



Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**



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 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

Actualmente, a la hora de añadir una máquina objetivo (*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, *targets*

Abstract

In the current time, when adding a host (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, targets

Tabla de contenidos

1. Introducción	1
1.1. Definiciones	1
1.2. Estado del arte	2
1.3. Objetivo del trabajo	5
2. Desarrollo	7
2.1. Base de datos	7
2.2. Frontend	9
2.2.1. ElectronJS	9
2.2.2. Interfaz Gráfica	9
2.3. Backend	10
2.3.1. Docker	10
2.3.1.1. Imágenes	10
2.3.1.2. Contenedores	10
2.3.2. Ansible	11
2.3.2.1. Inventario	11
2.3.2.2. Playbooks	11
2.3.3. Exporter	12
2.3.4. Backend con Python	12
3. Resultados, conclusiones y trabajo futuro	15
4. Análisis de impacto	17
Bibliografía	19
Anexo	21

Capítulo 1

Introducción

1.1. Definiciones

En este apartado explicaremos brevemente los siguientes conceptos, para disponer de un conocimiento base sobre las herramientas más utilizadas y sobre las que se basa este trabajo.

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.

Estos datos, vienen en la forma de métricas: vectores de dimensión variable, en función de las etiquetas asociadas, y con un valor numérico.

Target

En este proyecto, un *target* se entiende como un par $\langle host : puerto \rangle$, donde *host* es el nombre de una máquina que se desea monitorizar, y el *puerto* indica el puerto de red de la máquina en el que se encuentran las métricas publicadas por los *exporters*.

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.

1.2. Estado del arte

La configuración de Prometheus se realiza mediante un fichero `YAML`. En este fichero se especifican, entre otras cosas, los distintos trabajos (*jobs*) a los que serán asignados los *targets*.

Estos *jobs* estarán preferiblemente asociados al *exporter* que se encarga de publicar las métricas, para facilitar la comprensión de los datos. En ellos se especifica el nombre del *job* y la lista de *targets* que serán monitorizados.

A la hora de añadir un *target* a la configuración de un *job*, 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`:

Esto es eficiente únicamente cuando se desean tener monitorizaciones estáticas, o de poca envergadura, como podría ser un proyecto personal o la monitorización de las máquinas de un hogar. Esto se debe a que a cada modificación que sufra el fichero `prometheus.yml` debe reiniciarse el servicio de Prometheus para que acepte la nueva configuración.

Algo similar ocurre tanto con Alertmanager como con Grafana, puesto que en cada modificación de los ficheros `alertmanager.yml` y `grafana.ini` respectivamente, es necesario reiniciar el servicio.

Una buena alternativa presente actualmente únicamente en Prometheus consiste en

Introducción

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:

Esto nos permite poder añadir los *targets* en un fichero JSON separado, sin tener que modificar el fichero `prometheus.yml` en cada actualización, con el subsiguiente reinicio del servicio Prometheus. Así, especificando el tiempo que tardará Prometheus en reinspeccionar los distintos ficheros JSON que se indiquen (en el ejemplo superior, dos minutos), podremos añadir, modificar o eliminar *targets* sin tener que reiniciar el servicio cada vez.

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 a partir de los datos introducidos por el usuario, con mayor rapidez y menor número de errores posibles, y que reinicia el servicio de los servicios si fuera necesario porque se haya modificado el fichero de configuración correspondiente.

Ejemplo de flujo de trabajo de forma manual

Supongamos que tenemos un proyecto de monitorización de un cliente en nuestra empresa, que tiene las siguientes máquinas a monitorizar, y quiere monitorizar los

siguientes datos:

Entorno	Nombre	SO
Producción	win-prod	Windows
Producción	linux-prod	Linux
Preproducción	win-pre	Windows
Preproducción	linux-pre	Linux
Pruebas	win-test	Windows
Pruebas	lin-test	Linux

- CPU
- Memoria
- Red
- Disco o Sistema de Ficheros (en función del sistema operativo)
- Estado de servicios

A la hora de añadirlos en la monitorización, tendríamos que crear un fichero `JSON`, y añadir lo siguiente para cada máquina:

```
1 {
2   "targets": [
3     "win-prod:9182"
4   ],
5   "labels": {
6     "jobname": "Windows Exporter Produccion",
7     "environment": "Produccion",
8     "SO": "Windows",
9     ... // Otras etiquetas sobre la maquina
10  }
11 }
```

Esto es, porque las etiquetas (*labels*) son distintas para cada máquina. Estas etiquetas son útiles para, a la hora de visualizar los datos, realizar filtros. Como son introducidas en cada métrica que Prometheus asocia a este job, de ponerlas juntas no podrían diferenciarse las máquinas entre sí.

Por tanto, habría que duplicar el código anterior (en este caso) seis veces para poder diferenciar entre las distintas máquinas. En el caso de que fuera un proyecto más extenso, esta tarea podría resultar tediosa.

Además de eso, habría que conectarse remotamente a cada una de las máquinas (con la subsecuente solicitud de acceso, creación de usuario y asignación de permisos necesarios, comunicación entre estas máquinas y nuestro servidor de Prometheus, etc.), e instalar y configurar manualmente cada uno de los agentes que vamos a necesitar (en este caso, *Node Exporter* para las máquinas Linux, y *Windows Exporter*

para las máquinas Windows).

Después, habría que crear un fichero de alertas para dichos datos, de forma que Prometheus pueda notificar al Alertmanager que tenga configurado cuándo una alerta está activa. Afortunadamente, este fichero es inusual que se vea modificado, salvo en el filtrado del nombre del cliente, por lo que sería simplemente copiar la plantilla (ver Anexo 1) y hacer las modificaciones necesarias.

Finalmente, habría que añadir las rutas de alertado que el cliente especifique (por ejemplo, por correo electrónico), y añadir el filtro de alertas que serán enviados a dichas rutas. Esto tendrá que hacerse manualmente, pues actualmente tampoco se dispone de una herramienta que lo automatice. Después habría que reiniciar el servicio de Alertmanager, para que tenga en cuenta las modificaciones.

1.3. Objetivo del trabajo

Este trabajo pretende solventar este problema, utilizando herramientas de automatización y una interfaz sencilla para que el usuario pueda realizar estas configuraciones de manera más rápida y eficiente.

También busca mitigar el error humano que pueda aparecer al redactar los ficheros de configuración de Prometheus, y que puedan ser difíciles de detectar. Solventando la mayoría de esos errores mediante la automatización, podríamos reducir exponencialmente el tiempo invertido.

Capítulo 2

Desarrollo

En este capítulo vamos a observar los distintos componentes de los que se forma la aplicación.

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. En esta base de datos, alojaremos los nuevos proyectos que se quieran monitorizar, los *targets* asociados al mismo, y los *exporters* utilizados por la aplicación.

Diagrama

El diagrama de la base de datos es el siguiente:

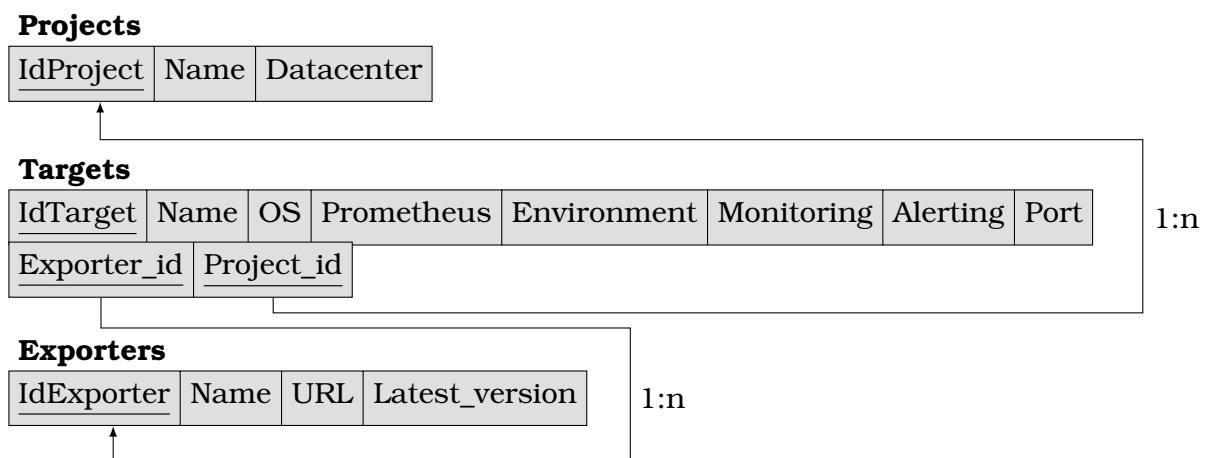


Figura 2.1: Diagrama relacional de la Base de Datos utilizada

Tabla de Proyectos

En esta tabla, almacenaremos los distintos proyectos que tenemos monitorizados con la aplicación. Cada proyecto tendrá:

- **IdProject:** Un identificador único
- **Name:** El nombre del proyecto
- **Datacenter:** El nombre del datacenter donde se encuentra monitorizado

Tabla de *targets*

En esta almacenaremos todos los *targets* que se estén monitorizando. Cada target tendrá:

- **IdTarget:** Un identificador único
- **Name:** El nombre del target
- **OS:** El sistema operativo que utiliza
- **Prometheus:** Si utilizamos el servidor de Prometheus de producción (prod) o no (nonprod)
- **Environment:** El nombre del entorno al que pertenece
- **Monitoring:** Si se está monitorizando o no
- **Alerting:** Si se está alertando o no
- **Port:** El puerto de la máquina desde el que se recogen las métricas
- **Exporter_id:** El identificador del exporter que se utiliza para este *target*
- **Project_id:** El identificador del proyecto al que pertenece

Tabla de Exporters

En esta almacenaremos todos los *exporters* que usemos en la aplicación. Para cada *exporter* tendremos:

- **IdExporter:** Un identificador único
- **Name:** El nombre del exporter
- **URL:** La url del repositorio donde se aloja
- **Latest_version:** La última versión disponible, para un acceso rápido

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*.

2.2.2. Interfaz Gráfica

En esta sección observaremos las distintas ventanas que se utilizan en la aplicación y cómo crear proyectos y añadir *targets* a los mismos.

Pantalla de inicio

En esta primera pantalla (véase Figura 2), podemos buscar un proyecto en la base de datos. Sólomente si el proyecto fue añadido por la aplicación estará disponible para consulta. En caso de no encontrarlo, mostrará una alerta indicándolo (véase Figura 3).

Creación de un nuevo proyecto

Para poder crear y añadir un proyecto a la base de datos de la aplicación, se deberán introducir únicamente el nombre del proyecto y, opcionalmente, el *Datacenter* en el que se ubica (véase Figura 4).

Añadir *targets*

A la hora de añadir nuevos *targets* a la monitorización, elegiremos un proyecto del desplegable, que contiene todos los proyectos presentes en la base de datos (véase Figura 5). En caso de que el proyecto no esté, nos dará la opción de añadirlo directamente sin tener que cambiar de ventana para ello (véase Figura 6).

Una vez seleccionado el proyecto al que pertenece el nuevo *target*, aparecerá una tabla con los campos que se deben rellenar (véase Figura 7). También aparecerán los *targets* ya presentes en el proyecto, de forma que podamos comprobar fácilmente si un *target* en cuestión ya está introducido y no generar duplicados.

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.

Estos contenedores tienen montados los directorios de la aplicación que necesita cada uno para funcionar, de forma que simulan un entorno de ejecución. 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. 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.

2.3.2.1. Inventario

Ansible 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.

2.3.2.2. 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 2) 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.

2.3.3. 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.

2.3.4. Backend con Python

Como ya hemos visto previamente (ver sección 2.2.2), al interactuar con la aplicación se llama a un backend de Python que ejecuta los distintos componentes.

Llamadas a la base de datos

Al crear un nuevo proyecto, o añadir un *target* a un proyecto ya existente, se realizan llamadas a la base de datos. Esto se realiza mediante el módulo de Python llamado que nos permite interactuar con las bases de datos que le configuremos más cómodamente que con las que nos proporciona la librería propia de Python.

Ansible

Una vez se ha añadido un nuevo *target* a la aplicación, se llama a una clase propia de Python que comunica con ansible mediante el módulo oficial. Esta clase primero añadirá el nombre del *target* a un fichero de inventario de ansible, y luego ejecutará el *playbook* creado para instalar el agente *node_exporter* contra las máquinas presentes en el inventario.

Prometheus y Alertmanager

Al crear un nuevo proyecto, Python genera un fichero `JSON`, que deja inicialmente vacío, a la espera de rellenarse con nuevos *targets*. También se crea un nuevo fichero de alertas para las métricas que se reciban relativas al proyecto una vez se añadan *targets* al mismo. Este fichero será plenamente funcional, puesto que hasta que no se reciban métricas que contengan el proyecto (introducidas por la configuración de Prometheus), no se harán efectivas. Seguidamente, se reinicia el servicio de Prometheus, para tener en cuenta las nuevas alertas.

Al añadir un nuevo *target*, se añade una nueva entrada al fichero `JSON`, con el mismo formato que se ha explicado previamente, y con los datos introducidos al añadir el *target* (véase sección 2.2.2). Este fichero `JSON` será inspeccionado automáticamente por Prometheus en función del tiempo que se le haya asignado en la configuración (ver ejemplo en el Listing 1.2).

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.

Flujo de trabajo con la aplicación

Retomando el ejemplo visto anteriormente (véase sección 1.2), ahora vamos a ver cómo sería el mismo flujo de trabajo con esta aplicación.

En este caso simplemente habría que crear un nuevo proyecto con el nombre del cliente, e introducir los datos de las máquinas en la tabla correspondiente (ver Figura 4 y Figura 7). Al confirmar los datos introducidos, se generaría el fichero JSON y el fichero de alertado correspondientes, se instalaría de forma automática el agente necesario en cada máquina, y se añadirían las rutas configuradas en el fichero de

configuración de Alertmanager, reiniciando dicho servicio.

Evidentemente, esto supone un cambio importante en la cantidad de tiempo invertido, ya que si fuera un proyecto de mayor envergadura, la única diferencia que nos supondría es tener que añadir más datos en la aplicación.

Trabajo futuro

En próximas versiones de la aplicación, sería interesante poder instalar más agentes además del *node_exporter*, para poder conseguir una monitorización más completa de cada máquina.

También valoramos la posibilidad de que la aplicación sea más extensible, pudiendo añadir más servidores de Prometheus, Alertmanager y Grafana; para poder trabajar con la infraestructura de Alta Disponibilidad de Prometheus que pueda haber en algunas empresas.

Por otro lado, podríamos pensar en una versión de la aplicación que permita la configuración de máquinas alojadas en servicios cloud, como por ejemplo *Microsoft Azure*, *Google Cloud Platform (GCP)*, *Amazon Web Services (AWS)*.

Por último, sería interesante añadir la opción de crear nuevos dashboards en Grafana a partir de plantillas predefinidas, especialmente cuando sean dashboards específicos para cada proyecto.

Capítulo 4

Análisis de impacto

Con respecto a la Agenda 2030 de los Objetivos de Desarrollo Sostenible[11] (ODS) de la Organización de las Naciones Unidas (ONU), vemos que cumplimos las siguientes metas, de las secciones “8 - Crecimiento económico”[12] e “9 - Infraestructura”[13], respectivamente:

- 8.2** Lograr niveles más elevados de productividad económica mediante la diversificación, la modernización tecnológica y la innovación, entre otras cosas centrándose en los sectores con gran valor añadido y un uso intensivo de la mano de obra.
- 9.4** De aquí a 2030, modernizar la infraestructura y reconvertir las industrias para que sean sostenibles, utilizando los recursos con mayor eficacia y promoviendo la adopción de tecnologías y procesos industriales limpios y ambientalmente racionales, y logrando que todos los países tomen medidas de acuerdo con sus capacidades respectivas.

Esto es, porque consideramos que, a nivel empresarial, el uso de esta aplicación permite aprovechar mejor los recursos dedicados a la monitorización, puesto que requiere de un menor tiempo y dedicación por parte de los trabajadores, resultando en un aumento de la productividad por parte de los mismos. Además, consideramos que la aplicación permite a los usuarios tener una mayor facilidad de uso de los recursos, ya que permite que los mismos sean utilizados de forma más eficiente y eficaz.

Un posible efecto adverso podría ocurrir a nivel de recursos pasivos utilizados para la aplicación.

Considerando, por ejemplo, que se quiera tener una Alta Disponibilidad (*High Availability* en inglés), sería necesario alojar la aplicación en múltiples servidores, para

poder ser accesible de manera ininterrumpida en caso de fallo de alguno de ellos. También, por ejemplo, sería necesario duplicar todos los recursos utilizados por la aplicación, como serían la base de datos, las interfaces de red, etc.

Todo esto conllevaría un mayor gasto computacional y energético, lo cual ocasionaría una mayor emisión de gases de efecto invernadero, con el consecuente impacto medioambiental[14].

Sin embargo, también podemos considerar que el impacto medioambiental que pueda surgir de cada uno de los trabajadores realizando la tarea de manera manual, como se ha explicado anteriormente (véase la sección 1.2), es mayor a este impacto surgido de la aplicación alojada con Alta Disponibilidad.

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] U. Nations. (2022) Objetivos y metas de desarrollo sostenible. [Online]. Available: <https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/>
- [12] ——. (2022) Crecimiento económico - desarrollo sostenible. [Online]. Available: <https://www.un.org/sustainabledevelopment/es/economic-growth/>

- [13] —. (2022) Infraestructura - desarrollo sostenible. [Online]. Available: <https://www.un.org/sustainabledevelopment/es/infrastructure/>
- [14] C. R. Sampedro Guamán, S. A. Machuca Vivar, D. P. Palma Rivera, and B. E. Villalta Jadan, "Impacto ambiental por consumo de energía eléctrica en los Data Centers," *Dilemas contemporáneos: educación, política y valores*, vol. 8, 00 2021. [Online]. Available: http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S2007-78902021000600034&nrm=iso
- [15] RedHat. (2022) Ansible inventory. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html
- [16] —. (2022) Ansible playbooks. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html
- [17] S. Lukasczyk and G. Fraser, "Pynguin: Automated unit test generation for python," ser. 44th International Conference on Software Engineering Companion (ICSE '22 Companion), 2022.
- [18] ETSIINF. (2017) Recomendaciones sobre el contenido de la memoria final. [Online]. Available: <http://www.fi.upm.es/?pagina=1475>

Anexo

Prometheus

```
1 groups:
2 - name: #Project_Name
3   rules:
4
5 # CPU
6 - alert: CPU_Warning
7   expr: cpu_percentage_usage{PROJECT=""} > 90
8   for: 5m
9   labels:
10    severity: warning
11  annotations:
12    summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
13             the host {{ $labels.HOST }} has {{ humanize $value }}% CPU
14             warning usage.
15
16 - alert: CPU_Critical
17   expr: cpu_percentage_usage{PROJECT=""} > 95
18   for: 5m
19   labels:
20    severity: critical
21  annotations:
22    summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
23             the host {{ $labels.HOST }} has {{ humanize $value }}% CPU
24             critical usage.
25
26 # MEMORY
27 - alert: Memory_Warning
```

```

24 expr: memory_percentage_usage{PROJECT=""} > 90
25 for: 5m
26 labels:
27   severity: warning
28 annotations:
29   summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
        the host {{ $labels.HOST }} has {{ humanize $value }}%
        Memory warning usage.
30
31 - alert: Memory_Critical
32 expr: memory_percentage_usage{PROJECT=""} > 95
33 for: 5m
34 labels:
35   severity: critical
36 annotations:
37   summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
        the host {{ $labels.HOST }} has {{ humanize $value }}%
        Memory critical usage.
38
39 # FILESYSTEM/DISK
40 - alert: Filesystem_Warning
41 expr: filesystem_percentage_usage{PROJECT="",mountpoint!~/mnt.*"
        } > 90
42 for: 5m
43 labels:
44   severity: warning
45 annotations:
46   summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
        the host {{ $labels.HOST }} has {{ humanize $value }}%
        usage in {{ $labels.mountpoint }} [warning usage].
47
48 - alert: Filesystem_Critical
49 expr: filesystem_percentage_usage{PROJECT="",mountpoint!~/} > 95
50 for: 5m
51 labels:
52   severity: critical
53 annotations:

```

```
54      summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
        the host {{ $labels.HOST }} has {{ humanize $value }}%
        usage in {{ $labels.mountpoint }} [critical usage].
55
56 - alert: Filesystem_No_Exist_Critical
57 expr: filesystem_exist_mountpoint{PROJECT="",mountpoint=~""} == 1
58 for: 5m
59 labels:
60     severity: critical
61 annotations:
62     summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
        the host {{ $labels.HOST }} {{ $labels.mountpoint }} not
        found.
63
64 - alert: Disk_Warning
65 expr: disk_percentage_usage{PROJECT=""} > 90
66 for: 5m
67 labels:
68     severity: warning
69 annotations:
70     summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
        the host {{ $labels.HOST }} has {{humanize $value}}% in
        volume {{ $labels.volume }} [warning usage].
71
72 - alert: Disk_Critical
73 expr: disk_percentage_usage{PROJECT=""} > 95
74 for: 5m
75 labels:
76     severity: critical
77 annotations:
78     summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
        the host {{ $labels.HOST }} has {{humanize $value}}% in
        volume {{ $labels.volume }} [critical usage].
79
80 # DISK LOAD
81 - alert: Disk_Load_Critical
82 expr: disk_load{PROJECT=""} > 0
83 for: 5m
```

```

84  labels:
85      severity: critical
86  annotations:
87      summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
           the host {{ $labels.HOST }} has excessive disk load (read/
           write).
88
89  # EXPORTER
90  - alert: Exporter_Down_Critical
91      expr: up{PROJECT=""} == 0
92      for: 5m
93      labels:
94          severity: critical
95      annotations:
96          summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
           the host {{ $labels.HOST }} has the {{ $labels.EXPORTER }}
           is down.
97
98  # NETWORK
99  - alert: Network_Warning
100      expr: network_errors{PROJECT=""} > 1
101      for: 5m
102      labels:
103          severity: warning
104      annotations:
105          summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
           the host {{ $labels.HOST }} has warning network problems.
106
107  - alert: Network_Critical
108      expr: network_errors{PROJECT=""} > 50
109      for: 5m
110      labels:
111          severity: critical
112      annotations:
113          summary: From {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
           the host {{ $labels.HOST }} has critical network problems.
114
115  # SERVICES

```



```

116 - alert: Services_Critical
117   expr: service_status{PROJECT=""} == 0
118   for: 5m
119   labels:
120     severity: critical
121   annotations:
122     summary: Form {{ $labels.PROJECT }} {{ $labels.ENVIRONMENT }}
        the host {{ $labels.HOST }} has the service {{ $labels.name
        }} down.

```

Listing 1: Fichero estándar de configuración de alertas en Prometheus

Diagrama UML de la aplicación

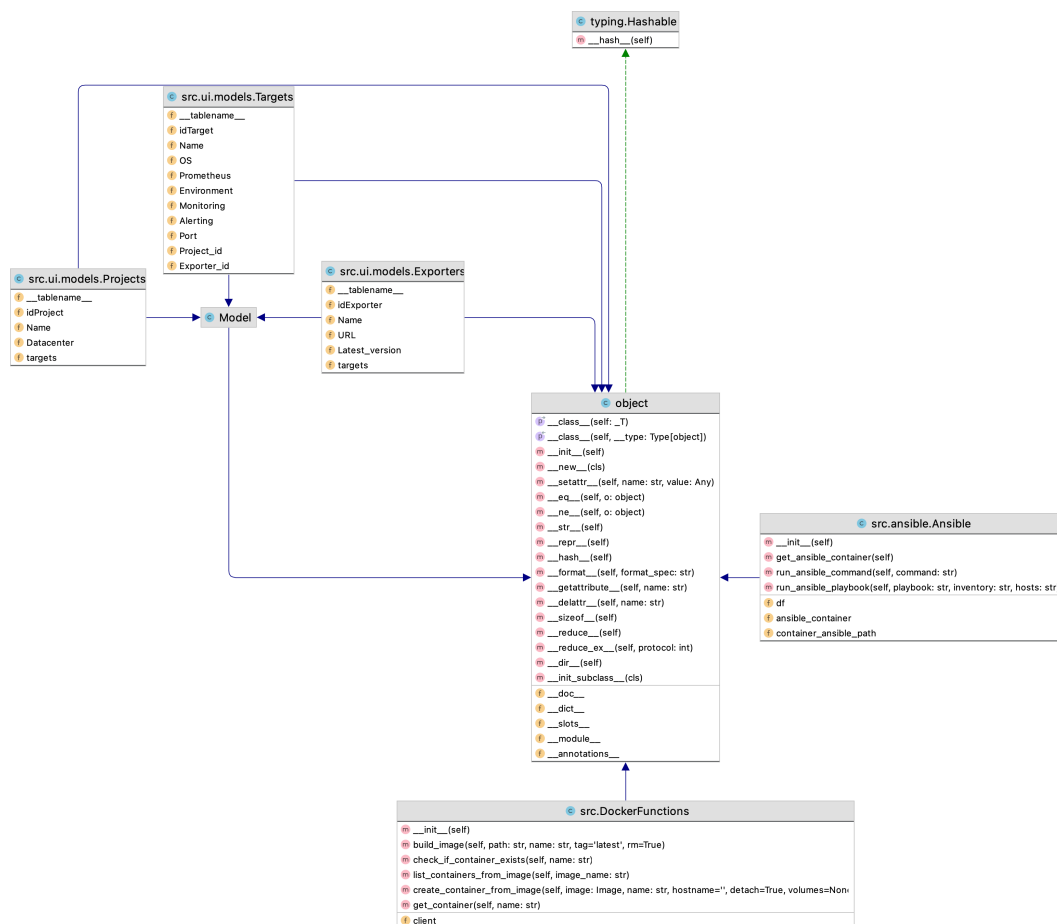


Figura 1: Diagrama UML del backend

Ansible

```

1 - name: Install and start Node Exporter
2   hosts: node_exporter
3   tasks:
4     - name: Create node_exporter dir
5       become: yes
6       file:
7         path: '/opt/exporters/node_exporter'
8         state: directory
9         owner: root
10        group: root
11    - name: Install node exporter
12      become: yes
13      import_role:
14        name: cloudealchemy.node_exporter
15      vars:
16        node_exporter_version: latest
17        node_exporter_web_listen_address: "0.0.0.0:9100"
18        node_exporter_web_telemetry_path: "/metrics"
19        node_exporter_textfile_dir: "/opt/exporters/node_exporter/
20                                     textfile_collector"
21        node_exporter_enabled_collectors:
22          - systemd
23          - textfile:
24              directory: "{{ node_exporter_textfile_dir }}"
25        # - filesystem:
26        #     ignored-mount-points: "^/(sys|proc|dev) ($|/)"
27        #     ignored-fs-types: "^ (sys|proc|auto) fs$"
28        node_exporter_disabled_collectors: [ ]
29        # Internal variables.
30        _node_exporter_binary_install_dir: "/opt/exporters/
31                                             node_exporter"
32        _node_exporter_system_group: "prometheus"
33        _node_exporter_system_user: "{{ _node_exporter_system_group
34                                         }}"

```

Listing 2: Playbook creado para instalar *node_exporter*

Interfaz Gráfica

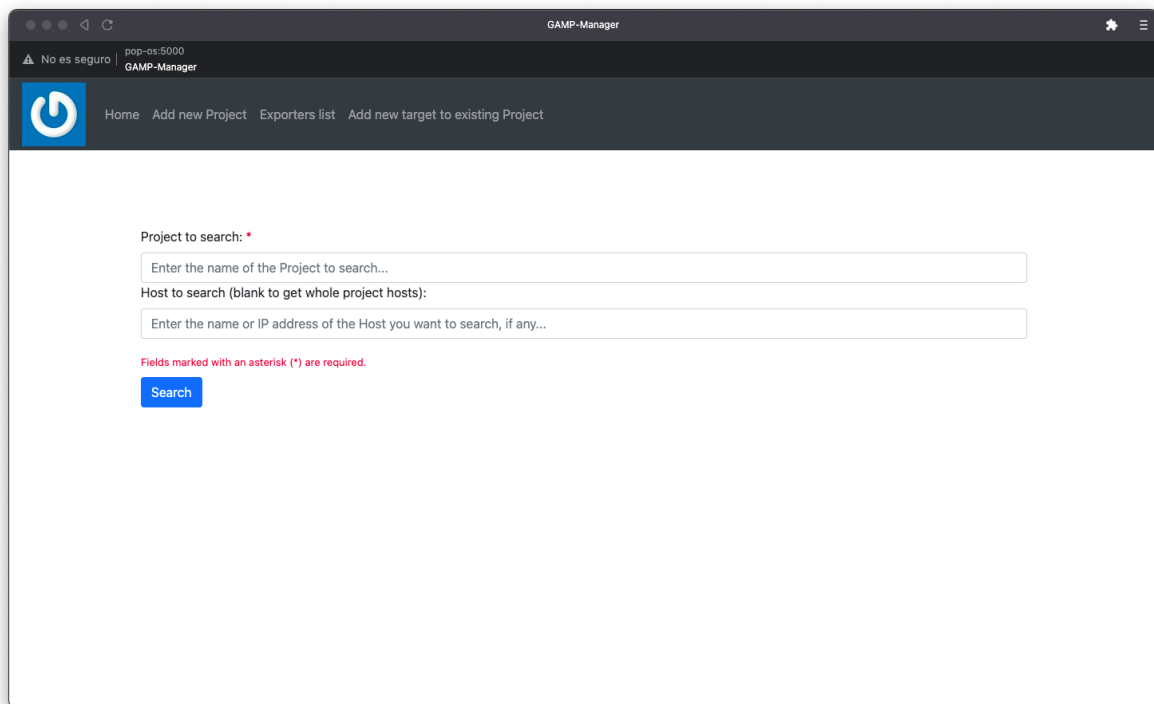


Figura 2: Página principal de selección de proyectos

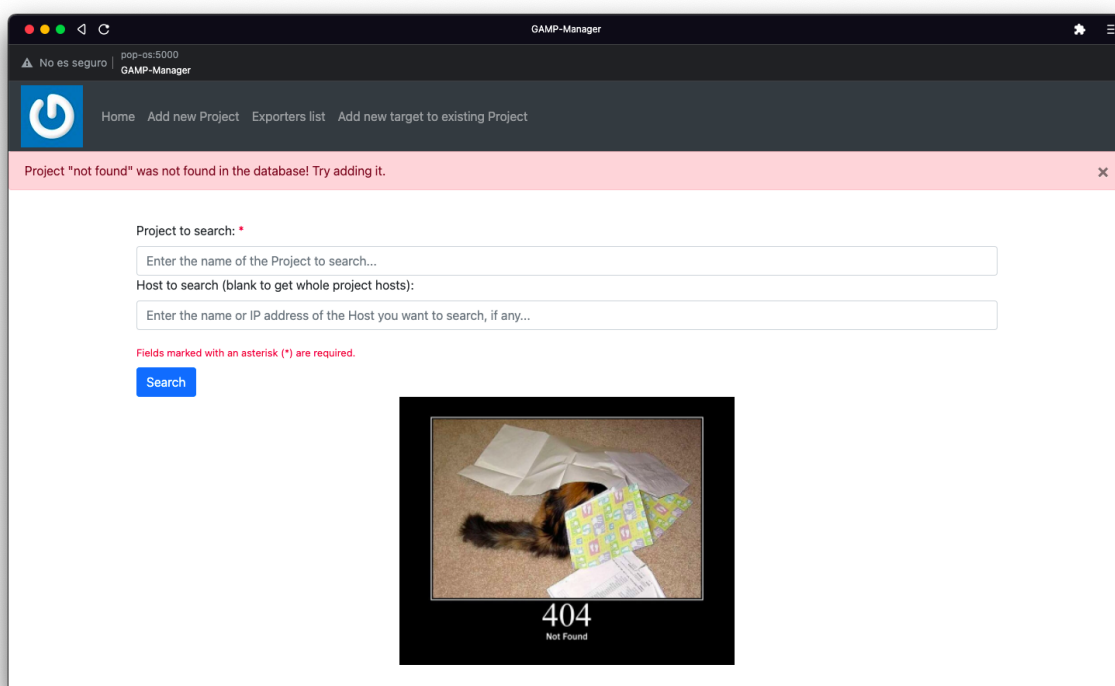


Figura 3: Proyecto no encontrado en la base de datos

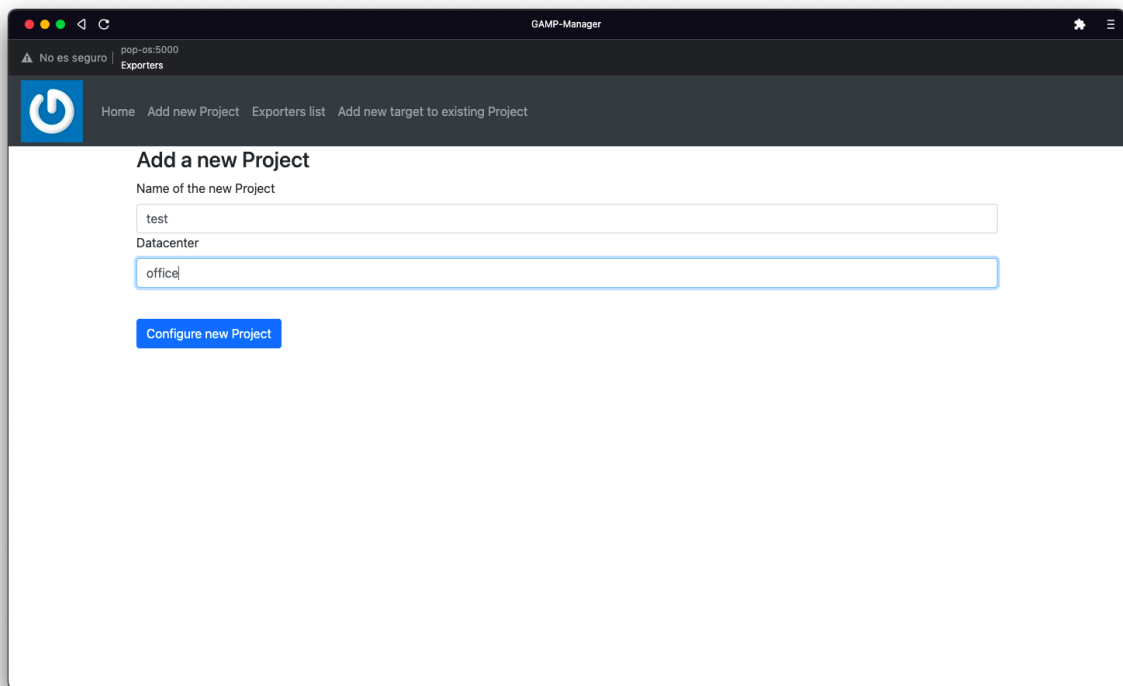


Figura 4: Creación de un proyecto nuevo

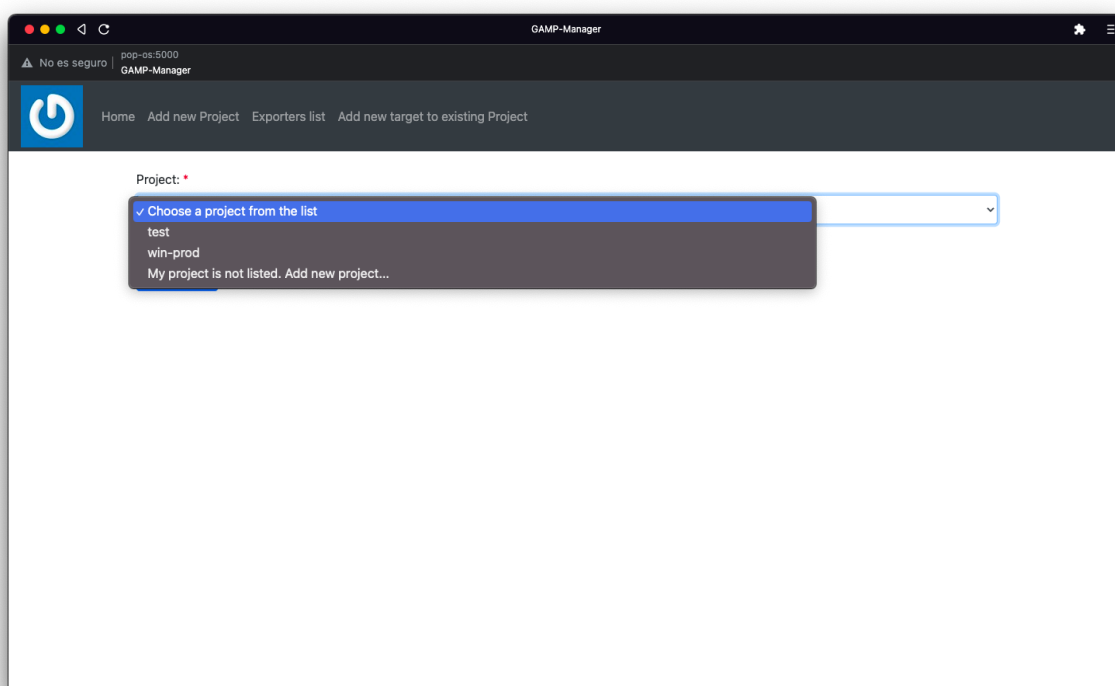
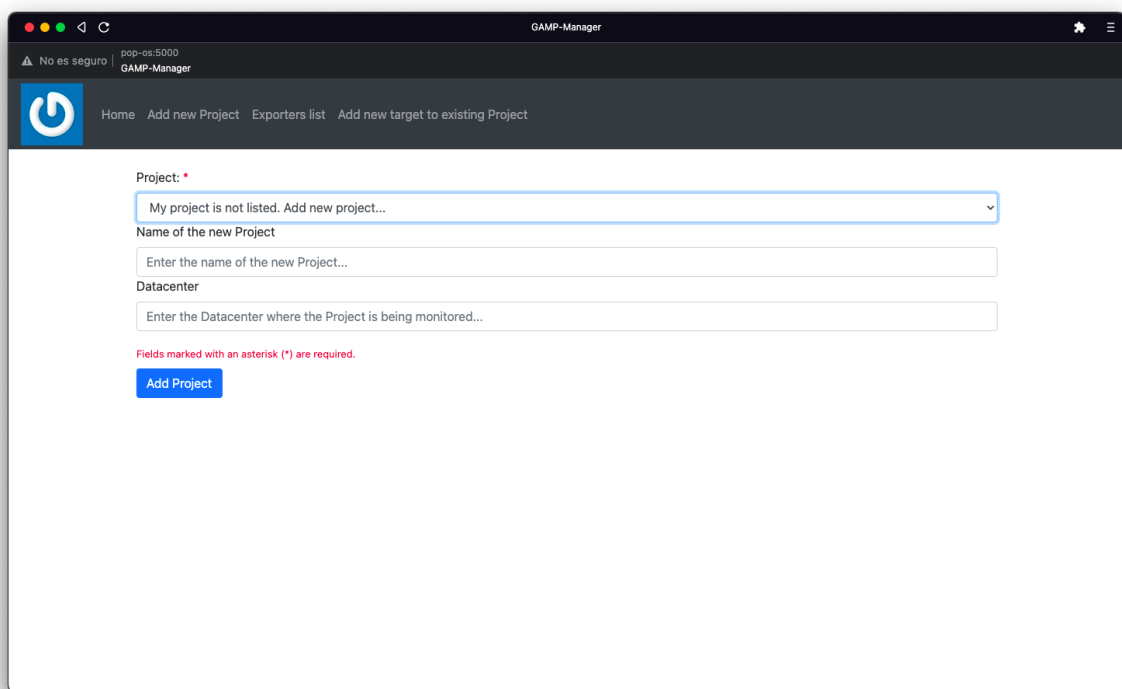


Figura 5: Elección de proyecto para añadir *targets*



The screenshot shows a web browser window with the title 'GAMP-Manager'. The address bar shows 'pop-os:5000' and 'GAMP-Manager'. The browser's security warning says 'No es seguro'. The page has a dark header with a logo and navigation links: 'Home', 'Add new Project', 'Exporters list', and 'Add new target to existing Project'. The main content area is white and contains a form for adding a new project. The form has three required fields, each marked with a red asterisk: 'Project:', 'Name of the new Project', and 'Datacenter'. The 'Project:' field is a dropdown menu with the text 'My project is not listed. Add new project...'. The 'Name of the new Project' field is a text input with the placeholder 'Enter the name of the new Project...'. The 'Datacenter' field is a text input with the placeholder 'Enter the Datacenter where the Project is being monitored...'. Below the fields, there is a red text warning: 'Fields marked with an asterisk (*) are required.' and a blue 'Add Project' button.

Figura 6: Creación de un nuevo proyecto al intentar añadir *targets*

Project: *

test

Targets already present in the Project test:

Name	OS	Prometheus	Environment	Monitoring	Alerting	Port	Exporter
New target	Choose	Choose if it is a Produ	Environment...	Choose if the target	Choose if the tar	Port...	Exporter...

Fields marked with an asterisk (*) are required.

Add Target

Figura 7: Creación de un target en un proyecto existente