node_exporter
    ssh root@$2 mkdir /etc/node_exporter/text
    ssh root@$2 systemctl daemon-reload
    ssh root@$2 systemctl restart node_exporter
    ssh root@$2 chkconfig node_exporter on
    if [ "$3" != "no" ]; then
```

```
    hos=$(ssh root@$2 "hostname")
    echo $hos
    node ./resolver_prometheus/main.js $1 $hos
   fi
   ;;
 "elasticsearch")
   Scp etc/systemd/system/elasticsearch_exporter.service
root@$2:/etc/systemd/system/elasticsearch_exporter.service
   scp usr/bin/elasticsearch_exporter root@$2:/usr/bin/elasticsearch_exporter
   ssh root@$2 systemctl daemon-reload
   ssh root@$2 systemctl restart elasticsearch_exporter
   ssh root@$2 chkconfig elasticsearch_exporter on
   if [ "$3" != "no" ]; then
    hos=$(ssh root@$2 "hostname")
    echo $hos
    node ./resolver_prometheus/main.js $1 $hos
   fi
   ;;
 "redis")
   scp etc/systemd/system/redis_exporter.service root@$2:/etc/systemd/system/redis_exporter.service
   scp usr/bin/redis_exporter root@$2:/usr/bin/redis_exporter
   ssh root@$2 systemctl daemon-reload
   ssh root@$2 systemctl restart redis_exporter
   ssh root@$2 chkconfig redis_exporter on
   if [ "$3" != "no" ]; then
    hos=$(ssh root@$2 "hostname")
    echo $hos
    node ./resolver_prometheus/main.js $1 $hos
   fi
   ;;
 "mysqld")
   scp etc/systemd/system/mysqld_exporter.service root@$2:/etc/systemd/system/mysqld_exporter.service
   scp usr/bin/mysqld_exporter root@$2:/usr/bin/mysqld_exporter
   scp -r etc/mysqld_exporter root@$2:/etc/
   ssh root@$2 "mysql -e \"CREATE USER 'prometheus'@'localhost' IDENTIFIED BY password';\""
   ssh root@$2 "mysql -e \"GRANT PROCESS, REPLICATION CLIENT, SELECT ON *.* TO
'prometheus'@'localhost' WITH MAX_USER_CONNECTIONS 3;\""
   ssh root@$2 systemctl daemon-reload
   ssh root@$2 systemctl restart mysqld_exporter
   ssh root@$2 chkconfig mysqld_exporter on
   if [ "$3" != "no" ]; then
    hos=$(ssh root@$2 "hostname")
    echo $hos
```

```
    node ./resolver_prometheus/main.js $1 $hos
  fi
  ;;
"nginx")
  scp etc/systemd/system/nginx_exporter.service root@$2:/etc/systemd/system/nginx_exporter.service
  scp etc/nginx/conf.d/status.conf root@$2:/etc/nginx/conf.d/status_prom.conf
  scp usr/bin/nginx_exporter root@$2:/usr/bin/nginx_exporter
  ssh root@$2 systemctl daemon-reload
  ssh root@$2 nginx -s reload
  ssh root@$2 systemctl restart nginx_exporter
  ssh root@$2 chkconfig nginx_exporter on
  if [ "$3" != "no" ]; then
    hos=$(ssh root@$2 "hostname")
    echo $hos
    node ./resolver_prometheus/main.js $1 $hos
  fi
  ;;
"postgres")
  scp etc/systemd/system/postgres_exporter.service root@$2:/etc/systemd/system/postgres_exporter.service
  scp -r etc/postgres_exporter root@$2:/etc/
  scp usr/bin/postgres_exporter root@$2:/usr/bin/postgres_exporter
  ssh root@$2 systemctl daemon-reload
  ssh root@$2 systemctl restart postgres_exporter
  ssh root@$2 chkconfig postgres_exporter on
  if [ "$3" != "no" ]; then
    hos=$(ssh root@$2 "hostname")
    echo $hos
    node ./resolver_prometheus/main.js $1 $hos
  fi
  ;;
"csf")
  ssh root@$2 csf -a IP_PROMETHEUS Prometheus
  ssh root@$2 csf -a IP_PROMETHEUS Prometheus
  ssh root@$2 csf -a IP_PROMETHEUS Prometheus
  ssh root@$2 csf -a IP_PROMETHEUS Prometheus

  ;;
"iptables")
  ip=$(ssh root@$2 "ifconfig  | grep 10.12 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | tr -d '\n'")
  if [ -z "$ip" ]
  then
     ip=$(ssh root@$2 "ifconfig  | grep 10.30 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | tr -d '\n'")
     if [ -z "$ip" ]
```

```
    then
      ip=$(ssh root@$2 "ifconfig  | grep 109.71 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | head -1")
      if [ -z "$ip" ]
      then
        ip=$(ssh root@$2 "ifconfig  | grep 130.185 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | head -1")
        if [ -z "$ip" ]
        then
          ip=$(ssh root@$2 "ifconfig  | grep 94.46 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | head -1")
          if [ -z "$ip" ]
          then
            ip=$(ssh root@$2 "ifconfig  | grep 185.15 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | head -1")
          fi
        fi
      fi
    fi
  fi

  echo "machine ip: $ip"

  echo "#prometheus"
  echo "-A INPUT -s IP_PROMETHEUS -j ACCEPT"
  echo "-A INPUT -s IP_PROMETHEUS -j ACCEPT"
  echo "-A INPUT -s IP_PROMETHEUS -j ACCEPT"
  echo "-A INPUT -s IP_PROMETHEUS -j ACCEPT"
  echo "vi /etc/sysconfig/iptables"
  ssh root@$2
  ;;
"host")
  ip=$(ssh root@$2 "ifconfig  | grep 10.12 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | tr -d '\n'")
  hostname=$(ssh root@$2 "hostname")

  if [ -z "$ip" ]
  then
    ip=$(ssh root@$2 "ifconfig  | grep 10.30 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | tr -d '\n'")
    if [ -z "$ip" ]
    then
      ip=$(ssh root@$2 "ifconfig  | grep 109.71 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | head -1")
      if [ -z "$ip" ]
      then
        ip=$(ssh root@$2 "ifconfig  | grep 130.185 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | head -1")
        if [ -z "$ip" ]
        then
          ip=$(ssh root@$2 "ifconfig  | grep 94.46 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | head -1")
```

```
                if [ -z "$ip" ]
                then
                   ip=$(ssh root@$2 "ifconfig  | grep 185.15 | grep 'inet ' | grep -oP '(?<=inet\s)\d+(\.\d+){3}' | head -1")
                fi
             fi
          fi
       fi
    fi

    echo $ip $hostname
    ssh root@IP_PROMETHEUS "echo $ip $hostname >> /etc/hosts"
    ;;
  *)
    echo "You have failed to specify what to do correctly."
    exit 1
    ;;
esac
```

Anexo C – Estrutura repositório git

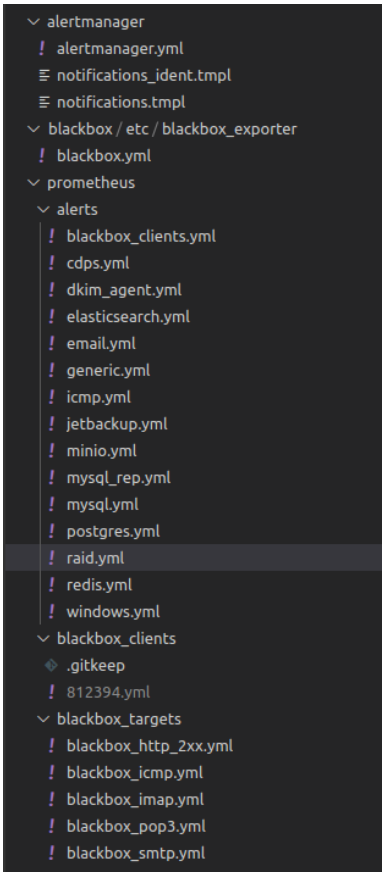

Figura 17 – Anexo C – Estrutura de pastas de repositório GIT (parte 1)

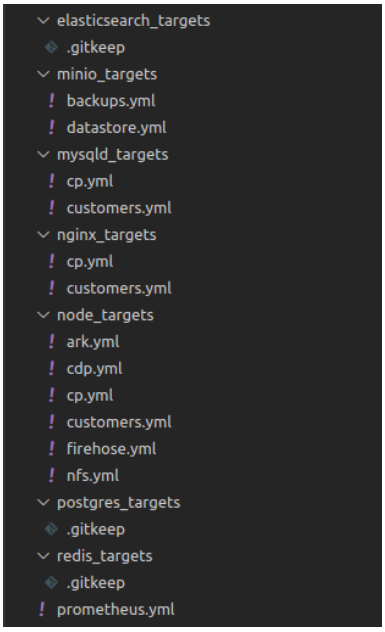

Figura 18 – Anexo C – Estrutura de pastas de repositório GIT (parte 2)

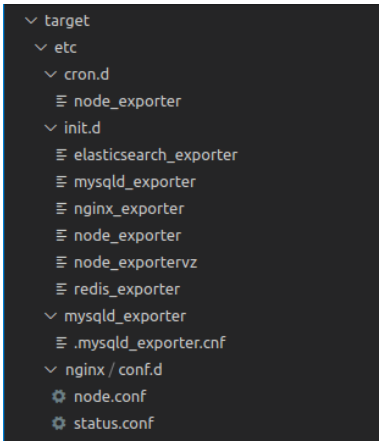

Figura 19 - Anexo C – Estrutura de pastas de repositório GIT (parte 3)

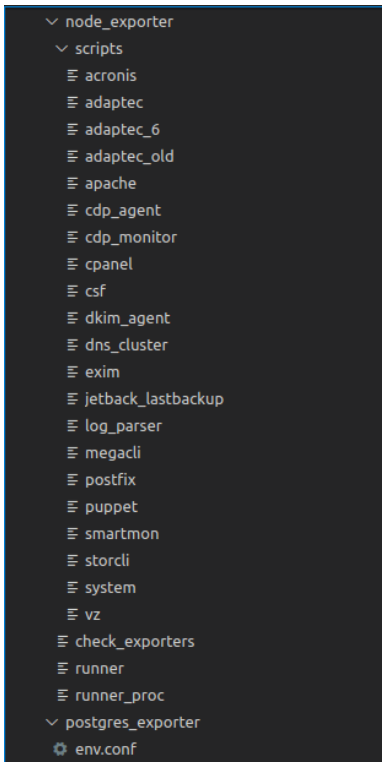

Figura 20 - Anexo C – Estrutura de pastas de repositório GIT (parte 4)

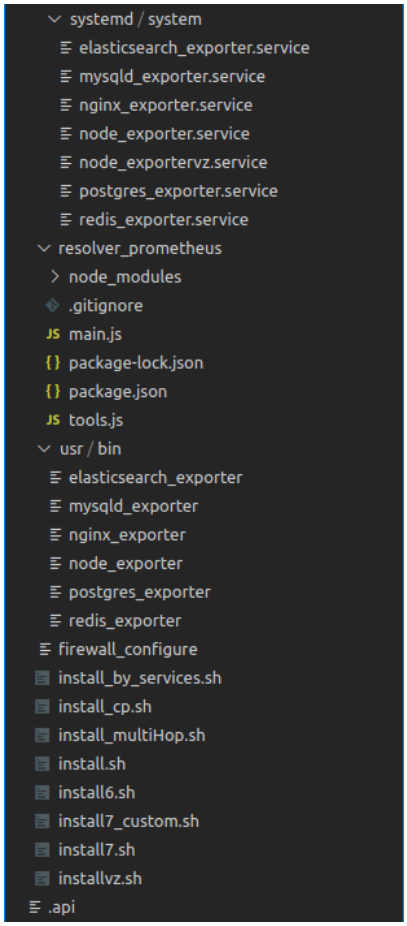

Figura 21 - Anexo C – Estrutura de pastas de repositório GIT (parte 5)

Anexo D – Código de CI/CD

Tabela 6 – Anexo D – Codigo de Implementação CI/CD e Testes

```
deploy_production:
  image: node:latest

  stage: deploy

  environment:
    name: production

  before_script:
    - mkdir -p ~/.ssh
    - echo "$SSH_PRIVATE_KEY" > ~/.ssh/id_rsa
    - chmod 400 ~/.ssh/id_rsa
    - echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config

  script:
    - ssh root@PROMETHEUS_IP "rm -rf /root/prometheus; git clone git@gitlab:gitlab/prometheus.git";
    - ssh root@PROMETHEUS_IP "cp /root/prometheus/prometheus/prometheus.yml
/etc/prometheus/prometheus.yml; chown prometheus:prometheus /etc/prometheus/prometheus.yml";
    - ssh root@PROMETHEUS_IP "rm -rf /etc/prometheus/alerts; cp -r /root/prometheus/prometheus/alerts
/etc/prometheus/alerts; chown prometheus:prometheus /etc/prometheus/alerts";
    - ssh root@PROMETHEUS_IP "cp /root/prometheus/alertmanager/alertmanager.yml
/etc/alertmanager/alertmanager.yml; chown alertmanager:alertmanager /etc/alertmanager/alertmanager.yml";
    - ssh root@PROMETHEUS_IP "cp /root/prometheus/alertmanager/notifications.tmpl
/etc/alertmanager/notifications.tmpl; chown alertmanager:alertmanager /etc/alertmanager/notifications.tmpl";

    - ssh root@PROMETHEUS_IP "rm -rf /etc/prometheus/blackbox_targets; cp -r
/root/prometheus/prometheus/blackbox_targets /etc/prometheus/blackbox_targets; chown prometheus:prometheus
/etc/prometheus/blackbox_targets";
    - ssh root@PROMETHEUS_IP "rm -rf /etc/prometheus/node_targets; cp -r
/root/prometheus/prometheus/node_targets /etc/prometheus/node_targets; chown prometheus:prometheus
/etc/prometheus/node_targets";
    - ssh root@PROMETHEUS_IP "rm -rf /etc/prometheus/mysqld_targets; cp -r
/root/prometheus/prometheus/mysqld_targets /etc/prometheus/mysqld_targets; chown prometheus:prometheus
/etc/prometheus/mysqld_targets";
    - ssh root@PROMETHEUS_IP "rm -rf /etc/prometheus/nginx_targets; cp -r
/root/prometheus/prometheus/nginx_targets /etc/prometheus/nginx_targets; chown prometheus:prometheus
/etc/prometheus/nginx_targets";
    - ssh root@PROMETHEUS_IP "rm -rf /etc/prometheus/elasticsearch_targets; cp -r
/root/prometheus/prometheus/elasticsearch_targets /etc/prometheus/elasticsearch_targets; chown
prometheus:prometheus /etc/prometheus/elasticsearch_targets";
```

```
    - ssh root@PROMETHEUS_IP "rm -rf /etc/prometheus/redis_targets; cp -r
/root/prometheus/prometheus/redis_targets /etc/prometheus/redis_targets; chown prometheus:prometheus
/etc/prometheus/redis_targets";
    - ssh root@PROMETHEUS_IP "rm -rf /etc/prometheus/minio_targets; cp -r
/root/prometheus/prometheus/minio_targets /etc/prometheus/minio_targets; chown prometheus:prometheus
/etc/prometheus/minio_targets";
    - ssh root@PROMETHEUS_IP "rm -rf /etc/prometheus/postgres_targets; cp -r
/root/prometheus/prometheus/postgres_targets /etc/prometheus/postgres_targets; chown prometheus:prometheus
/etc/prometheus/postgres_targets";

    - ssh root@PROMETHEUS_IP "curl -X POST http://localhost:9090/-/reload; curl -X POST
http://localhost:9093/-/reload";

    - ssh root@PUPPET_IP "cd /root/files/prometheus; git reset --hard; git pull;";

  only:
    - master


all_tests:
  image: node:latest

  before_script:
    - echo "the project directory is - $CI_PROJECT_DIR"
    - mkdir -p ~/.ssh
    - echo "$SSH_PRIVATE_KEY" > ~/.ssh/id_rsa
    - chmod 400 ~/.ssh/id_rsa
    - echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config

  script:
   - ssh root@PROMETHEUS_IP "rm -rf /root/prometheus";
   - scp -r $CI_PROJECT_DIR root@PROMETHEUS_IP:/root/prometheus
   - ssh root@PROMETHEUS_IP "promtool check config prometheus/prometheus/prometheus.yml";
   - ssh root@PROMETHEUS_IP "amtool check-config prometheus/alertmanager/alertmanager.yml";
```

Anexo E – Script Principal (Runner)

Anexo E-1 Runner

Tabela 7 – Anexo E-1 - Código do Script Principal (Runner)

```
#!/bin/bash

for file in /etc/node_exporter/scripts/*
do
   if [[ -f $file ]]; then
      python /etc/node_exporter/runner_proc "${file##*/}" &
   fi
done

for job in `jobs -p`
do
   wait $job
done
```

Anexo E-2 Runner_proc

Tabela 8 – Anexo E-2 – Código do Script Principal (Runner_proc)

```
#!/usr/bin/env python

import os
import signal
import subprocess
import sys
from threading import Timer

pathScripts = "/etc/node_exporter/scripts"
pathText = "/etc/node_exporter/text"

f = sys.argv[1]

myCmd = "%s/%s" % (pathScripts,f)

def kill():
   print p.pid
   os.kill(os.getpgid(p.pid), signal.SIGTERM)
```

```
print "####################"
print "cmd: ", myCmd

p = subprocess.Popen(myCmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True)

timer = Timer(30, kill)

try:
   timer.start()
   (output, err) = p.communicate()
   p_status = p.wait()

   print "err: ", err
   print "output: ", output
   print "exit code: ", p_status

   if p_status == 0 and len(err) == 0:
      filePath = "%s/%s.prom" % (pathText, f)
      f = open(filePath, "w")
      f.write(output)
      f.close()
   else:
      print output
      print p_status
finally:
   timer.cancel()
```

Anexo F – Script customizado

Tabela 9 – Anexo F- Exemplo de um script customizado

```
#!/bin/sh

echo "system_threads_total $(ps -eo nlwp | tail -n +2 | awk '{ num_threads += $1 } END { print num_threads }')"
```

Anexo G – Ficheiro de Configuração do AlertManager

Tabela 10 – Anexo G – Ficheiro de configuração alertmanager.yml

```
global:


route:
  group_by: ['instance']
  group_wait: 30s
  group_interval: 15s
  repeat_interval: 1h
  receiver: 'ptisp'
  routes:
    - receiver: blackhole
      match:
        severity: informative
    - receiver: 'promapi'
      match_re:
        clientId: '.+'
    - receiver: 'ptisp'
receivers:
- name: 'ptisp'
  slack_configs:
  - api_url: https://URL/hooks/ID
    channel: 'meeting'
    title: '{{ template "custom_title" . }}'
    text: '{{ template "custom_slack_message" . }}'
    send_resolved: true
- name: blackhole
- name: 'promapi'
  webhook_configs:
  - url: http://localhost:3000/webhooks/blackbox
inhibit_rules:
  - source_match:
      severity: 'critical'
    target_match_re:
      severity: 'warning'
    equal: ['alertname', 'instance', 'job']
templates:
  - notifications.tmpl
```

Anexo H – Dashboards Grafana

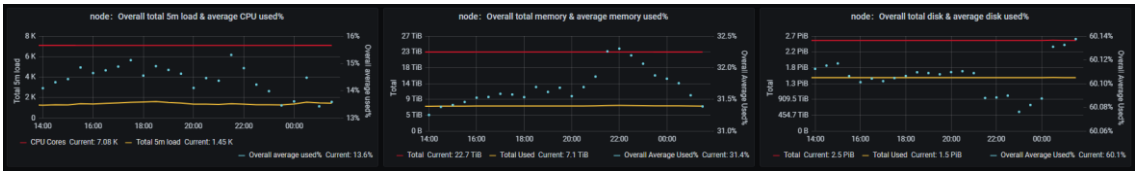

Figura 22 – Anexo H – Dashboard com métricas overall node exporter

Figura 23 – Anexo H – Dashboard Node Expoter

Figura 24 – Anexo H – Dashbaord Mysql Exporter

Figura 25 – Anexo H – Dashboard Nginx Exporter