Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Grado en Ingeniería Informática

Trabajo Fin de Grado

Grafana, Alert-Manager and Prometheus Manager

Autor: Arturo Vidal Peña
Tutor(a): Pérez Costoya, Fernando
Empresa colaboradora: Accenture Technologies S.L.

Madrid, Junio 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Grado en Ingeniería Informática

Título: Grafana, Alert-Manager and Prometheus Manager

Junio 2022

Autor:	Arturo Vidal Peña
Tutor:	Pérez Costoya, Fernando Departamento de Arquitectura y Tecnología de Sistemas Informáticos ETSI Informáticos Universidad Politécnica de Madrid
Empresa colaboradora:	Accenture Technologies S.L.

Resumen

A la hora de añadir un *target* a la monitorización *onpremise* con Prometheus, se puede hacer de dos formas distintas:

- De manera estática, indicando en un mismo fichero de configuración todos los *targets* de cada proyecto, con las distintas variaciones de etiquetas de cada uno.
- De manera dinámica, separando los distintos proyectos en ficheros `JSON`, que contienen la lista de *targets* de cada uno.

Sin embargo, ambas configuraciones siguen requiriendo una inversión de tiempo bastante grande, ya que se debe añadir manualmente cada uno de los targets a la configuración de Prometheus.

Este proyecto busca solventar esto, usando una aplicación externa, que, según se especifique, genere automáticamente los ficheros JSON necesarios y añada los nuevos targets a la configuración de Prometheus.

Palabras clave: prometheus, configuración, automatización

Abstract

When adding a target to an on-premise Prometheus monitoring configuration, it can be done in two ways:

- Statically, specifying all the targets in a single configuration file, with different variations of tags for each one.
- Dynamically, separating the different projects in `JSON` files, which contain the list of targets for each one.

However, both configurations require a large investment of time, because it is necessary to manually add each one of the targets to the Prometheus configuration.

This projects aims to solve this by using an external application, which, depending on how it is specified, will generate automatically the `JSON` files needed and add the new targets to the Prometheus configuration.

Keywords: prometheus, configuration, automation

Tabla de contenidos

Capítulo 1

Introducción

Prometheus

Prometheus[1] es una herramienta open-source para monitorizar sistemas. Esto lo consigue mediante el uso de agentes exportadores de métricas (*exporters*), instalados en las máquinas objetivo (*targets*), que recogen y publican los datos de las mismas.

Alertmanager

Alertmanager[2] maneja las alertas enviadas por prometheus. Se encarga de eliminar duplicados, agrupar y distribuir las alertas a los servicios configurados. Es capaz de crear silencios e inhibiciones entre alertas.

Grafana

Grafana[3] es una herramienta utilizada para crear *dashboards* y paneles en los que tener una representación más visual de los datos recogidos por, en este caso, Prometheus.

1.1. Estado del arte

A la hora de añadir un *target* a la configuración de la monitorización con Prometheus, se debe añadir una nueva entrada en el fichero `prometheus.yml`, según el siguiente formato:

```
1    ---
2     - job: prometheus
```

```
3     static_configs:
4     - targets:
5       - "localhost:9090"
6       labels:
7         jobname: prometheus
8         monitoring: true
9         alerting: true
```

Listing 1.1: Configuración estática de prometheus.yml:

Una buena modificación consiste en usar descubrimiento dinámico de ficheros (*File System Discovery*) para obtener la lista de targets de manera dinámica:

```
1     ---
2     - job: prometheus
3     file_sd_configs:
4     - files:
5       - "targets/*.json"
6       refresh_interval: 2m
7     relabel_configs:
8     - source_labels: [jobname]
9       regex: 'prometheus'
10      action: keep
```

Listing 1.2: Usando file_sd_configs:

```
1       [
2         {
3           "targets": [
4             "localhost:9090"
5           ],
6           "labels": {
7             "jobname": "prometheus
                 ",
8             "monitoring": "true",
9             "alerting": "true"
10          }
11        }
12      ]
```

Listing 1.3: Especificación de targets en JSON:

Sin embargo, esto sigue requiriendo una enorme cantidad de tiempo, especialmente con proyectos de larga envergadura, al seguir teniendo que generar los ficheros de manera manual. Es por eso que surge la idea de esta aplicación, que genere dichos ficheros y configuraciones de manera automática, con mayor rapidez y menor número de posibles errores.

Capítulo 2

Desarrollo

Intro al desarrollo

2.1. Base de datos

Se ha desarrollado una base de datos en SQLite3[4], para facilitar la portabilidad de la aplicación entre máquinas durante la fase de desarrollo. Sin embargo, de cara a una implementación productiva, se recomienda portar a una base de datos relacional en SQL.

El diagrama de la base de datos se encuentra en la Figura 1 del Anexo.

2.2. Frontend

Para desarrollar la interfaz (GUI) sobre la cual el usuario interactúa con el programa, se utilizó el framework ElectronJS[5].

2.2.1. ElectronJS

ElectronJS es un *framework* de programación que permite crear aplicaciones de escritorio con tecnología web. Actualmente está siendo usado para aplicaciones como *Visual Studio Code*, *Microsoft Teams* y *Twitch*, entre muchas otras.

Este *framework* se basa en *Chromium* y *Node.js* para construir aplicaciones de escritorio usando *HTML*, *CSS* y *JavaScript*, de manera que su ejecución sea rápida, eficiente y fácilmente configurable. También permite construir las aplicaciones independientemente de la plataforma y sistema operativo, pudiendo ser ejecutadas en *Windows*, *Linux* y *MacOS*.

2.3. Backend

Al realizar operaciones en la GUI explicada anteriormente, se realizarán llamadas a un backend en Python, que se encargará de la gestión de la base de datos, la ejecución de los playbooks de Ansible y la configuración de los targets.

2.3.1. Docker

Docker[6] es una herramienta open source multiplataforma, con la cual se crean contenedores en los que ejecutar y probar la infraestructura, de forma que se pueda ejecutar cualquier pieza de software independientemente al sistema operativo.

2.3.1.1. Imágenes

Las imágenes[7] de Docker son colecciones de instrucciones, directorios y ficheros, a modo de plantilla, sobre la cual se crea un nuevo contenedor. Podrían ser comparables a una imagen de sistema, por ejemplo, de una máquina virtual.

Se pueden crear imágenes nuevas tomando como base otras ya existentes, pudiendo añadir distintas configuraciones que estarán disponibles en los contenedores que se creen a partir de ellas.

2.3.1.2. Contenedores

Los contenedores[8] son instancias ejecutables de una imagen[7] de Docker. Un contenedor consta de:

- Una imagen
- Un entorno de ejecución
- Un conjunto de instrucciones y comandos

Uso de Docker en este proyecto

Para este proyecto se han creado cuatro contenedores en Docker:

- Uno para funcionar como servidor de Prometheus, que recogerá las métricas de los targets que se usen en la fase de pruebas.
- Uno para funcionar como Alertmanager, para configurar avisos y silencios que se requieran y comprobar que las alertas se configuran correctamente mediante la aplicación.

- Un tercero, con Grafana, para comprobar los datos recogidos de manera más visual que usando la interfaz propia de Prometheus.
- El último, con la aplicación, que la ejecutará y nos permitirá introducir los datos necesarios para cada configuración.

De esta forma, reducimos las posibilidades de error por encontrarnos en distintas máquinas, y nos abastecemos de una infraestructura básica con la que simular un entorno productivo.

2.3.2. User Interface

2.3.3. Ansible

Ansible[9] es una herramienta de automatización, aprovisionamiento y configuración. Utiliza el protocolo SSH (aunque también puede utilizar otros como Kerberos o LDAP si fuera necesario) para realizar operaciones en las máquinas objetivo sin necesidad de tener un agente instalado en ellas.

2.3.3.1. Inventario

Ansible puede puede atacar a distintos nodos o *hosts* al mismo tiempo. Para ello, lo más común es crear un listado de los mismos, agrupándolos según una serie de criterios, como por ejemplo:

- Región
- Entorno (productivo, test, preproductivo, etc.)
- Funcionalidad (servidor web, base de datos, etc.)

De esta manera, un mismo *host* puede estar incluido en varios grupos al mismo tiempo.

Además, también pueden especificarse distintas variables, que Ansible usará en la ejecución, tanto para cada *host* como para cada grupo.

2.3.3.2. Playbooks

Los *playbooks* son una forma de automatizar tareas repetitivas en las máquinas. En ellos se especifican una serie de tareas a realizar, y los *hosts* en los que se ejecutará cada una.

Uso de Ansible en este proyecto

En este proyecto, se ha creado un playbook (ver Listing 1) que se encarga de instalar el agente *node_exporter* en las máquinas Linux que se especifiquen. Para ello, utiliza un rol de la comunidad de Ansible, que se encarga de realizar la instalación.

2.3.4. Exporter

Una de las funciones de la aplicación es llevar un registro de las versiones de los exporters que hay instaladas en cada máquina, para poder ser actualizados en caso de que hubiera actualizaciones.

Para ello, se ha creado un exporter, siguiendo la guía oficial[10], que añade una métrica a las publicadas por *node_exporter* llamada *version_info*, que contiene la versión de cada agente instalado y la versión más reciente disponible.

Esto nos permite enviar alertas cuando un agente queda desactualizado, y tener un registro de las versiones de los agentes instalados en cada máquina.

Capítulo 3

Resultados, conclusiones y trabajo futuro

Con esta nueva aplicación, consideramos que se solventa en gran medida el problema de la configuración de los targets, ya que se puede realizar de forma automática, sin necesidad de intervención humana, lo cual minimiza los errores que pudieran cometerse, aumenta la productividad y la eficiencia.

Del trabajo extraemos el saber que es una herramienta con gran potencial para la monitorización con Prometheus, puesto que permite ahorrar tiempo en la configuración, dejando espacio para otros proyectos y avances que requieran de una mayor dedicación.

De igual forma, opinamos que es una herramienta con alto potencial de crecimiento, pudiendo abarcar una amplia gama de agentes que instalar y configurar, pudiendo incluso configurar *dashboards* tipo en Grafana o rutas predefinidas en Alertmanager para los envíos de correo de nuevos clientes o proyectos.

Capítulo 4

Análisis de impacto

Rellenar

En este capítulo se realizará un análisis del impacto potencial de los resultados obtenidos durante la realización del TFG, en los diferentes contextos para los que se aplique:

- Personal
- Empresarial
- Social
- Económico
- Medioambiental
- Cultural

En dicho análisis se destacarán los beneficios esperados, así como también los posibles efectos adversos.

Se recomienda analizar también el potencial impacto respecto a los Objetivos de Desarrollo Sostenible (ODS), de la Agenda 2030, que sean relevantes para el trabajo realizado (ver enlace)

Además, se harán notar aquellas decisiones tomadas a lo largo del trabajo que tienen como base la consideración del impacto.

Bibliografía

[1] Prometheus. (2022) Prometheus documentation. [Online]. Available: https://prometheus.io/docs/introduction/overview/

[2] ——. (2022) Alertmanager documentation. [Online]. Available: https://prometheus.io/docs/alerting/latest/alertmanager/

[3] Grafana. (2022) Grafana documentation. [Online]. Available: https://grafana.com/docs/grafana/latest/introduction/

[4] SQLite. (2022) Sqlite documentation. [Online]. Available: https://www.sqlite.org/about.html

[5] O. Foundation. (2022) Electronjs documentation. [Online]. Available: https://www.electronjs.org/docs/latest

[6] Docker. (2022) Docker documentation. [Online]. Available: https://docs.docker.com/

[7] A. S. Gillis. (2021) Docker image. [Online]. Available: https://www.techtarget.com/searchitoperations/definition/Docker-image

[8] Docker. (2022) Docker container. [Online]. Available: https://docs.docker.com/glossary/#container

[9] RedHat. (2022) Ansible documentation. [Online]. Available: https://docs.ansible.com/ansible/latest/

[10] Prometheus. (2022) Prometheus documentation. [Online]. Available: https://prometheus.io/docs/instrumenting/writing_exporters/

[11] S. Lukasczyk and G. Fraser, "Pynguin: Automated unit test generation for python," ser. 44th International Conference on Software Engineering Companion (ICSE '22 Companion), 2022.

[12] ETSIINF. (2017) Recomendaciones sobre el contenido de la memoria final. [Online]. Available: http://www.fi.upm.es/?pagina=1475

[13] RedHat. (2022) Ansible inventory. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html

[14] ——. (2022) Ansible playbooks. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html

[15] U. Nations. (2022) Crecimiento económico - desarrollo sostenible. [Online]. Available: https://www.un.org/sustainabledevelopment/es/economic-growth/

[16] ——. (2022) Infraestructura - desarrollo sostenible. [Online]. Available: https://www.un.org/sustainabledevelopment/es/infraestructure/

[17] ——. (2022) Objetivos y metas de desarrollo sostenible. [Online]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

Anexo

Base de datos

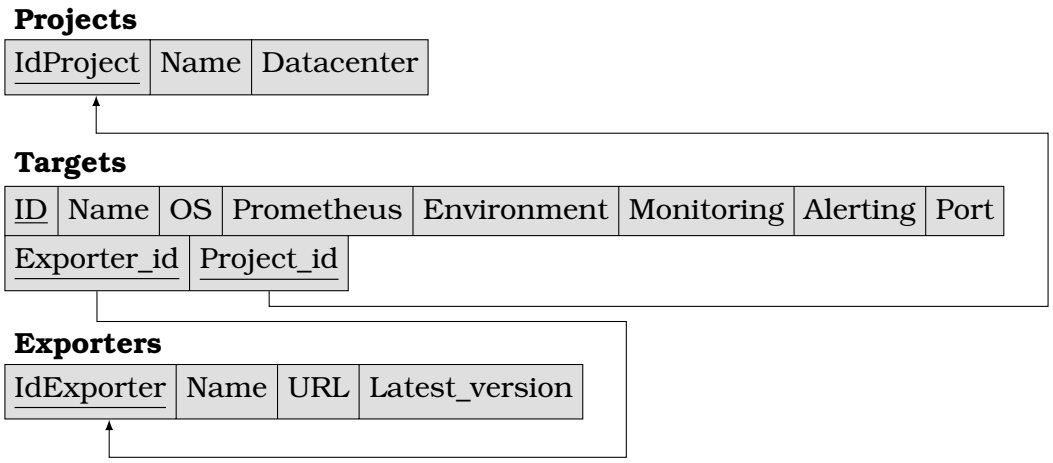

Figura 1: Diagrama relacional de la Base de Datos utilizada

Ansible

```
- name: Install and start Node Exporter
  hosts: node_exporter
  tasks:
    - name: Create node_exporter dir
      become: yes
      file:
        path: '/opt/exporters/node_exporter'
        state: directory
        owner: root
        group: root
    - name: Install node exporter
      become: yes
      import_role:
        name: cloudalchemy.node_exporter
      vars:
        node_exporter_version: latest
        node_exporter_web_listen_address: "0.0.0.0:9100"
        node_exporter_web_telemetry_path: "/metrics"
        node_exporter_textfile_dir: "/opt/exporters/node_exporter/
           textfile_collector"
        node_exporter_enabled_collectors:
          - systemd
          - textfile:
              directory: "{{ node_exporter_textfile_dir }}"
        #  - filesystem:
        #      ignored-mount-points: "^/(sys|proc|dev)($|/)"
        #      ignored-fs-types: "^(sys|proc|auto)fs$"
        node_exporter_disabled_collectors: [ ]
        # Internal variables.
        _node_exporter_binary_install_dir: "/opt/exporters/
           node_exporter"
        _node_exporter_system_group: "prometheus"
        _node_exporter_system_user: "{{ _node_exporter_system_group
           }}"
```

Listing 1: Playbook creado para instalar *node_exporter*