Universitat Oberta de Catalunya

uoc.edu

ProtecTor: Sistema de detección de anomalías y alerta para la red Tor

David García Núñez
Máster Universitario de Ciberseguridad y Privacidad
Protocolos criptográficos y aplicaciones de seguridad

Profesora consultora: **Silvia Puglisi**
Profesora responsable de la asignatura: **Cristina Pérez Solà**

Diciembre de 2021

Este trabajo está dedicado a mi hijo Gabriel, a quién tantas y tantas horas de nuestro tiempo he robado.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*ProtecTor: Sistema de detección de anomalías y alerta para la red Tor*
Nombre del autor:	*David García Núñez*
Nombre del consultor/a:	*Silvia Puglisi*
Nombre del PRA:	*Cristina Pérez Solà*
Fecha de entrega (12/2021):	12/2021
Titulación:	*Máster Universitario en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*Protocolos criptográficos y aplicaciones de seguridad*
Idioma del trabajo:	*Español*
Palabras clave	*Tor, Detección de anomalías, Anticensura*

Resumen del Trabajo:

El presente trabajo tiene por objeto la implementación de un sistema de detección de anomalías y alerta para la red Tor. Para su desarrollo y despliegue se ha empleado un diseño basado en componentes independientes y orientado a prototipos.

El sistema se compone de varios módulos que permiten obtener datos de telemetría publicados por el proyecto Tor, procesarlos, almacenarlos, analizarlos a través de un componente modular de análisis de anomalías y emitir alertas a través de un canal bidireccional en forma de bot de la plataforma Telegram que, adicionalmente, permite interactuar con el sistema a través de múltiples comandos.

Además, añade la visualización de datos a través de cuadros de mandos servidos por Grafana y un módulo de plotting desarrollado para visualizar gráficas a través del bot.

El resultado es una plataforma funcional, modular y extensible que permite a cualquier persona interesada o grupo de interés realizar un despliegue sencillo gracias a la implementación basada en contendores Docker de todo el sistema.

El trabajo, fruto de la asimilación de múltiples competencias, ha sido publicado bajo licencia GPL. Se ha realizado de forma satisfactoria en todas sus etapas, presenta un alto grado de madurez y permite su explotación en ambientes productivos.

Si bien, se recomienda la extensión del componente de análisis en forma de módulos que implementen algoritmos de detección de anomalías adicionales

para permitir un análisis comparativo de las diferentes alertas que puedan producirse.

Abstract:

This work aims at implementing an anomaly detection and alerting system for the Tor network. A prototype-oriented, independent component-based design has been used for its development and deployment.

The system is composed of several modules that allow obtaining telemetry data published by the Tor project, processing, storing, and analyzing them through a modular anomaly analysis component and issuing alerts through a bidirectional channel in the form of a bot based on the Telegram platform that, additionally, allows interacting with the system through multiple commands.

It also adds data visualization through dashboards served by Grafana and a plotting module developed to visualize graphs through the bot.

The result is a functional, modular and extensible platform that allows any interested person or group of interest to perform a simple deployment thanks to the Docker container-based implementation of the entire system.

This work is the result of the assimilation of multiple competences and has been released under GPL license. It has been carried out satisfactorily in all its stages, presents a high degree of maturity and allows its exploitation in productive environments.

However, it is recommended that the analysis component be extended in the form of modules that implement additional anomaly detection algorithms to allow a comparative analysis of the different alerts that may occur.

Tabla de contenido

Lista de figuras

1 Introducción

1.1 Contexto y justificación del trabajo

El presente trabajo de Fin de Máster tiene por objeto la construcción de un sistema de captación de eventos de red, detección de anomalías, visualización de datos y emisión de alertas para la red Tor (ver anexo I).

Estos sistemas permiten, mediante la lectura, registro y procesamiento de datos, detectar anomalías sobre el estado de una red. Entendiendo por anomalía, aquellos eventos que influyen en algunas de las capacidades de la red. Por ejemplo, la desconexión de múltiples nodos, una bajada súbita en los usuarios, etc.

El interés por este tipo de sistemas es de naturaleza múltiple. En primer lugar, la red es un medio cambiante y de una utilidad que no requiere de justificación, por lo que es del interés general de sus usuarios conocer los diferentes eventos que influyen en su forma y capacidad para ayudar a tomar decisiones.

Del mismo modo, es de utilidad llegar a comprender cómo y porqué suceden estos eventos. El origen de estos revela información importante que a menudo trasciende la curiosidad técnica y nos permite obtener un prisma geopolítico de intereses variados. Un ejemplo claro son los episodios de censura a la libertad de expresión que son sufridos por los ciudadanos de regímenes incompatibles con la plena democracia.

Actualmente, los sistemas de alertas de eventos anómalos existentes exponen la información en la web o las distribuyen en listas de correo, además, se basan en un solo sistema de detección (o técnica) y, en muchos casos, no permiten que un usuario u organización replique dicho sistema de forma independiente y sencilla.

La labor de este proyecto es, por lo tanto, crear un sistema modular y escalable que permita la adopción de datos de distinta naturaleza, fácil extensión de nuevos algoritmos de detección a modo de complementos (*plugins*), canalización de alertas e información a través de medios que permitan interacción con el sistema.

El proyecto también aspira a democratizar la adopción de este tipo de sistemas, presentando una instalación simple y al alcance de distintos perfiles de usuario, además de una licencia de uso sin trabas comerciales, que permita ampliar el sistema y compartir esos avances con la comunidad de usuarios.

1.2 Objetivos del trabajo

Los objetivos pueden dividirse en dos aspectos conjuntos. Aquellos que son el resultado tangible del trabajo y los propuestos académicamente.

1.2.1 Objetivos materiales

Del propio contexto y justificación del trabajo ya se puede realizar un dibujo de los diferentes objetivos a alcanzar en el ámbito del proyecto. En líneas generales, el proyecto aspira a crear un sistema que comprenda los siguientes aspectos o hitos:

- Subsistema de captación que permita recolectar datos de fuentes externas de forma programática. Esto es, con una periodicidad basada en calendario que permita automatizar las captaciones.

- Subsistema de integración modular de algoritmos de detección de anomalías. Esto es, un sistema modular que nos permita integrar diferentes metodologías de detección de anomalías.

- Exposición de datos a través de cuadros de mando. El sistema poseerá un servicio de visualización de datos independiente del sistema. En dicho lugar se podrá obtener una representación de los datos en reposo que han sido procesados. Se proveerán varias vistas a modo de demostración, pero se podrá editar el cuadro de mando según las necesidades.

- Establecimiento de un canal de alertas e interacción con el sistema. Se tratará de un sistema que permita tanto la emisión de alertas, resúmenes de actividad, así como la comunicación bidireccional a través de comandos.

- Fácil instalación y uso. El sistema no deberá trasladar la complejidad de su integración al usuario. Se realizará una instalación sencilla, con el conjunto menor de dependencias posible y de uso fácil e intuitivo.

1.2.2 Objetivos académicos

Al margen de los entregables planificados, el objetivo de este trabajo es poner en práctica la ejecución de un proyecto basado en las materias cursadas durante el presente Máster, además de las ya cursadas durante el Grado de Ingeniería Informática.

Por lo tanto, los objetivos académicos serán:

- Estudio y comprensión de un problema o necesidad relacionada con el ámbito de los estudios.

- Identificación del estado del arte y la detección de las carencias existentes y oportunidades abiertas.

- Trazado en forma de fases de los objetivos para dar respuesta al problema propuesto.

- Elaboración de un marco temporal que permita la ejecución de las diferentes fases.

- Puesta en marcha del proyecto, seguimiento, detección y corrección de posibles riesgos que impidan la consecución de las diferentes fases

- Cierre del proyecto.

1.3 Enfoque y metodología empleadas

Examinados los objetivos y necesidades de estos, resultaba obvio que existían dos líneas de trabajo separadas en las que se tenía que invertir grandes cantidades de tiempo.

Una de las líneas eran los componentes que soportan la obtención, tratamiento, almacenamiento, recuperación, cuadro de mando y comunicaciones de alertas. La otra línea de trabajo, en la que ya se preveía un consumo destacado de tiempo era la investigación e implementación de algoritmos de detección de anomalías basados en series temporales.

Además, a todo ello, habría que sumarle el trabajo, por así decirlo, administrativo. Es decir, la documentación y confección de las PEC y entregas periódicas. Estos documentos deben reflejar con fidelidad los avances para una correcta tutorización, por lo que se cumplimentaron como una línea adicional y complementaria del proyecto con la importancia debida.

Transmitir de forma clara la información y con datos suficientes era otra meta por cumplir, por lo que se consideró que no era una línea de trabajo menor e iba a requerir dedicación adicional.

1.3.1 Metodología respecto a los componentes software

No se ha empleado una metodología formal (SCRUM, Agile, etc.) en el sentido estricto de marco de trabajo y coordinación de equipos, pero, si se ha empleado una metodología de software adecuada a las dimensiones del proyecto. En concreto, una aproximación al diseño basado en prototipos, sobre los cuales se iba añadiendo más funcionalidad y enlace con el resto de los componentes.

En primer lugar, se tenían claro los componentes del sistema y la comunicación entre estos a un alto nivel de abstracción. Esto dio lugar a un esquema conceptual en base a los diferentes requisitos que emanaban de la propia descripción del proyecto:

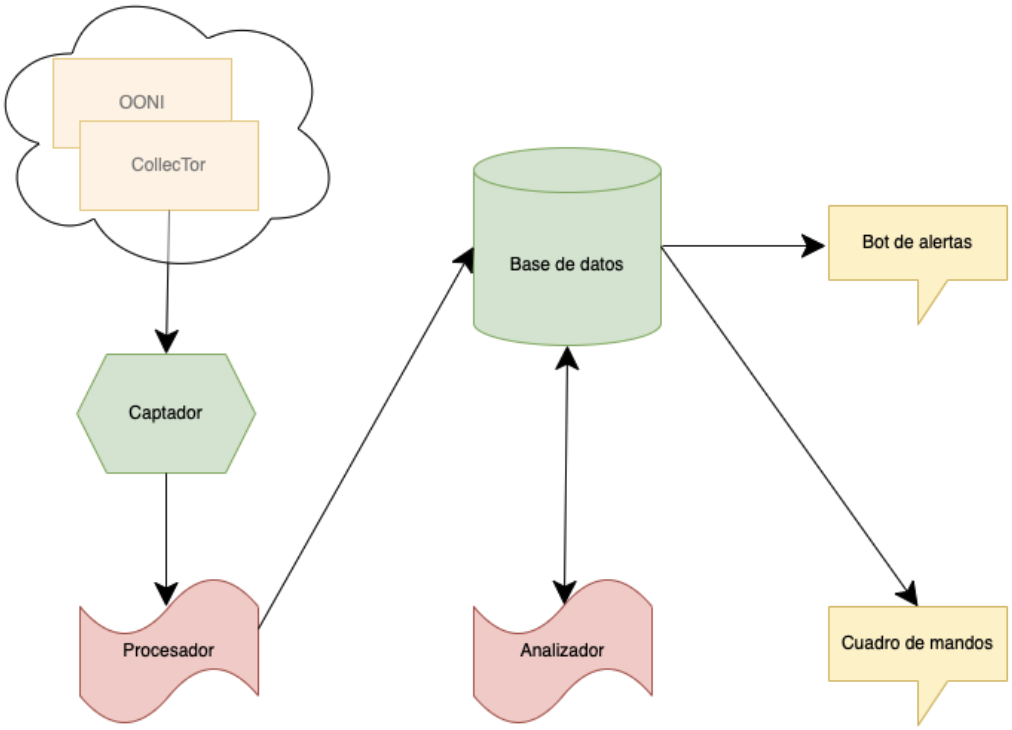


Ilustración 1 Diseño a alto nivel de los componentes

Obtenida la organización inicial de los componentes, se procedió al estudio de estos de manera independiente. Esto era así porque como componentes debían tener el menor acoplamiento posible en términos de funcionalidad.

Aunque la salida de uno es la entrada de otro, esa comunicación era fácilmente sustituible por datos sintéticos, para ir **efectuando prototipos iniciales**. Por lo que no representó un problema trabajarlos de forma aislada.

En una primera etapa se dejó el analizador para una etapa posterior (como se verá, dicho componente fue abordado más tarde, sobre la PEC-2), por lo que se volcó el desarrollo en los componentes de captación.

No obstante, antes de comenzar con el subsistema de captación, se procedió a estudiar las posibles fuentes de datos y su tipología, así como la adecuación a las necesidades del proyecto.

De esta fase, saldría el diseño inicial de la base de datos, sobre la que descansarán los datos en reposo una vez estos han sido depositados por el captador.

1.3.2 Metodología respecto a la investigación e implementación de algoritmos de detección de anomalías en series temporales de datos.

Respecto a esta parte, la metodología a seguir fue la investigación del estado del arte mediante la bibliografía existente, ensayos y artículos en su mayoría. El campo de investigación es amplio en el área de anomalías estadísticas, pero se estrecha si nos centramos en aquellos que lo hacen en series temporales y más concretamente en eventos producidos por tráfico de red.

Si bien, las anomalías de red son un rico campo de trabajo en ciberseguridad, las investigaciones se centran en detectar y detener ataques basados en la explotación de vulnerabilidades, tráfico achacable a malware, etc., siendo la base de este tipo de productos la implementación de reglas mediante un lenguaje.

No se trata, por lo tanto, de la misma casuística que buscamos en este trabajo, donde disponemos tan solo de estadísticas cuantitativas. Es decir, el **ámbito de nuestro trabajo es la detección de anomalías en la forma de la red y no en el contenido que pasa por esta**.

Una vez acotado el campo de trabajo, se identificaron múltiples trabajos previos de detección de anomalías en redes y más concretamente en la red Tor. Trabajos que más adelante, en la sección correspondiente, se citarán adecuadamente. Además, pueden verse relatados en la bibliografía empleada.

1.4 Planificación del trabajo

1.4.1 Medios empleados

El sistema se ha elaborado con el siguiente hardware detallado:

- Ordenador personal portátil Apple MacBook Pro
  - CPU Intel i5 2.3 Ghz, 4 núcleos
  - 8Gb RAM
  - 256 Gb SSD

Respecto al software empleado, se han usado los siguientes programas:

- Docker (sistema de contenedores)
- Git (gestor de versiones de código)
- Python 3.9 (lenguaje de programación)
- MySQL (base de datos)
- Grafana (cuadro de mandos)
- Redis (bróker mensajería)

Todos los servicios están integrados en tecnología de contenedores por lo que la única dependencia requerida es Docker.

Docker es un sistema de gestión de contenedores. Cada contenedor emplea una imagen, que es en si misma un conjunto de dependencias y configuración autosuficiente.

Por ello, no se precisa de la instalación en local de programas o librerías adicionales, permitiendo al usuario abstraerse de estas tareas y, a su vez, evitando polucionar su propio sistema operativo.

1.4.2 Resumen de las tareas a realizar

1. Diseño e implementación del captador de datos con disparo de tareas basada en calendario
2. Estudio de los datos obtenidos
3. Diseño del modelo de datos y elección de la base de datos
4. Diseño e implementación del procesador de datos
5. Diseño e implementación del módulo de análisis
6. Integración del cuadro de mandos
7. Elaboración del despliegue y producción de entregable

1.4.2.1 Diseño e implementación del captador de datos con disparo de tareas basada en calendario

Se trata de un módulo que permite la programación de tareas cronológicas de forma que se detecta la agregación de nuevos datos y su solicitud desde fuentes externas de forma periódica.

Este módulo visita las fuentes externas en busca de nuevos datos. Las visitas están ser lo suficientemente separadas entre si para evitar peticiones innecesarias, acoplándonos al calendario de publicación de la fuente externa.

1.4.2.2 Estudio de los datos obtenidos

Una vez seleccionadas las primeras fuentes externas, se estudiará la composición y formato de los datos.

Esta fase es fundamental, puesto que de aquí debe salir el modelo de datos que se empleará en todo el sistema.

El estudio de los datos obtenidos nos permitirá centrar la atención en aquellos que consideremos fundamentales para nuestros objetivos y descartar los que no nos aportan significación suficiente como para justificar su inclusión.

1.4.2.3 Diseño del modelo de datos y elección de la base de datos

Una vez estudiados los datos, se procede a diseñar el modelo de datos que va a ser empleado. Una vez obtenido este modelo, se discute y analiza el diseño de la base de datos; apropiada para el almacenamiento de los datos del modelo.

Dado que también debemos tener en cuenta elementos posteriores del sistema, será, con mucha probabilidad, una fase iterativa, en la cual la decisión de la elección de un motor u otro de base de datos vendrá condicionada por posteriores elecciones.

1.4.2.4 Diseño e implementación del procesador de datos

Se trata de un módulo de acoplamiento entre los datos externos entrantes y el modelo de datos que reposa en la base de datos.

Dicho sistema, simplemente, se encarga de procesar los datos para extraer los necesarios, enriquecerlos (si procede) y almacenarlos de vuelta en base de datos ya procesados.

Finalmente, disparará una señal para que el módulo de análisis pueda procesar la llegada de los nuevos datos.

1.4.2.5 Diseño e implementación del módulo de análisis

Este módulo es el que lleva el peso analítico de todo el sistema. Se encarga de procesar los datos introducidos en la base de datos y genera la correlación de los datos, detección de eventos y anomalías.

El diseño se ha efectuado para que pueda ser modular. Esto es, integrar diferentes implementaciones de algoritmos de detección de anomalías y poder tener una óptica distinta en relación con la salida de cada uno de ellos.

1.4.2.6 Integración del cuadro de mandos

El cuadro de mandos permite a las personas usuarias la visualización de los datos, eventos y anomalías detectadas en el análisis de datos.

Dado que existen soluciones de fuente abierta para dicha tarea, independiente de la fuente de datos, se emplea una solución basada en Grafana[1] que permite una alta configuración y adaptación a la consecución de los objetivos.

Esta fase también incluye el montaje del sistema de alertas y comunicación bidireccional con el sistema basadas en el sistema de mensajería Telegram[2].

1.4.2.7 Elaboración del despliegue y producción de entregable

[1] https://grafana.com/
[2] https://core.telegram.org/

Se trata de un sistema de construcción del proyecto y despliegue de fácil uso, en el que las posibles personas u organizaciones usuarias puedan poner en marcha el sistema sin complicaciones artificiales.

El proyecto está alojado en GitHub[3] como repositorio de código. Desde éste, será posible clonar el proyecto mediante una sola instrucción o comando del sistema (será necesario Git y Docker).

Una vez clonado el proyecto, se debería realizar el compilado (si procede) y despliegue, idealmente, con un conjunto reducido de órdenes. De manera qué, una vez finalizada esta fase, el sistema deberá encontrarse en funcionamiento.

1.4.3 Planificación temporal de los objetivos

A continuación, se expone un diagrama de Gantt como representación de las tareas, su interrelación, si cabe, y extensión cronológica.

La siguiente figura describe la cronología general de las entregas según el calendario académico:

Ilustración 2 Distribución según calendario académico

Una vez fijados los objetivos y teniendo en cuenta las demarcaciones cronológicas del calendario académico, se obtiene la planificación de las tareas:

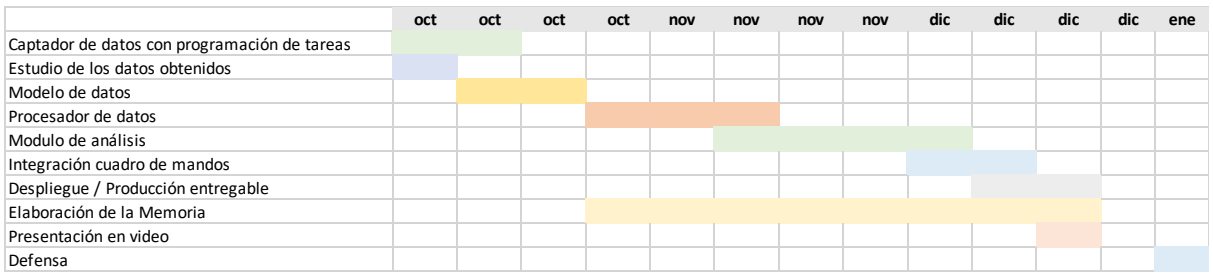


Ilustración 3 Distribución cronológica de las tareas

1.4.4 Relación con las PEC

[3] https://github.com/

A continuación, se detalla el desglose de tareas realizadas respecto de las diferentes entregas durante el curso.

1.4.4.1 PEC 1 - Planificación

Esta PEC, constituía el arranque del proyecto. Se estudió el problema a resolver, se consultó la bibliografía existente sobre el tema en cuestión y se analizó que huecos aun no estaban cubiertos en la problemática.

Una vez expuesto el problema y las posibles soluciones sobre el tablero, se definieron los objetivos principales, medios a emplear, enumeración de las tareas y una cronología de tiempos sincronizada con el resto de PECs a entregar.

Justo después de esta entrega se dio comienzo al estudio de los datos a obtener de las fuentes y los posibles modelos de datos a emplear en los componentes del sistema.

En esta PEC, se incluyo:

- Definición del problema a resolver.
- Objetivos.
- Metodología a emplear.
- Lista de tareas detalladas a emprender.
- Planificación de las tareas y entregas.
- Revisión del estado del arte.

1.4.4.2 PEC 2 – Entrega de seguimiento (I)

En esta entrega ya se dispuso de una versión preliminar del proyecto, el cual era capaz de obtener datos, procesarlos, componer el modelo de datos y visualizarlo en un cuadro de mandos (Grafana).

No obstante, el primer paso fue identificar las fuentes de datos, su posibilidad de uso y el formato que estas entregaban.

Afortunadamente, el sub-proyecto Metrics del proyecto Tor posee un rico sistema de publicación de datos de uso, telemetría y capacidad, debidamente anonimizados y actualizados.

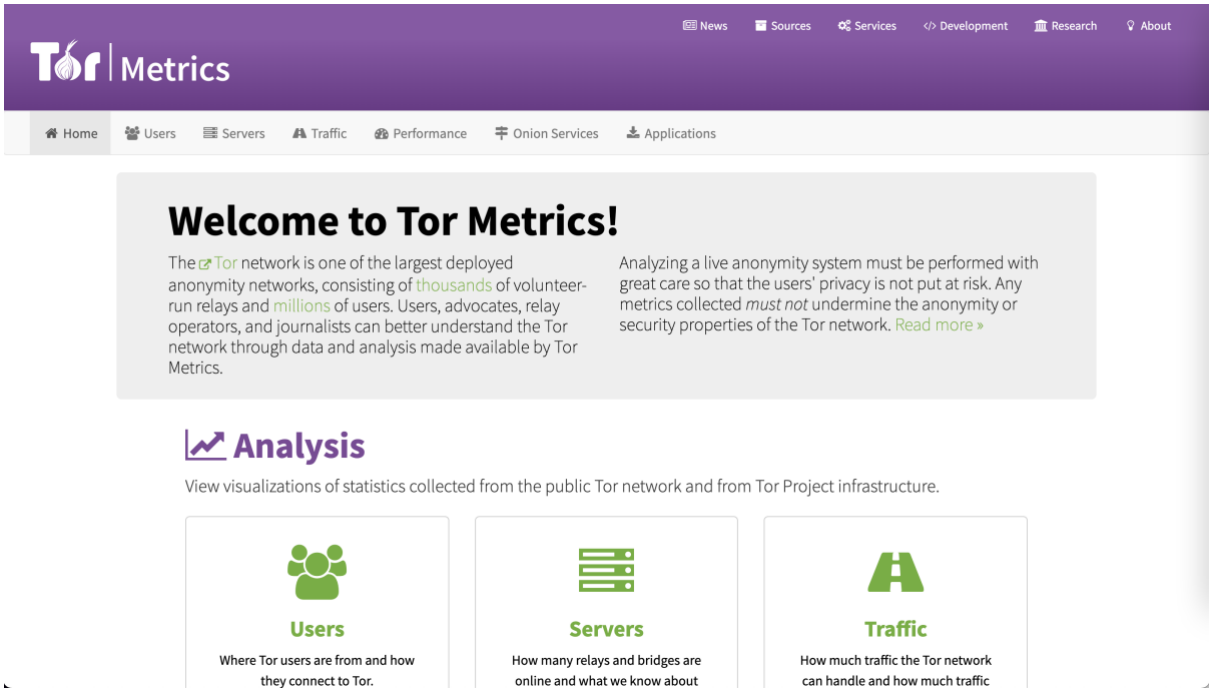


Ilustración 4 Detalle del portal de metrics.torproject.org

Esto, facilita enormemente la labor de captación de datos de uso, ya que es una fuente fiable, abierta y ética.

Programado el componente captador de datos y su procesamiento preliminar, estábamos en disposición de almacenarlos en base de datos de acuerdo con un modelo apropiado para el uso y consumo de estos posteriormente.

Tras ello, ya se disponía de un sistema que actualizaba los datos continuamente, por lo que se pasó a estudiar la viabilidad de monitorizar gráficamente estos datos a través del cuadro de mandos.

Dado que el cuadro de mandos sería implementado en Grafana (a través de un contenedor Docker), podíamos levantar el servicio y proveernos de una vista preliminar de algunas gráficas.

Ya en esta entrega, el cuadro de mandos poseía un aspecto cercano al definitivo:

Ilustración 5 Versión preliminar del cuadro de mandos

Esto, poseer un punto de visualización, fue de gran importancia puesto que servía de guía visual para acometer la siguiente PEC, en la que íbamos a necesitar retroalimentación para calibrar la sensibilidad del detector de anomalías.

En otras palabras: debíamos servirnos de alguna guía que nos identificase cuándo el actuador del algoritmo de detección debería, de forma razonable, proporcionarnos un positivo, tanto en la detección de anomalías negativa como positiva; entendiendo por detección, un súbito ascenso de usuarios o su contrario (señal de sospecha sobre posibles actos de censura).

En esta PEC, se incluyó:

- Componente captador de datos.
- Procesamiento de datos.
- Modelo de datos.
- Operaciones sobre base de datos.
- Vista preliminar del cuadro de mandos.
- Instrucciones de instalación.
- URL del repositorio con el código fuente del proyecto.

1.4.4.3 PEC 3 – Entrega de seguimiento (II)

Esta entrega estuvo centrada casi en su totalidad en la elección e implementación de los algoritmos de detección de anomalías.

La primera fase se dedicó a la recopilación de los algoritmos de detección publicados. Recordemos que no es el objetivo de este trabajo la realización de un algoritmo novedoso, sino de implementaciones de algoritmos conocidos y agregados de forma modular al sistema.

Dado que el objetivo era demostrar precisamente la modularidad del conjunto, y se necesitaba un módulo funcional, se trató de desarrollar en paralelo dos algoritmos, el de Wright [2] y el de Danezis [12]. Solo pudo incluirse este último en esta entrega, debido a la complejidad en la implementación del primero, que consumió un tiempo considerable sin resultado favorable.

Mientras que podíamos obtener resultados del módulo de anomalías a través de pruebas concretas:

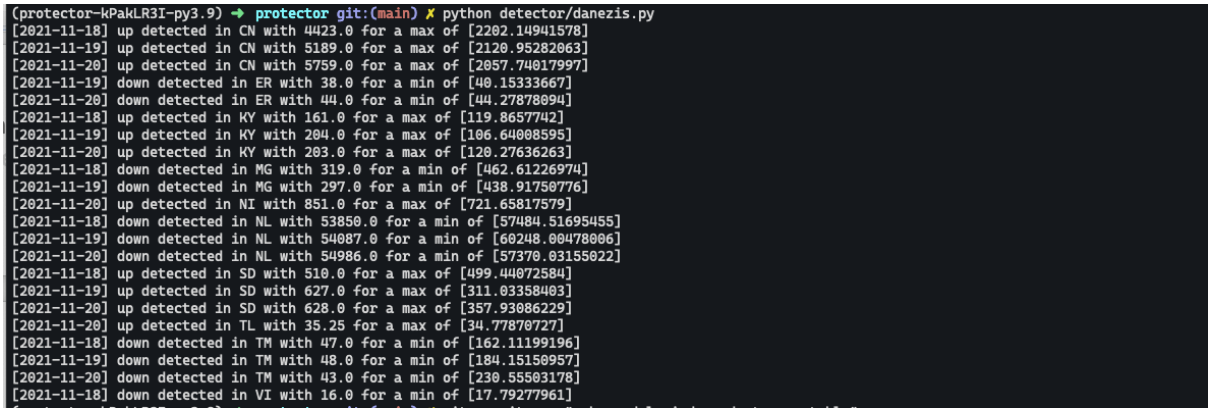


Ilustración 6 Salida del módulo de análisis

Se consideró la posibilidad de establecer en este punto el comienzo del componente de alerta y comunicación bidireccional, por lo que se montó un canal de comunicaciones basado en el servicio de mensajería Telegram. La vista preliminar de este componente ya ofrecía la posibilidad de satisfacer dos comandos que controlaban la existencia de alertas y estadísticas por país:

Ilustración 7 Vista preliminar del componente bot

En este punto, el sistema era prácticamente funcional. Se había demostrado la viabilidad de la arquitectura y la solución, a falta de dotarla de solidez y profesionalidad, respondía a los parámetros fijados en los objetivos.

En esta PEC, se incluyó:

- Módulo de análisis y alerta.
- Implementación del algoritmo de Danezis.
- Creación del bot de comunicaciones bidireccionales.
- Mejoras en todos los componentes entregados en la PEC-2

1.4.4.4 PEC 4 – Memoria final y entrega del producto

Aunque como ya se mencionó en la planificación, no se quiso dejar la redacción de la memoria para el último sprint, sin lugar a duda, es en esta entrega donde se consume más tiempo en su confección; lógicamente, debido a que es cuando poseemos un resultado sustancial sobre el cual podemos plasmar en texto.

No obstante, se empleó tiempo en el desarrollo, para concluir con la cohesión de todos los componentes, esta vez orquestados por el gestor de colas de trabajo Celery, con la idea de que el sistema eche a andar con un solo comando.

Además, se pulieron partes del código fuente que eran parte de los prototipos y que fueron cambiadas o eliminadas para la versión final.

Otro punto de mejora fue la emisión de alertas para incluir símbolos gráficos (emojis) para una mejor comprensión desde el punto de vista del usuario.

En esta PEC, se incluyó:

- Memoria del proyecto.
- Versión final del sistema.

1.5 Análisis de riesgos

A continuación, se enumeran una serie de riesgos previsibles e identificados que podrían afectar a la continuidad del proyecto e incluso a su viabilidad. Es importante tener en cuenta, no solo su anticipación, sino la capacidad para detectar la existencia de ese riesgo (o cualquier otro no identificado aquí) y poseer capacidad de reacción suficiente para minimizar el impacto y sus consecuencias.

1.5.1 Objetivos con sobrecoste en recursos

Es probable que sea un riesgo común a todos los proyectos del cualquier magnitud y complejidad. Ya sea por desconocimiento del campo o infravaloración del esfuerzo necesario a invertir, se establecen unos objetivos inabarcables en el tiempo programado para obtener resultados mínimamente viables.

Este riesgo puede afectar a uno, varios o todos los objetivos del proyecto.

Mitigación:

En primer lugar, deberíamos preguntarnos: ¿Qué recursos son necesarios para llegar a una finalización correcta, es decir, al objetivo acabado? Si prevemos que el objetivo aun requiere recursos y tiempo que someterían al mismo riesgo a otros objetivos, deberíamos evaluar si podemos permitir ese trasvase de recursos de un objetivo a otro.

En caso positivo, es decir, es posible tomar tiempo y recursos de otro objetivo se deberá reajustar el reparto de recursos y tiempos de forma que refleje estos cambios.

En caso contrario: no podemos dedicar otros recursos y es inviable, se debe reajustar el propio objetivo valorando que hemos conseguido hasta ahora y como podemos aprovecharlo para generar una nueva meta. Es decir, un nuevo objetivo.

Este riesgo se materializó en la realización del espacio de tiempo correspondiente a la PEC-3. En concreto, durante la implementación de los algoritmos de detección de anomalías.

El algoritmo de Wright supuso una inversión de tiempo demasiado prolongada debido a la complejidad del algoritmo y puso en riesgo la meta de realizar una entrega de

seguimiento exitosa. Al final, se consideró alternar la implementación por la de Danezis, el cual si se realizó y probó con éxito.

1.5.2 Problemas de integración entre componentes

En un proyecto de esta naturaleza, con múltiples componentes que han de entenderse entre sí, es posible encontrar inconsistencias e incompatibilidades entre ellos que impidan un acoplamiento adecuado e incluso posible.

El principal riesgo es apostar por una tecnología que posteriormente se demuestra incapaz de conectarse con otras ya adoptadas. Esto, puede ocasionar pérdida de tiempo y recursos ya que se invierten en un componente que posteriormente se ha de sustituir o que requiere de costes sobrevenidos para paliar o minimizar el riesgo.

Mitigación:

Si el componente no se acopla bien con el resto o con otro componente, debemos preguntarnos si existe un medio que provea de esta capacidad y actúe a modo de intermediario o interlocutor (proxy) entre ellos.

Si no existe o no podemos integrarlo (falta de recursos, coste financiero, etc.) ¿Podemos sustituir alguno de los componentes implicados? Supongamos que tenemos los componentes A, B y C. A y B pueden hablar con C, pero no entre A y B. El coste de sustituir A es el mismo que el de sustituir B+C por D, pero sustituir A es costoso en tiempo, por ejemplo.

En el caso anterior, debemos sopesar que operación nos interesa más. Al final, deberemos efectuar una sustitución y está deberá regirse por los costes principales y asociados.

Durante la realización de la PEC2 surgió esta eventualidad debido a que la base de datos inicial seleccionada, MongoDB, no era una opción viable para el rescate de datos de Grafana (componente de visualización).

La solución era adquirir un producto comercial con una licencia restrictiva y un elevado precio. Dado que era un problema agudo y los costes eran inasumibles se evaluó el cambio de base de datos a una soportada por todos los componentes. Finalmente, se seleccionó MySQL; alternativa también gratuita y con licencia open source.

El cambio de base de datos requirió cambios en el código fuente, pero gracias a la arquitectura (que se comentará en el capítulo adecuado) fue soportable y pudo ejecutarse dentro de calendario.

1.6 Sumario de los productos obtenidos

Se ha obtenido un sistema de componentes heterogéneos que coordinados entre si, permiten obtener datos del estado de la red Tor, constituir un modelo en base a dichos

datos, detectar anomalías, publicar un cuadro de mandos y desplegar un agente (o bot) para comunicación humada-máquina y propagación de alertas.

El resultante se ha liberado en un repositorio público de código fuente y está disponible bajo una licencia de fuente abierta.

1.7 Descripción de los capítulos de la memoria

1.7.1 Capítulo 2 (Declaración de principios éticos en el uso de datos)

Se trata de una declaración de principios acerca de la ética y respeto al uso de datos.

1.7.2 Capítulo 3 (Descripción de los datos de ingesta)

Describe que datos se van a obtener y procesar, así como su origen y significado dentro del proyecto. Responde a preguntas de porqué un dato y de que nos puede ser útil.

1.7.3 Capítulo 4 (Captador de datos)

Corresponde al componente dedicado a la ingesta. Se describe su funcionamiento y diseño, así como el proceso de ingesta hacia la base de datos.

1.7.4 Capítulo 5 (Modelos de datos)

Se describe la elección de la base de datos y composición o diseño del esquema empleado para recoger los datos de la ingesta procedentes del captador.

1.7.5 Capítulo 6 (Cuadro de mandos)

Este capítulo nos presenta el componente de cuadro de mandos, encargado de la presentación y monitorización de los datos que van siendo ingestados en la base de datos.

1.7.6 Capítulo 7 (Analizador modular de algoritmos de detección de anomalías)

Se describe el diseño del componente encargado de recibir los nuevos datos procedentes del captador (previo procesado) y enviarlos a los diferentes módulos de detección.

También se encarga de producir datos de alerta que serán consumidos por el bot, encargado de emitir los mensajes de interés hacia los subscriptores.

1.7.7 Capítulo 8 (Bot y emisión de alertas)

Se presenta el bot, un mecanismo o artefacto que dispone de un canal de comunicaciones bidireccionales con las personas usuarias.

El bot se encarga de monitorizar la existencia de alertas respecto del módulo de detección, elaborar los mensajes y emitirlos hacia los subscriptores.

Al posee un canal bidireccional, las personas usuarias pueden enviar comandos para que el bot los tramite y devuelva una respuesta.

1.7.8 Capítulo 9 (Conclusión y futuros trabajos)

Se cierra el proyecto con las conclusiones, valoraciones y las posibilidades que ofrece respecto a su explotación, así como diversas tareas que se proponen para darle continuidad y expansión.

2 Declaración de principios éticos en el uso de datos

El uso de los datos en este trabajo se adhiere a lo establecido en el uso ético de los datos por el proyecto Tor:

https://blog.torproject.org/ethical-tor-research-guidelines

Así mismo, se quiere dejar constancia y citar la fuente de los datos indicada para ello, tanto aquí como en el apartado de bibliografía:

- **How Much Anonymity does Network Latency Leak?** (**PDF**) (Cached: **PDF**) by Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin.
  *In ACM Transactions on Information and System Security **13**(2), February 2010.* (**BibTeX entry**)**.**

3 Descripción de los datos de ingesta

La detección de anomalías posee como necesitad perentoria la obtención de datos de las fuentes apropiadas. En nuestro caso, el proyecto Tor publica una gran cantidad de datos ya estructurados y categorizados de forma periódica. Adicionalmente, es posible descargar el histórico completo de cada una de las fuentes de datos desde que existe registro.

La fuente principal de datos de Tor está recogida en la siguiente dirección web, correspondiente al proyecto 'metrics':

https://metrics.torproject.org/

Desde donde es posible tanto efectuar la descarga[4] de los distintos conjuntos de datos, como visionar gráficos relativos a estos.

Las categorías principales son:

- **Users**. Enfocado al número de usuarios de la red Tor, por países, tipo de conexión, etc.
- **Servers**. Ilustran o detallan la composición de la red Tor.
- **Traffic**. Nos indica la cantidad de tráfico, ancho de banda anunciado, etc.
- **Performance**. Métricas relacionadas con el rendimiento de la red Tor.
- **Onion Services**. Como indica su nombre, número estimado de servicios por versión del protocolo.
- **Applications**. Se refiere a las estadísticas relativas a las descargas y actualizaciones de las aplicaciones Tor Browser y Tor Messenger.

3.1 Elección de las fuentes y justificación

3.1.1 Usuarios

Una de las principales categorías de datos es la de **usuarios de la red** particionada por países de origen. Es una fuente perfecta para indicarnos localmente (por país) como se está comportando la población de un determinado país.

Si, por ejemplo, el uso de Tor en un país aumenta drásticamente, podría ser un indicio (no significa que directamente sea una causa) de que existen prácticas totalitaristas (por ejemplo, romper el secreto de las comunicaciones de los ciudadanos) o estas comienzan a ser una preocupación por parte de la población.

[4] https://metrics.torproject.org/stats.html

Por el contrario, un descenso agudo del número de usuarios de un país puede deberse a una técnica de censura que impida el acceso a la red Tor e incluso al propio Internet.

Obviamente, hablamos de circunstancias que posteriormente se deben comprobar de forma fehaciente, puesto que hablamos de indicios y no de hechos contrastados.

Estos datos son tomados de (Relays) y (Bridges):

https://metrics.torproject.org/userstats-relay-country.csv
https://metrics.torproject.org/userstats-bridge-country.csv

3.1.2 Tamaño de la red

Muy relacionada con la métrica anterior, añadimos los datos relacionados con la **descarga del Tor Browser por lenguaje** (o localización). Esto podría indicarnos si un crecimiento súbito en las descargas de un idioma coincide con un evento político. La métrica es obtenida desde:

https://metrics.torproject.org/webstats-tb-locale.csv

A continuación, tomamos datos de varías métricas que nos indican el tamaño y ancho de banda disponible en la red Tor:

- Ancho de banda:

https://metrics.torproject.org/bandwidth.csv

- Tamaño de la red, en número de Relays y Briges:

https://metrics.torproject.org/networksize.csv

- Número de Hidden services

https://metrics.torproject.org/hidserv-rend-relayed-cells.csv

Con estas métricas obtenemos una idea de la **capacidad neta de la red Tor**. Por supuesto, dado que los diferentes componentes de la red Tor poseen una dirección (IP) pública, están expuestos a ataques y censura, por lo que es de suma importancia monitorizar el tamaño de la red y medir sus contracciones (disminución atípica) y expansiones en tamaño.

3.1.3 Rendimiento de la red

Con el último grupo, disponemos de los datos de sensores y herramientas de rendimiento que nos permiten tomarle el pulso a la red Tor:

Tiempo de descarga de un archivo:

https://metrics.torproject.org/torperf.csv

Fallos en la descarga de archivos:

https://metrics.torproject.org/torperf-failures.csv

Latencia o Round-trip Time dentro del circuito:

https://metrics.torproject.org/onionperf-latencies.csv

Tiempo de establecimiento de un circuito:

https://metrics.torproject.org/onionperf-buildtimes.csv

Throughput (Nivel de rendimiento)

https://metrics.torproject.org/onionperf-throughpu.csv

Con estas tres categorías disponemos de datos que nos deberían permitirían, a priori, detectar ciertos eventos anómalos que podrían relacionarse con:

- Aumento de la represión en países
    - Por ejemplo, un aumento repentino de descargas de Tor Browser podría indicar que la ciudadanía teme por el secreto de sus comunicaciones.
- Censura
    - Un descenso acusado de los usuarios de Tor de un país determinado podría ser causa de un ataque de fuente gubernamental que impida conectar a la ciudadanía con los relays de Tor.
- Ataques directos a la red Tor
    - La propia red Tor podría ser víctima de ataques premeditados.
- Ataques indirectos a Internet que afectan el rendimiento o volumen de la red Tor
    - No solo la propia red, de forma indirecta, un ataque podría causar un deterioro en el rendimiento de la red Tor; dado que esta posee su infraestructura asociada a la de Internet.

3.2 Alternativas y motivación de la elección

OONI es observatorio de la censura en Internet, no solo sobre la red Tor, sino que también contempla otros protocolos y aplicaciones de comunicación. Posee una excelente API bien documentada y disponible para su consulta.

https://ooni.org/data/

No se empleo OONI dado que tenemos disponibles los datos de Tor de forma directa en un formato adecuado para el procesamiento en lote y centrados en Tor de forma exclusiva.

No obstante, hubiese sido un proveedor de datos óptimo si el presente proyecto hubiese tenido en cuenta más tipos de aplicaciones de comunicación.

Adicionalmente a 'metrics', Tor también cuenta con el sub-proyecto 'onionoo', el cual posee una API para obtener datos sobre bridges y relays de forma histórica.

Está disponible desde: https://metrics.torproject.org/onionoo.html

Tampoco se empleo puesto que 'metrics' posee, como se ha dicho anteriormente, un formato apropiado para la ingesta y no necesitamos realizar múltiples llamadas a la API para captar estos datos.

La elección de la fuente directa de datos frente a las alternativas era bastante clara desde el principio y cae por peso propio las ventajas frente a las alternativas: Se centran en Tor, el formato es adecuado y la petición de datos es directa (descargar un archivo).

4 Captador de datos

En resumen, el captador de datos es un servicio que corre continuamente, pero solo ejerce su función una vez cada 24 horas (aunque el periodo puede ser cambiado según la necesidad), permaneciendo dormido el resto del tiempo.

Es una tarea insertada en el gestor de colas de trabajo Celery, el cual, mediante un calendario (scheduler) despertará a la tarea para que compruebe si existen nuevos datos y descargue las actualizaciones.

4.1 Descarga parcial de datos

El captador no descarga los archivos en su totalidad. 'metrics' provee de dos parámetros en su URL que permiten indicar un espacio de tiempo concreto entre dos fechas.

Esto podemos verlo claramente en las líneas 41 y 44 del código fuente del archivo donde está implementada la mayoría de funcionalidad del captador:

```
31  def fetch():
32      check_tables()
33      for model in modules.names:
34          logger.info("Processing stats for model %s", model.__name__)
35          last = model.last()
36
37          if last:
38              past = last["date"]
39              # If we've got a last record then download +1 day past last record
40              past = past + relativedelta(days=+1)
41              from_date = "{}-{}-{}".format(past.year, past.month, past.day)
42
43              now = today()
44              to_date = "{}-{}-{}".format(now.year, now.month, now.day)
45
46              logger.info(
47                  "Downloading stats file for %s from %s to %s",
48                  model.__name__,
49                  from_date,
50                  to_date,
51              )
52              c = download(model.URL.format(from_date, to_date))
```

Ilustración 8 Detalle de la parametrización de segmentos cronológicos

Además, está implementada para todas las fuentes de datos de 'metrics', ya que estas comparten esos dos parámetros. Gracias a esto, podemos abstraernos en la descarga de cada fuente de datos (o endpoint) y utilizar una función genérica para todas ellas.

¿Qué ganamos con esto?: Pedir solamente la franja de tiempo que necesitamos y que irá desde el último día que descargamos, y que está registrado en base de datos más uno, al día actual. De esta forma, evitamos solicitar todo el archivo completo al servidor; con el consiguiente ahorro en ancho de banda.

4.2 Abstracción del método de procesamiento de datos por fuente

Una vez el archivo de cada fuente es descargado, el módulo captador llama a un método personalizado donde el modelo es el encargado de realizar un cribado de los datos, trasformación y limpieza y finalmente, almacenamiento en la base de datos.

El captador se abstrae de dichas funciones y llama de forma genérica al método o función 'purify' del modelo (línea 58):

```
56              c = download(model.URL.split('?')[0])
57
58          c = model.purify(c)      You, 2 months ago • stats now supports list of models to download and…
59          if not c:
60              logger.info("Nothing to update on %s", model.__name__)
61          else:
62              entries = [model.get_class()(*entry) for entry in c]
63              model.ingest(entries)
64
```

Ilustración 9 Llamada al método 'purify' de tratamiento de datos

Cada modelo sí portará su propia implementación del procesamiento de datos. Apropiada a cada tipo de archivo y contenido asociado. Además, los modelos se abstraen de dicho procesamiento, que está implementado a un nivel superior y

simplemente se dedican a inyectar la parametrización adecuada a su caso (en el ejemplo ilustrado, parte del modelo ‘user’):

```
58  def purify(entries: list[str]) → list[list[str]]:
59      return meta.purify(entries=entries, fields=6, banned=["date", "#"])
```

Ilustración 10 Implementación de la llamada a 'meta.purify' de forma paramétrica

Aquí, en ‘fetch/models/meta.py’, es donde radica la implementación real del método, que procesará la información acorde a la parametrización adecuada.

```
61  def purify(entries: list[str], fields: int, banned: list[str]) → list[list[str]]:
62      tmp = []
63
64      for entry in entries:
65
66          # Skip comment lines
67          if any([entry.startswith(ban_word) for ban_word in banned]):
68              continue
69
70          # Split str into list of params
71          entry = entry.split(",")
72
73          # Skip entries with less than ',' params
74          if not len(entry) == fields:
75              continue
76
77          tmp.append(entry)
78
79      return tmp
```

Ilustración 11 Implementación de 'meta.purify'

Lo que se ha efectuado aquí es una abstracción de las tareas de ingesta y procesamiento, tomando todo aquello que es común a los diferentes conjuntos de datos.

Luego, los modelos podrán actuar sobre el comportamiento de la función afectándola con la parametrización, como ya hemos comentado, adecuada a su proceso.

Al final, queda un código simple, sencillo, genérico y parametrizable que nos permitirá ingestar todos los datos con pocos pasos.

4.3 Ingesta de datos procedentes del captador

De modo similar al procesamiento de datos, la ingesta también posee elementos comunes entre las distintas fuentes, por lo que el diseño es similar en las formas que el visto anteriormente.

Se dispone de un método ‘ingest’ por cada modelo, pero también se encuentra abstraído de forma genérica en ‘meta’.

```
            else:
                entries = [model.get_class()(*entry) for entry in c]
                model.ingest(entries)

```

Ilustración 12 Llamada al método genérico 'ingest'

Luego, en cada modelo (en el ejemplo, ‘users.py’):

```
def ingest(entries: list[UsersEntry]):
    return meta.ingest(
        entries=entries,
        table_name=COLLECTION,
    )
```

Ilustración 13 Llamada al método 'ingest' en el modelo 'users'

y finalmente, la implementación propia en ‘fetch/models/meta.py’:

```
def ingest(entries: list, table_name: str) -> None:
    try:
        validated = [e.serialize() for e in entries if e.validate()]
        length = len(validated)
        inserted = 0

        con = db.connect()
        cur = con.cursor()

        if length > MAX_SIZE:
            logger.info("Due to ingest size commits will be batched")
            chunk_size = round(length / CHUNKS)

            for index in range(0, CHUNKS):
                if index == CHUNKS - 1:
                    shift = None
                else:
                    shift = index * chunk_size + chunk_size

                chunk = validated[index * chunk_size : shift]

                cur.executemany(
                    db.insertions[table_name],
                    chunk,
                )

                inserted += cur.rowcount
                con.commit()
        else:
            cur.executemany(db.insertions[table_name], validated)
            inserted += cur.rowcount
            con.commit()
```

Ilustración 14 Implementación completa de la ingesta

Huelga decir que la ingesta tiene dos modos, el modo normal en el que cada nuevo fragmento temporal es asimilado y el modo por lotes que ingestará el equivalente a un archivo completo, siendo este último el modo más probable en la primera ejecución, cuando no se posee dato alguno.

4.4 Ventajas del diseño modular

La ventaja de este mecanismo de funciones proxy es que el módulo principal de captación de datos se abstrae de como han de tratarse los datos y como deben ser almacenados una vez se han procesado.

De este modo, al estar orientado a módulos, incluir una nueva fuente de datos es trivial. Simplemente introduciríamos un nuevo módulo que integre las funciones ‘purify’ e ‘ingest’ que necesite y se daría de alta en el módulo de filiación ‘fetch/models/modules.py’:

Ilustración 15 Detalle del módulo de filiación

Una vez recogido aquí, pasaría a formar parte del resto de módulos y su programación temporal. El código resultante es compacto, limpio y de bajo acoplamiento.

4.5 Alternativas y motivación de la elección

Este sistema también puede ser implementado con el sistema ‘cron’ típico de los sistemas operativos de la familia UNIX.

Simplemente, se programaría una tarea ‘cron’ que cuando se disparase ejecute un script que baje el archivo, lo procese y lo inserte en base de datos.

Aunque en apariencia más sencillo, este diseño despega este componente del sistema y lo desplaza fuera, hacia el sistema operativo, aislándolo demasiado del conjunto y perdiendo cohesión con el resto.

Además, cada vez que se inserta un nuevo módulo se debería programar una tarea ‘cron’ o indicar a un script que identifique los nuevos módulos y los ejecute, algo que podría incluso ser problemático y crear una vulnerabilidad en el sistema.

El sistema de programación de tareas Celery está perfectamente integrado en la solución al estar escrito en Python, lo cual permite utilizar el componente como una librería propia del sistema.

5 Modelo de datos

5.1 Aproximación simplista al modelo de datos

El modelo de datos descansa sobre una base de datos relacional. En concreto, MySQL.

El diseño es sencillo y directo, replica 1:1 el formato de salida de los archivos generados por la fuente de los datos, 'metrics'. Tan solo, en la fase de limpieza de estos datos (vista anteriormente), se adecua al tipo de dato en reposo, esto es, el tipo SQL adoptado por MySQL.

Las tablas no poseen relación entre estas, por lo que el diseño se simplifica al máximo. No obstante, no es necesario complicarlo. El sistema aspira a ser lineal desde su concepción: entran nuevos datos, se procesan, pasan al módulo de análisis y se emiten alertas.

Si observamos la cabecera de uno de los archivos procedentes de la fuente de datos 'metrics':

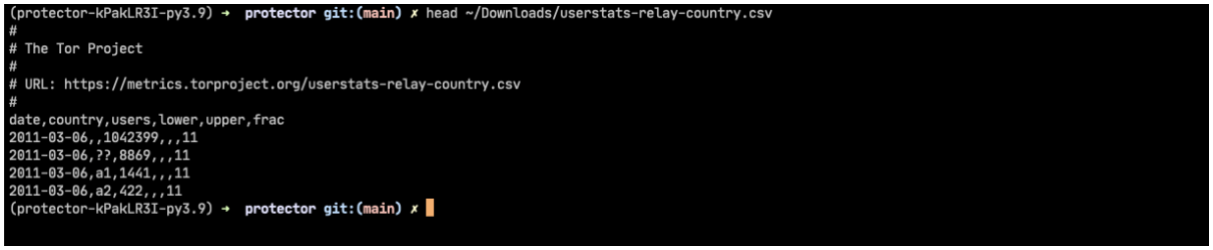
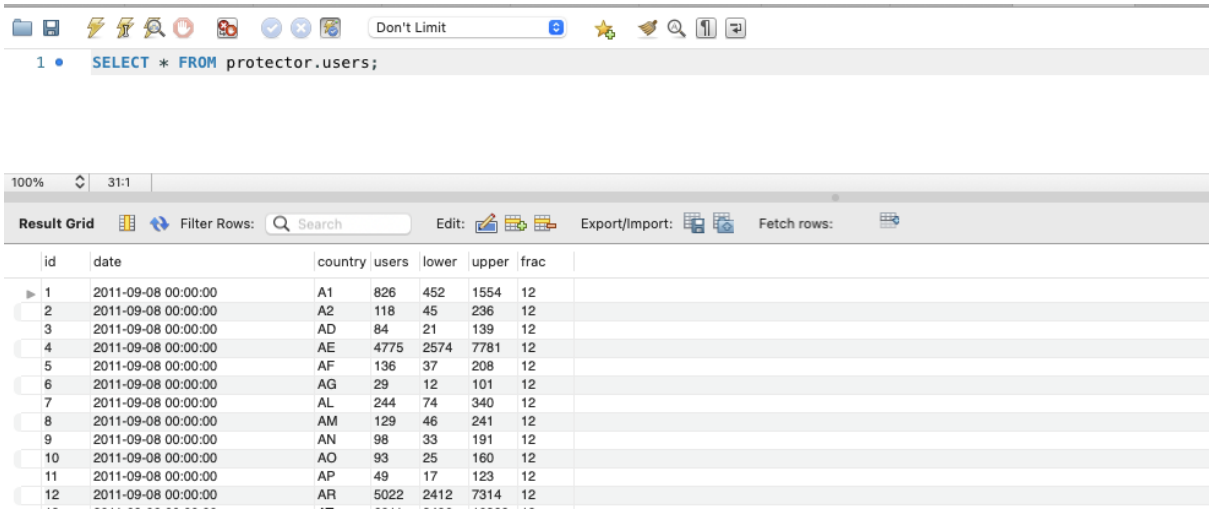


Ilustración 16 Detalle de la cabecera de un archivo procedente de la fuente de datos en 'metrics'

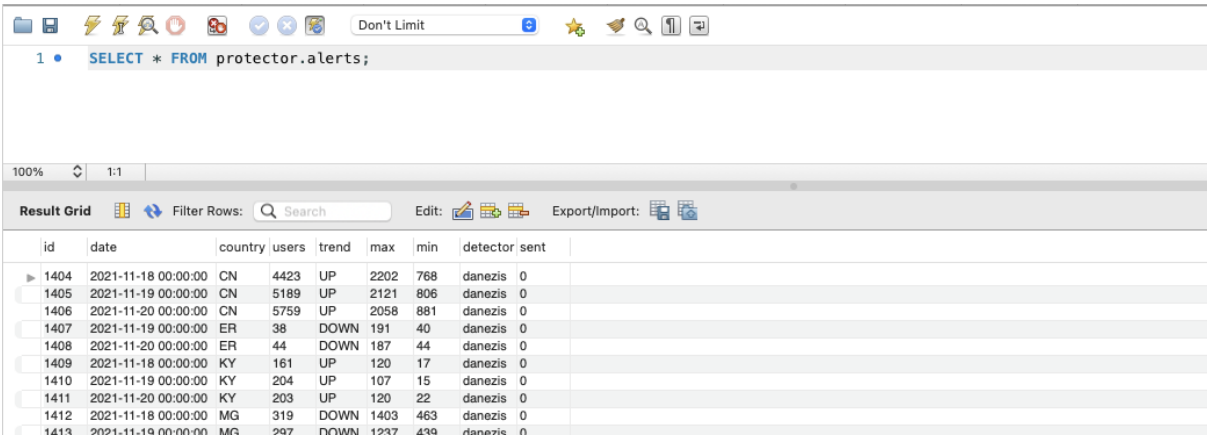
La tabla SQL correspondiente es casi una traducción directa de este formato:



id	date	country	users	lower	upper	frac
1	2011-09-08 00:00:00	A1	826	452	1554	12
2	2011-09-08 00:00:00	A2	118	45	236	12
3	2011-09-08 00:00:00	AD	84	21	139	12
4	2011-09-08 00:00:00	AE	4775	2574	7781	12
5	2011-09-08 00:00:00	AF	136	37	208	12
6	2011-09-08 00:00:00	AG	29	12	101	12
7	2011-09-08 00:00:00	AL	244	74	340	12
8	2011-09-08 00:00:00	AM	129	46	241	12
9	2011-09-08 00:00:00	AN	98	33	191	12
10	2011-09-08 00:00:00	AO	93	25	160	12
11	2011-09-08 00:00:00	AP	49	17	123	12
12	2011-09-08 00:00:00	AR	5022	2412	7314	12

Ilustración 17 Detalle de la tabla correspondiente al modelo 'users'

Tan solo la tabla de alertas es una adopción propia del sistema para su desempeño, el cual deriva de la salida del módulo de análisis y registra que alertas han sido ya emitidas para evitar repetición:



id	date	country	users	trend	max	min	detector	sent
1404	2021-11-18 00:00:00	CN	4423	UP	2202	768	danezis	0
1405	2021-11-19 00:00:00	CN	5189	UP	2121	806	danezis	0
1406	2021-11-20 00:00:00	CN	5759	UP	2058	881	danezis	0
1407	2021-11-19 00:00:00	ER	38	DOWN	191	40	danezis	0
1408	2021-11-20 00:00:00	ER	44	DOWN	187	44	danezis	0
1409	2021-11-18 00:00:00	KY	161	UP	120	17	danezis	0
1410	2021-11-19 00:00:00	KY	204	UP	107	15	danezis	0
1411	2021-11-20 00:00:00	KY	203	UP	120	22	danezis	0
1412	2021-11-18 00:00:00	MG	319	DOWN	1403	463	danezis	0
1413	2021-11-19 00:00:00	MG	297	DOWN	1237	439	danezis	0

Ilustración 18 Detalle de la tabla 'alerts'

5.2 Alternativas y motivación de la elección

Como alternativa, en principio el modelo descansaba sobre una base de datos no relacional (NoSQL) orientada a documentos: MongoDB[5]. Sin embargo, debido a incompatibilidad con otros componentes (se describe en la sección de riesgos) se tuvo que migrar a la actual MySQL[6].

Otra opción hubiese sido (más simple si cabe) trabajar directamente con archivos CSV en depósito y serializarlos cuando hubiese sido necesario trabajar con ellos.

Esta vía “fácil” elimina un componente costoso en el sentido de los recursos máquina necesarios (recursos computacionales): la base de datos, pero hace perder toda plasticidad con el trabajo de los datos y más si cabe en futuras ampliaciones, por no mencionar la pérdida en rendimiento cuando el tamaño de los archivos va ganando peso.

Otra opción más hubiese sido trabajar con ‘sqlite[7]’, base de datos incluida de serie en la biblioteca estándar de Python. No obstante, perderíamos potencia si en un futuro decidiéramos aumentar la complejidad del sistema con nuevos módulos. La propia naturaleza de ‘sqlite’ es de ser una base de datos sencilla, apropiada para usos internos de una aplicación autosuficiente.

De hecho, al residir la base de datos ‘sqlite’ en un archivo dentro del contenedor, tendríamos que acoplar un sistema que permitiera el acceso a estos datos por parte de otros módulos que estarían en módulos independientes.

[5] https://www.mongodb.com/es
[6] https://www.mysql.com/
[7] https://www.sqlite.org/index.html

MySQL nos permitía trabajar de forma sencilla con tablas carentes de complicación y además, tenemos la posibilidad de crear una base robusta preparada para posibles ampliaciones.

6 Cuadro de mandos

6.1 Despliegue del componente

El cuadro de mandos proviene de Grafana. Este es un componente independiente, que absorbe cualquier tipo de datos previa definición del conector apropiado y la definición de la estructura de datos con la que trabajar.

Grafana es una solución que solo precisa de un despliegue y definición de la conexión de datos. Una vez desplegado, tan solo queda definir los distintos cuadros de mandos asociados a los datos.

Dado que Grafana se despliega a través de un contenedor no es necesario realizar ningún tipo de instalación o configuración. Es más, se han definido ya varios cuadros de mandos para que se pueda visualizar la información desde el comienzo de la instalación.

```
13  grafana:
14    image: grafana/grafana:8.2.1
15    container_name: protector_grafana
16    ports:
17      - 3000:3000
18    volumes:
19      - ./grafana_data:/var/lib/grafana
20
```

Ilustración 19 Detalle de la configuración Docker de Grafana

6.2 Definición de los cuadros de mando

Un cuadro de mando expone de forma visualmente gráfica la información que reposa en base de datos. Es una forma de exponer los datos para favorecer el análisis visual y rápido.

Un ejemplo, procedente de uno de los cuadros de mandos que porta el sistema base:

Ilustración 20 Cuadro de mandos general

Como podemos observar en la ilustración, vemos datos directamente de servicios Onion, ancho de banda de la red Tor, tiempo de construcción de los circuitos, etc.

De este modo podemos obtener una visual inmediata de lo que está pasando (o ha pasado) en la red Tor de acuerdo con los datos disponibles.

Además, los cuadros de mando se actualizan (o refrescan) por un intervalo de tiempo programable, por lo que podemos desplegarlos y mantener una monitorización casi en tiempo real del estado de la red.

6.3 Alternativas y motivación de la elección

Una alternativa a Grafana[8] es Kibana[9], proyecto similar que ofrece prestaciones parecidas. No obstante, Kibana es un proyecto asociado a la pila tecnológica de ELK (Elasticsearch, Logstash y Kibana), por lo que su fuente de datos común suele ser Elasticsearch, aunque es posible utilizarla con otras fuentes.

La elección de uno u otro proyecto no posee evidentes ventajas técnicas que podamos asociar a factores objetivos. Grafana fue seleccionado por poseer más independencia de fuentes y estar asociado a un mejor tratamiento en series temporales y los componentes gráficos disponibles.

[8] https://grafana.com/
[9] https://www.elastic.co/es/kibana/

7 Analizador modular de algoritmos de detección de anomalías

7.1 Diseño modular

Desde el principio se quiso crear un analizador múltiple, que tuviese la posibilidad de aglutinar varios módulos con diferentes implementaciones de algoritmos de este tipo.

Tener una implementación única de detección demuestra la tecnología, pero disponer de varias detecciones enriquece la capacidad de tomar decisiones o de afinar la ventana de anomalías y centrarse a investigar las líneas más prometedoras.

Debido al tiempo disponible y la complejidad de las implementaciones, solo se pudo finalizar con éxito la implementación de Danezis.

En las implementaciones se utilizan librerías específicas para tratamiento de datos, tales como: Pandas[10], SciPy[11], Scikit-learn[12] o statsmodels[13].

7.2 Algoritmo de Wright

La implementación de este algoritmo [2] está incompleta y se deja para futuras ampliaciones de este trabajo.

El enfoque de Wright es interesante ya que difiere de otras aproximaciones en el sentido de, por ejemplo, que tiene en cuenta no solo las oscilaciones de clientes en la red Tor, sino aquellas variaciones en los accesos a través de Bridges.

7.3 Algoritmo de Danezis

La implementación de Danezis [12] (disponible en detector/danezis.py) ha sido completada con éxito y es funcional.

El método de detección de Danezis se basa en el modelo de comparar las conexiones de un determinado momento y país, con el número de conexiones pasadas de ese país obteniendo sobre una distribución de Poisson, teniendo en cuenta, además, la evolución de las conexiones en global. El modelo, permite modelar de forma eficaz los números de países con pocos usuarios de la red Tor, e incluso aquellos que poseen cero usuarios en determinados momentos.

Para calcular la evolución natural de conexiones, Danezis toma los 50 países con más peso de los datos disponibles de usuarios de Tor. Aquí vemos la función que extrae la lista de esos países:

[10] https://pandas.pydata.org/
[11] https://scipy.org/
[12] https://scikit-learn.org/stable/
[13] https://www.statsmodels.org/stable/index.html

```python
def top_countries(
    top_df: pandas.DataFrame, interval: int = TOP_COUNTRIES_INTERVAL
) -> list[str]:
    """Return a list with the MAX_MODEL_COUNTRIES top users countries

    Args:
        df (pandas.Dataframe): The dataframe to operated with.
        interval (int, optional): Number of days to be considered. Defaults to TOP_COUNTRIES_INTERVAL.

    Returns:
        list[str]: A list with MAX_MODEL_COUNTRIES countries as strings (ISO codes)
    """

    end_date = get_last_day()
    start_date = end_date - relativedelta(days=interval)

    top_df = top_df.loc[[start_date, end_date]]

    top_countries = (
        top_df.sum().sort_values().tail(MAX_MODEL_COUNTRIES).index.values.tolist()
    )
    top_countries.sort()
    return top_countries
```

Ilustración 21 Danezis. Función de corte de los 50 países con mayor número de usuarios en Tor

Sobre estos países se modela una serie que representa la tendencia de movimientos de usuarios teniendo en cuenta la estacionalidad de los datos (fundamentalmente, fines de semana).

Con ello, disponemos de una serie que nos indica los límites superior e inferior de tendencias de toda la red Tor, basada en un conjunto suficientemente representativo.

A continuación, la función de modelado de tendencias:

```python
def get_trends(df) -> tuple[pandas.Series]:
    countries = top_countries(df)
    trend_df = df.filter(countries)

    def diff(users) -> Union[None, float]:
        try:
            a, b = (float(users[0]), float(users[1]))
            if not b:
                return None
            return a / b
        except:
            return None

    trends = pandas.Series(dtype="float64")
    iter = trend_df.iteritems()
    for country, row in iter:
        dr = pandas.concat([row, row.shift(periods=TIME_INTERVAL_DAYS)], axis=1).apply(
            diff, axis=1
        )
        dr.name = country
        trends = pandas.concat([trends, dr], axis=1)

    # Clean NaN column/row      You, a month ago • advances in detector function
    trends = trends.iloc[TIME_INTERVAL_DAYS:, :].iloc[:, 1:]

    def norm_maxmin(series):
        loc, scale = norm.fit(series)
        return pandas.Series(
            {
                "max": norm.ppf(0.9999, loc, scale),
                "min": norm.ppf(1 - 0.9999, loc, scale),
            }
        )

    maxmin = trends.apply(norm_maxmin, axis=1)
    return (maxmin["max"], maxmin["min"])
```

Ilustración 22 Detalle de la función de Danezis para modelado de tendencias

Finalmente, se consideran uno a uno los datos de usuarios de cada país y se detecta una anomalía si dichos datos, la ratio de un país, están fuera del umbral del 99.99% de la distribución normal que propone la función de tendencias.

Salida de un conjunto de pruebas de la función de Danezis:

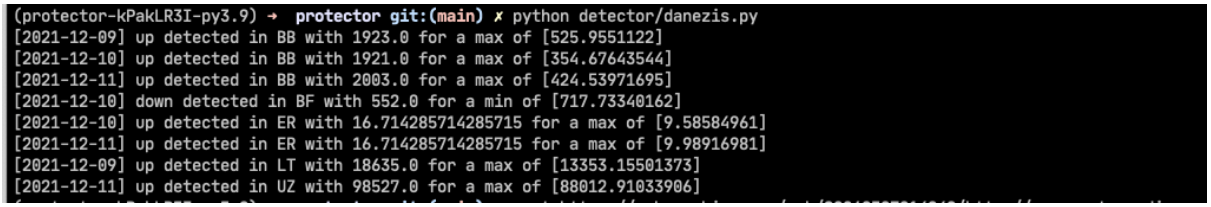


Ilustración 23 Salida de pruebas de la función Danezis

7.4 Alternativas y motivación de la elección

Danezis es un algoritmo relativamente sencillo de implementar siguiendo la metodología expuesta en la publicación homónima.

La alternativa de implementación hubiera sido emplear el lenguaje R para codificar los respectivos algoritmos, pero se descartó porque no suponía una mejora en la reutilización de código. Además, al disponer del equivalente en librerías científicas para el lenguaje Python, la cohesión con el proyecto iba a ser más elevada.

8 Bot y emisión de alertas

8.1 Plataforma de desarrollo de bots de Telegram

Un sistema de detección de anomalías debe alertar de alguna forma a los interesados. Es decir, debe establecerse un canal que permita emitir las alertas y dar conocimiento de eventos importantes.

El sistema, establece este canal a través de la plataforma de mensajería instantánea Telegram, concretamente, en su sistema de bots[14].

La integración se efectúa en forma de librería para el lenguaje de programación Python[15]. Esta puede verse en el archivo 'bot/alertbot.py'.

8.2 Canal de emisión de alertas

Cuando el módulo de análisis detecta anomalías, estas son registradas en base de datos y recuperadas por el bot, que notificará a las personas usuarias que se hayan subscrito al servicio de recepción de alertas.

[14] https://core.telegram.org/bots
[15] https://python-telegram-bot.readthedocs.io/en/stable/

El bot realiza una espera antes de rescatar las alertas de la base de datos, tras lo cual recorrerá la lista de subscripciones y enviará mensajes de alerta con los datos de estas. Podemos ver un ejemplo de este tipo de mensajes en la siguiente ilustración (se trata de una reemisión, pero el formato es exactamente el mismo):

Ilustración 24 Detalle de la reemisión de alertas a través del comando correspondiente

El bot puede ajustar su tiempo de comprobación de alertas gracias al motor de tareas asíncronas que posee incrustado (a través de la variable de entorno ALERTS_INTERVAL):

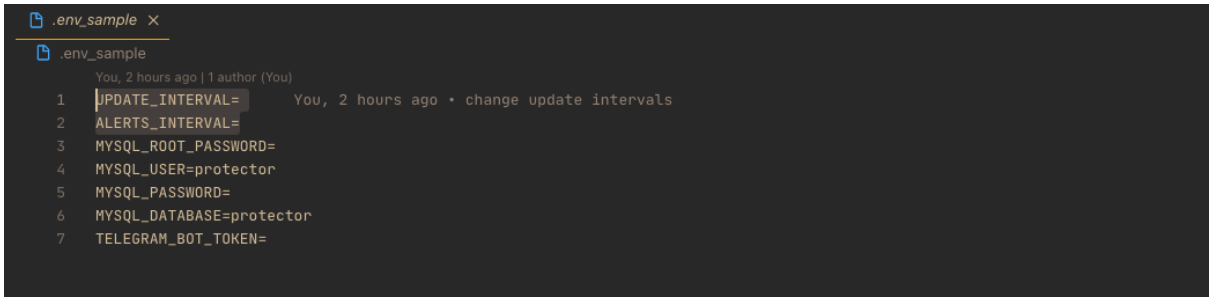


Ilustración 25 Configuración del intervalo de tiempo de comprobación de alertas

Para mejorar la visualización de alertas, se hace uso de emojis recodificando la salida de la base de datos a través de una función de enriquecimiento del texto:

```
def translate_alert(alerts):
    """This functions takes a list of alerts tuples and transform them into formatted strings
    Returns a list of formatted strings
    """
    f = lambda x: flag.flag(x[2])
    d = lambda x: x[1].date()
    ud = lambda x: "🔼" if x == "UP" else "🔽"
    n = lambda x: "{0:>6}".format(x)
    formatted = []
    for alert in alerts:
        formatted.append(
            f"{d(alert)} {f(alert)} {n(alert[3])} {ud(alert[4])} {n(alert[5])} {n(alert[6])} ({alert[7]})\n"
        )
    return formatted
```

Ilustración 26 Código de la función de enriquecimiento del texto de las alertas

La función de rescate de alertas verifica que existan subscriptores y en caso contrario no realiza procedimiento alguno. De hacerlo, extrae las alertas no enviadas, procede a su emisión y marca estas como ya enviadas en base de datos:

```
def once(context: CallbackContext) -> None:
    message = ""
    global global_id
    if not global_id:
        return

    try:
        c = connect()
        cursor = c.cursor()
        cursor.execute(selects["newalerts"])
        results = cursor.fetchall()
        if not results:
            message = "No alerts so far"
        else:
            results = translate_alert(results)
            message = "".join(results)
            cursor.execute(updates["marksentalerts"])
            c.commit()

    except Exception as e:
        logger.exception(e)

    for i in global_id:
        context.bot.send_message(chat_id=i, text=message)
```

Ilustración 27 Rutina de recuperación de alertas y emisión

8.3 Canal bidireccional de comandos

Como ya hemos tenido oportunidad de ver y se ha comentado, el bot no solo emite en una dirección, lo interesante, es que establece un completo canal de comunicaciones que permite interactuar con el sistema.

Es decir, se soporta un juego de comandos para poder emitir órdenes que completará el bot por nosotros y una vez ejecutadas nos mostrará su resultado.

Las posibilidades son innumerables dentro del contexto del sistema. Por ejemplo, se ha implementado una función para crear gráficas de estadísticas de uso por país:

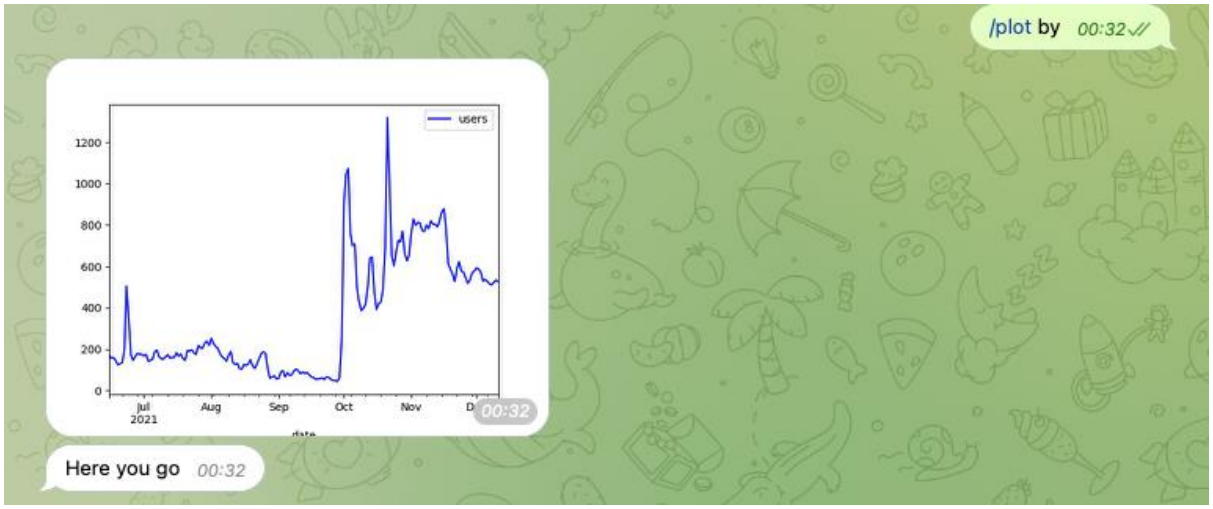


Ilustración 28 Gráfico de usuarios de Tor en Bielorusia

El bot posee un comando '/help' que lista los comandos que están actualmente implementados.

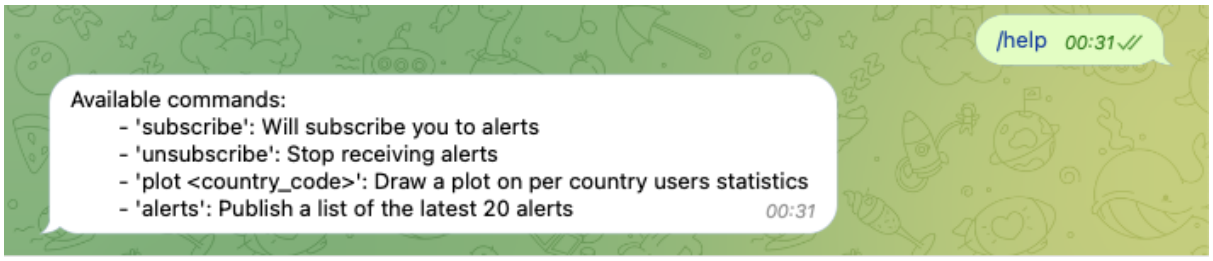


Ilustración 29 Salida del comando '/help'

La implementación de los comandos es relativamente sencilla. Se basa en la escritura de una función y posterior registro. En la ilustración, vemos la fase de registro de las diferentes funciones disponibles. Cada función va ligada a un comando (en bot/alertbot.py):

```
183  def main() -> None:
184      updater = Updater(TOKEN)
185      dispatcher = updater.dispatcher
186
187      dispatcher.add_handler(CommandHandler("help", help_command))      You, 3 weeks ago • added telegram bot
188      dispatcher.add_handler(CommandHandler("alerts", alerts_command))
189      dispatcher.add_handler(CommandHandler("plot", plot_command, pass_args=True))
190      dispatcher.add_handler(CommandHandler("subscribe", subscribe))
191      dispatcher.add_handler(CommandHandler("unsubscribe", unsubscribe))
192
193      q = updater.job_queue
194      q.run_repeating(once, ALERTS_DELAY)
195
196      updater.start_polling()
197      updater.idle()
198
199
200  if __name__ == "__main__":
201      main()
202
```

Ilustración 30 Registro de comando-función

Por ejemplo, la función de emisión de alertas está implementada del siguiente modo:

```python
def alerts_command(update: Update, context: CallbackContext) -> None:
    try:
        c = connect()
        cursor = c.cursor()
        cursor.execute(selects["alerts"])
        results = cursor.fetchall()
        if not results:
            update.message.reply_text(f"No results yet")
        else:
            results = translate_alert(results)
            update.message.reply_text("".join(results))

    except Exception as e:
        logger.exception(e)

```

Ilustración 31 Detalle de la implementación del comando '/alerts'

8.4 Alternativas y motivación de la elección

El método alternativo más común es el envío de un correo electrónico con el contenido de la alerta.

La única ventaja destacable sería que el correo electrónico posee un sistema de entrega semi-garantizada. Es decir, si no se ha podido enviar un correo al destinatario, el sistema seguirá intentándolo varias veces hasta darse por vencido. Además, acusará recibo de la entrega fallida, lo cual podría ser adecuado para intervenir ante dichos errores y examinar que ha podido ocurrir.

No obstante, las plataformas de mensajería implementan cierto nivel de recuperación de mensajes si la entrega no se ha podido realizar en el modo y tiempo previstos, aunque no llegan al nivel del correo electrónico.

La elección de Telegram obedece a las múltiples ventajas frente al correo electrónico: facilidad de despliegue (no es necesaria una cuenta de correo), sus mensajes no se descartan por spam y tenemos la posibilidad, que hemos explotado en este proyecto, de establecer una comunicación bidireccional con el sistema a través de comandos que podemos ampliar.

9 Conclusiones y trabajos futuros

9.1 Conclusiones y aprendizaje obtenido

El mayor aprendizaje es **abordar con éxito un proyecto**. Al final, la vida profesional está llena de proyectos (con variaciones en los objetivos, metodología y dimensiones) que van a necesitar de los conocimientos que nos otorgan las asignaturas que vamos superando, tanto en el Máster como en el Grado y es fundamental cerrar el círculo viviendo este proceso.

Además de **la experiencia que supone investigar un problema**, **proponer una solución y edificarla**, técnicamente se ha aprendido a sintetizar un conjunto de componentes heterogéneos de forma orquestada, en forma de, no un programa que se ejecuta a demanda, sino de un servicio 24/7 que corre continuamente en el sistema, pero solo consume recursos cuando es necesario abordar tareas de análisis por la llegada de nuevos datos.

Se han asimilado nuevos **conocimientos de funcionamiento de la red Tor**, un proyecto que facilita la libertad de expresión en países con serios problemas de censura e incluso persecución. A pesar de que el proyecto descrito en este TFM puede ser aplicado a otro tipo de redes, en la forma actual se encuentra orientado a los parámetros de funcionamiento de Tor, por lo que se ha tenido la oportunidad de aproximarse a la arquitectura y protocolos que permiten mantener en funcionamiento esta red.

Otro de los conocimientos cimentados con este trabajo es el de **trabajar con distintas tecnologías**, a priori, no relacionadas entre si. De hecho, si algo ha de ser destacado en la Ingeniería del Software es la capacidad de reutilización, lo cual representa un ahorro en multitud de posibles evaluaciones: impacto ecológico, ahorro económico, etc. En este proyecto se ha aprendido a combinar diferentes componentes para que trabajen entre si en un nuevo propósito.

9.2 Objetivos logrados

Salvo lo dispuesto en el siguiente punto, se han logrado prácticamente todos los objetivos impuestos y descritos con anterioridad.

9.3 Objetivos no logrados

Por falta de tiempo, no se han podido implementar otros módulos con diferentes algoritmos de detección de anomalías. Hubiese sido interesante disponer de otros módulos para realizar análisis comparativos de las diferentes detecciones e incluso combinar la salida de los diferentes módulos para encontrar nuevas perspectivas de análisis.

Naturalmente, estos objetivos no logrados pasan a formar parte de las futuras extensiones y trabajos a realizar tras traspasar el proyecto la barrera del presente TFM.

9.4 Respecto de la metodología

La metodología empleada ha funcionado de forma excepcional. Aun siendo un proyecto unipersonal, la ausencia y necesidad de coordinación no exime de que se hayan de cumplir los preceptos unidos a la disciplina impuesta.

Se ha utilizado control de versiones, documentación y diseño de componentes del mismo modo que si se hubiera ejecutado el proyecto con un equipo de personas.

El proyecto ha seguido por todos los componentes la misma filosofía: ir construyendo uno a uno desde el diseño y prototipo inicial hacia un componte aislado y funcional con capacidad de comunicarse con el resto del sistema y proporcionar un resultado acorde a las necesidades del conjunto.

9.5 Cambios de objetivos, temporización

No han existido cambios en los objetivos propuestos inicialmente a excepción de poder disponer de variedad en los algoritmos de detección. Si que se han detectado riesgos a los que, afortunadamente, se les ha aplicado la mitigación descrita en el capítulo correspondiente de forma satisfactoria.

Respecto a la temporización, todas las etapas propuestas han sido llevadas a cabo dentro del tiempo programado para estas, respetándose tanto las entregas de seguimiento (PECs) como la cronología expuesta que vimos anteriormente.

9.6 Futuros trabajos y ampliaciones

Existen varias vías principales y específicas de ampliación y reutilización del sistema que detallaremos a continuación.

9.6.1 Ampliación de módulos de análisis

Como ya se comentó, una de las posibles vías de extensión del proyecto es la implementación de nuevos algoritmos de detección de anomalías en series temporales.

Esto permitiría al sistema disponer de mayor variedad en las alertas, poder comparar puntuaciones o eventos de manera correlativa o simplemente disponer de más información para valorar un incidente.

9.6.2 Reutilización del sistema

El sistema está basado y hecho para la detección de anomalías en la red Tor, pero nada impide reutilizar los componentes para adaptarlos a otros modelos de red.

Es más, incluso se podría realizar una adaptación para incorporarlo al modelo de red privada de una organización y realizar un conteo de tráfico para detectar nodos caídos o, al contrario, nodos con excesivo consumo del ancho de banda disponible (algo que suele relacionarse con actividad maliciosa o compromiso del sistema).

9.6.3 Uso optimizado de los datos recabados

A pesar de que estamos recabando datos de métricas de la red Tor, solo estamos usando un subconjunto de estas, luego podríamos extender los algoritmos de detección a estos datos, ya sea adaptándolos o incluso creando nuevos tipos de detecciones. Por ejemplo, en vez de usar el número de usuarios por país podríamos usar el tamaño geolocalizado de los relays para detectar ataques a estos.

10 Glosario

- **Bot**: Aplicación autónoma que se ejecuta en un segundo plano y ofrece cierta funcionalidad en base al disparo de eventos o comandos.
- **Bridge**: Relays especiales que permiten evadir sistemas de censura que de otro modo impedirían o detectarían el acceso a la red Tor.
- **Celery:** Sistema de gestión de colas de trabajo implementado en el lenguaje de programación Python.
- **Grafana**: Aplicación de visualización rica de datos a partir de múltiples tipos de fuentes.
- **MySQL**: Base de datos SQL relacional.
- **Nodo (o Relay) de entrada**: Componente de la red Tor por el que los clientes acceden a esta.
- **Nodo (o Relay) intermedio**: Componente de la red Tor que enruta el tráfico hacia otro nodo intermedio o un nodo de salida.
- **Nodo (o Relay) de salida**: Componente de la red Tor que recibe tráfico de un nodo intermedio hacia Internet.
- **Onion Service**: Servidor que publica sus servicios dentro de la red Tor y al que solo puede accederse a través de esta.
- **Python**: Lenguaje de programación de propósito general con tipado dinámico.
- **Telegram**: Plataforma de mensajería.
- **Tor**: Red que permite a un cliente navegar por Internet protegiendo la identidad de este.
- **Redis**: Sistema de base de datos del tipo clave-valor usada por Celery como bróker de mensajes.

11 Bibliografía y fuentes consultadas

[1] Cleveland, Rb et al. “STL: A seasonal-trend decomposition procedure based on loess (with discussion).” (1990).

[2] Wright, Joss, Alexander Darer, and Oliver Farnan. "On identifying anomalies in tor usage with applications in detecting internet censorship." *Proceedings of the 10th ACM Conference on Web Science*. 2018.

[3] Mani, Akshaya et al. “Understanding Tor Usage with Privacy-Preserving Measurement.” *Proceedings of the Internet Measurement Conference 2018* (2018): n. pag.

[4] Backes, Michael et al. “(Nothing else) MATor(s): Monitoring the Anonymity of Tor's Path Selection.” *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014): n. pag.

[5] Wright, Joss & Darer, Alexander & Farnan, Oliver. (2018). On Identifying Anomalies in Tor Usage with Applications in Detecting Internet Censorship. 87-96. 10.1145/3201064.3201093.

[6] M. H. Bhuyan, D. K. Bhattacharyya and J. K. Kalita, "Network Anomaly Detection: Methods, Systems and Tools," in IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 303-336, First Quarter 2014, doi: 10.1109/SURV.2013.052213.00046.

[7] Loesing, Karsten & Murdoch, Steven & Dingledine, Roger. (2010). A Case Study on Measuring Statistical Data in the Tor Anonymity Network. 203-215. 10.1007/978-3-642-14992-4_19.

[8] **How Much Anonymity does Network Latency Leak?** (**PDF**) (Cached: **PDF**) by Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin.

[9] *In ACM Transactions on Information and System Security* ***13****(2), February 2010.* (**BibTeX entry**)

[10] *Sources - Tor Metrics* [en línea] [fecha de consulta: 25-sept-2021]. Disponible en: https://metrics.torproject.org/collector.html

[11] *OrNetStats* [en línea] [fecha de consulta: 28-sept-2021]. Disponible en: https://nusenu.github.io/OrNetStats/

[12] *An anomaly-based censorship-detection system for Tor* (PDF) [en línea] [fecha de consulta: 14-dic-2021] by Danezis, George.

Anexo I. La red Tor

Este anexo tiene por objeto describir de forma breve qué es y como funciona la red Tor[16].

¿Qué es la red Tor?

La red Tor es un proyecto de fuente abierta que tiene por objetivo la protección de la privacidad de las personas usuarias de Internet. Nace del concepto de enrutamiento cebolla, que consiste en el cifrado sucesivo, por capas, de un mensaje que transita a través de un número de nodos o componentes que pertenezcan a dicha red.

¿Cómo funciona?

Un usuario de la red Tor, emite un mensaje (por ejemplo, una petición web) pero no se comunica directamente con el destinatario (por ejemplo, un servidor web). En vez de ello, y una vez conectado a la red Tor, selecciona un nodo de entrada a la red y le pasa el mensaje, éste cifra el mensaje y lo retransmite a otro nodo intermedio de la red, el cual lo cifrará de nuevo y lo enviará a un nodo de salida, el cual, finalmente lo remitirá al servidor que está fuera de la red Tor.

Cuando se establece esa conexión entre cliente y nodos o relays, se denomina circuito.

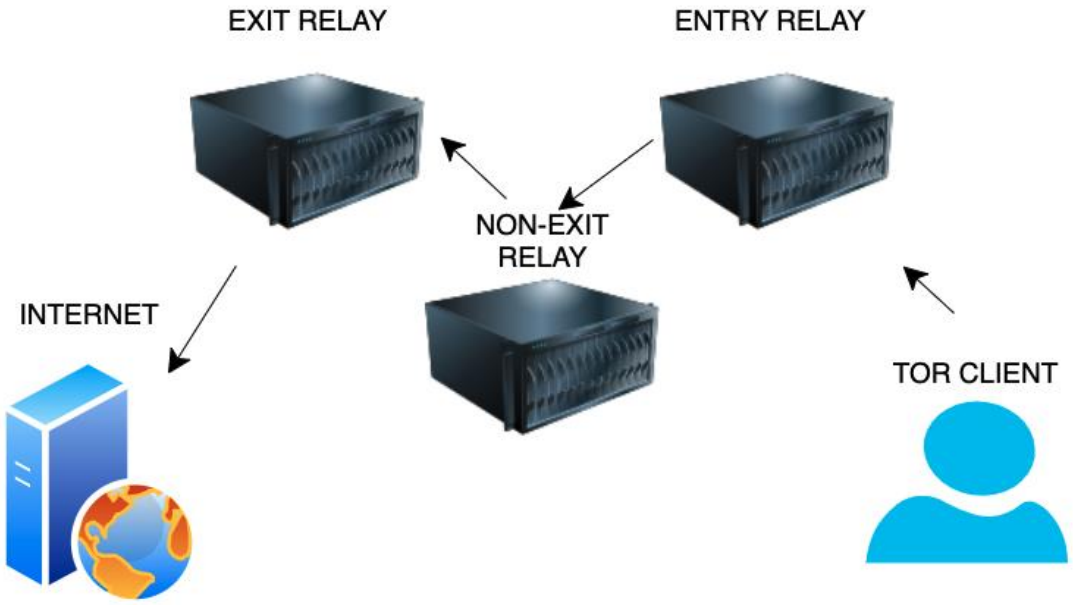


Ilustración 32 Esquema de funcionamiento de la red Tor (elaboración propia)

[16] https://www.torproject.org/

Como podemos observar, cada retransmisión del mensaje produce un cifrado adicional del mensaje, lo cual evita que se desvele tanto el contenido como el emisor original.

Tan solo la información necesaria para encaminar el mensaje a través de la red es proporcionada. Esa adición de capa sobre capa similar al concepto natural de las capas de una cebolla es lo que le dota de tan peculiar nombre.

Anexo II. Instalación y despliegue del sistema

El primer paso es instalar Docker de manera adecuada en el sistema y asegurarnos que la herramienta 'docker-compose' lo está. En algunos sistemas es una instalación independiente, mientras que en algunas distribuciones de Docker ya está integrada.

Docker es el sistema de contenedores que nos permitirá liberarnos de las dependencias e instalaciones adicionales.

De manera opcional, si tenemos Git instalado, procederemos a clonar el proyecto desde su repositorio en Github con el siguiente comando:

git clone –depth=1 https://github.com/deibit/protector

La opción '—depth=1' hace que solo nos bajemos la versión actual del repositorio, sin el historial de cambios. Lo cual ahorrará en tiempo de descarga.

<imagen de clonación del repositorio>

De forma alternativa, si no contamos con Git, podemos descargar directamente un archivo comprimido en formato zip el cual, podremos descomprimir donde queramos.

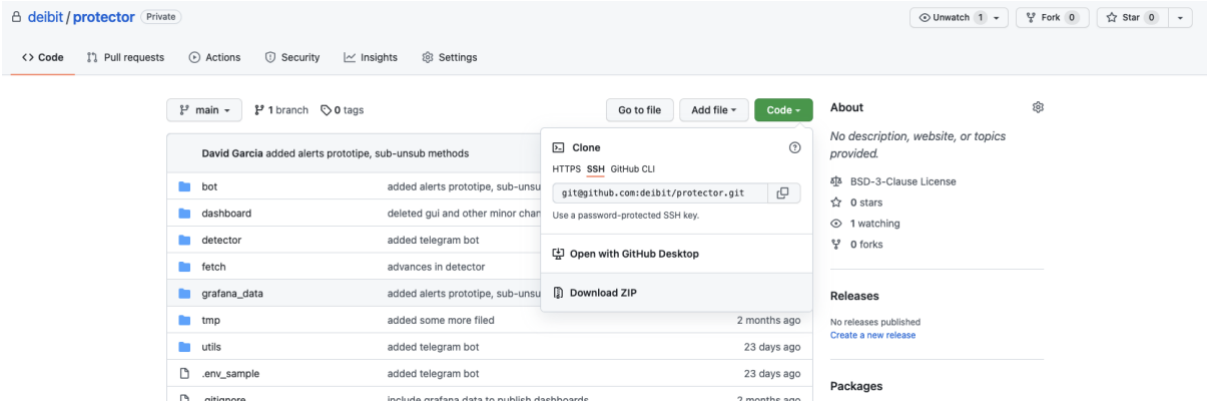


Ilustración 33 Descarga del archivo Zip desde el repositorio

Una vez en el directorio, abriremos el archivo '.env_sample' y editaremos las siguientes variables que serán inyectadas en el entorno:

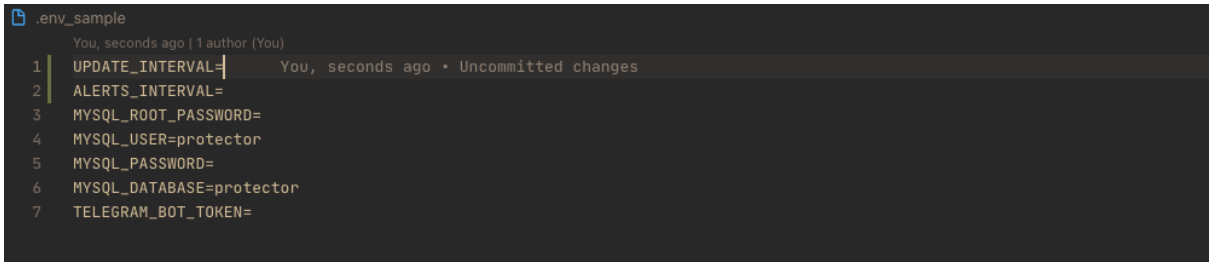


Ilustración 34 Variables de entorno necesarias para el arranque del sistema

Las variables de entorno son:

MYSQL_ROOT_PASSWORD, es la contraseña maestra del usuario root de MySQL. Debermos indicar una contraseña segura.

MYSQL_USER, usuario actual de MySQL, no cambiar del valor por defecto: ‘protector’.

MYSQL_PASSWORD, contraseña se acceso a la base de datos. Deberemos indicar una contraseña segura.

TELEGRAM_BOT_TOKEN, token del bot de Telegram (ver anexo específico para obtener dicho token)

Existen dos variables de entorno importantes con las que definiremos cuando queremos obtener actualizaciones y comprobar la existencia de alertas:

UPDATE_INTERVAL, indica cuantos segundos deben transcurrir hasta la siguiente actualización de los datos.

ALERTS_INTERVAL, indica cuantos segundos deben transcurrir hasta la siguiente comprobación de la existencia de alertas por parte del bot.

Posteriormente, el sistema se autoconstruye para ser desplegado con:

$ docker-compose up

Se construirá una imagen necesaria para desplegar el contenedor asociado al bot y a los componentes de análisis y captación de datos.

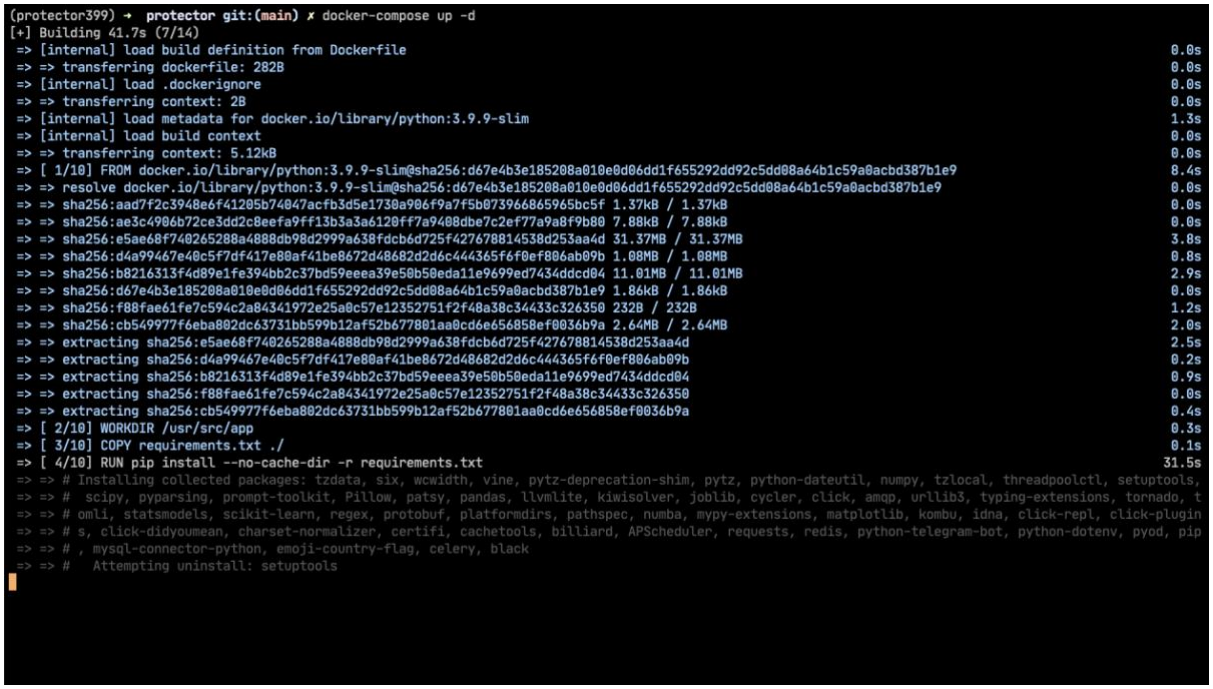
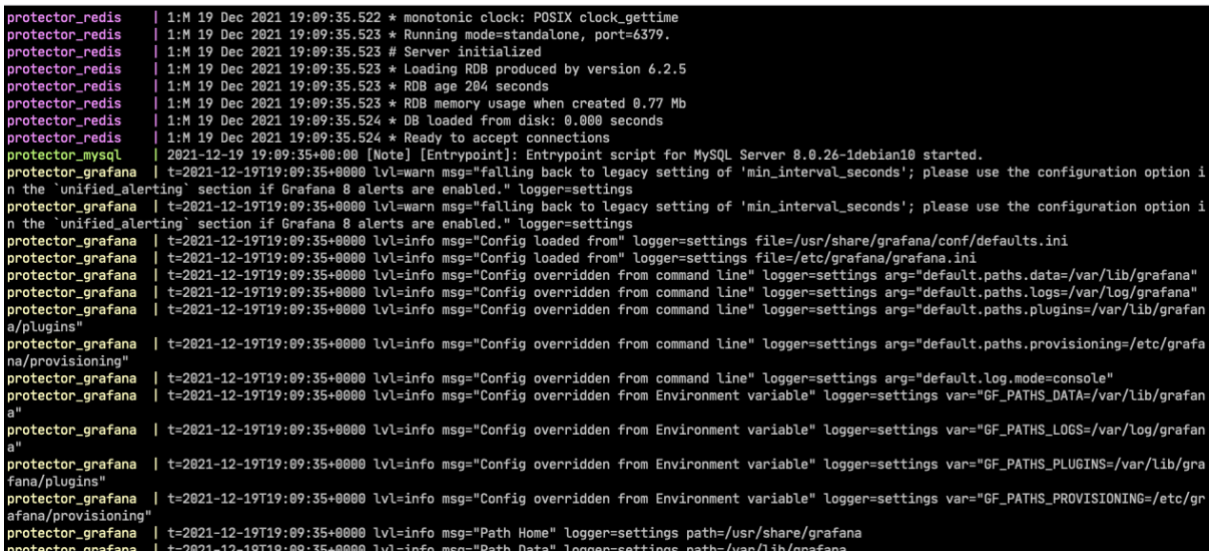


Ilustración 35 Detalle de la construcción de la imagen de ProtecTor

Una vez la imagen ha sido construida, el mismo comando en curso procede a ejecutar el sistema, se observará un registro de salida similar a este:



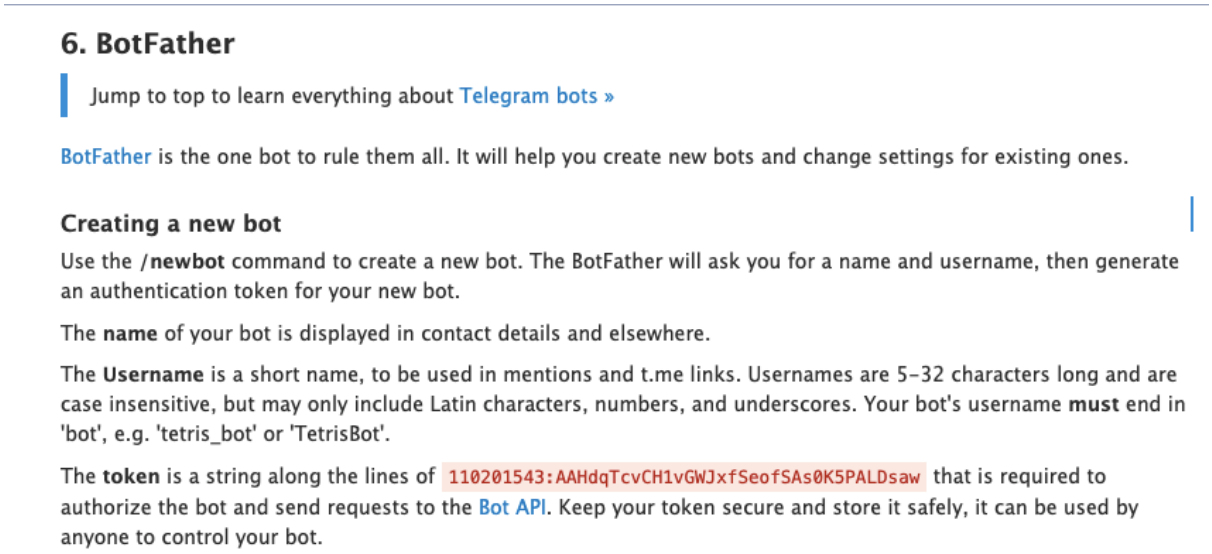
Dado que es una arquitectura orientada a servicios, no existe un “interfaz” en el sentido clásico del termino, lo más parecido sería el canal bidireccional del bot, que comenzará a atender comandos tan pronto el contenedor se ejecute y el cuadro de mandos provisto por Grafana.

El cuadro de mandos debería estar disponible en: https://localhost:3000

El bot estará disponible en la aplicación de Telegram y se deberá buscar por el nombre que se le haya dotado.

Anexo III. Obtención de un token para bots de Telegram

El proceso de obtención de un token para poner en marcha un bot en la plataforma de Telegram es sencillo y gratuito.



6. BotFather

> Jump to top to learn everything about Telegram bots »

BotFather is the one bot to rule them all. It will help you create new bots and change settings for existing ones.

Creating a new bot

Use the /**newbot** command to create a new bot. The BotFather will ask you for a name and username, then generate an authentication token for your new bot.

The **name** of your bot is displayed in contact details and elsewhere.

The **Username** is a short name, to be used in mentions and t.me links. Usernames are 5-32 characters long and are case insensitive, but may only include Latin characters, numbers, and underscores. Your bot's username **must** end in 'bot', e.g. 'tetris_bot' or 'TetrisBot'.

The **token** is a string along the lines of `110201543:AAHdqTcvCH1vGWJxfSeofSAs0K5PALDsaw` that is required to authorize the bot and send requests to the Bot API. Keep your token secure and store it safely, it can be used by anyone to control your bot.

Ilustración 36 Detalle de las instrucciones para obtener un token (https://core.telegram.org/bots#6-botfather)

En primer lugar, debemos tener una cuenta de Telegram como usuario. La forma de obtenerla es sencilla, pero hemos de poseer un número de teléfono para registrarla. Basta con instalar la aplicación y darse de alta mediante un proceso que la propia aplicación ofrece al usuario en la primera ejecución.

La forma de dar de alta un nuevo bot (asociado a nuestra cuenta) es a través de un bot especial de la propia plataforma denominado 'BotFather'.

A dicho bot, le enviamos un comando denominado '/newbot'. Tras preguntarnos por el nombre y el nombre de usuario del bot, nos proveerá de un token, necesario para la autenticación del bot y que el sistema recuperará para insertarla en el código de ProtecTor, concretamente de la variable de entorno **TELEGRAM_BOT_TOKEN**, que vimos anteriormente.

FIN DE LA MEMORIA