MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

Diseño e implementación de un entorno de pruebas para la evaluación del rendimiento de los sistemas de virtualización ligera

JAVIER GARCÍA CÉSPEDES

2023

MÁSTER uNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

trabajo fin de MÁSTER

**Título:** ……………….

**Autor:** Javier García Céspedes

**Tutor:** Joaquín Salvachua

**Ponente:** D. ………………

**Departamento:** …………

Miembros del Tribunal

**Presidente:** D. ……………

**Vocal:** D. …………..

**Secretario:** D. …………..

**Suplente:** D. ……………..

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de: ………

Madrid, a de de 20…

**UNIVERSIDAD POLITÉCNICA DE MADRID**

**ESCUELA TÉCNICA SUPERIOR**

**DE INGENIEROS DE TELECOMUNICACIÓN**

****

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

Diseño e implementación de un entorno de pruebas para la evaluación del rendimiento de los sistemas de virtualización ligera

JAVIER GARCÍA CÉSPEDES

2023

RESUMEN

El objetivo de este Trabajo de Fin de Máster de título: “Diseño e implementación de un entorno de pruebas para la evaluación del rendimiento de los sistemas de virtualización ligera” es diseñar e implementar un entorno para medir la velocidad de realización de diferentes tareas como pueden ser: cálculos matemáticos, peticiones web o despliegues de servicios en función de dos lenguajes de programación (golang y rust), y determinar cuál de los dos es mejor para cada tipo de tarea. Para ello, se desarrollarán diferentes pruebas que se implementarán en la nube bajo unos mismos parámetros de características de virtualización del sistema, para que posteriormente se recojan los resultados de las diferentes pruebas con ELK, que funcionará a modo de Pipeline y nos dotará con la capacidad para poder también visualizar los datos para su comparación. Estos resultados serán datos temporales, pues medimos el tiempo que tarda en realizarse una acción determinada, y para minimizar los errores de medida se realizarán las mismas pruebas de manera repetida hasta que el rango de tiempos sea del orden de los segundos. Las pruebas que se realicen serán desarrolladas en Python, para facilitar su desarrollo y posteriormente se procesarán con los dos lenguajes de programación previamente mencionados para poder ver como responden a las situaciones planteadas con anterioridad.

summary

The objective of this Master's Thesis titled 'Design and Implementation of a Test Environment for the Performance Evaluation of Lightweight Virtualization Systems' is to design and implement an environment to measure the speed of performing different tasks, such as mathematical calculations, web requests, or service deployments, using two programming languages (Go and Rust). The aim is to determine which language is better suited for each type of task. To achieve this, various tests will be developed and implemented in the cloud, under the same system virtualization parameters. Subsequently, the results of the different tests will be collected using ELK, which will function as a Pipeline and provide us with the capability to visualize the data for comparison as well. These results will be time-based data since we measure the time it takes to perform a specific action. To minimize measurement errors, the same tests will be repeated until the time range is in the order of seconds. The tests will be developed in Python to facilitate their development and will later be processed using the two programming languages to observe how they respond to the previously mentioned situations.

PALABRAS CLAVE

Deben reflejar el contenido del trabajo, deberían servir para localizar el TFM mediante búsqueda bibliográfica...

KEYWORDS

….

índice del contenido

[1. introducción y objetivos 8](#_Toc141813216)

[1.1. Introducción 8](#_Toc141813217)

[1.2. Objetivos 8](#_Toc141813218)

[2. desarrollo 2](#_Toc141813219)

[2.1. Entornos automáticos de pruebas de rendimiento 2](#_Toc141813220)

[2.1.1. Características de los entornos automáticos 2](#_Toc141813221)

[2.1.2. Curvas de caracterización 2](#_Toc141813222)

[2.2. ELK 3](#_Toc141813223)

[2.2.1. Elasticsearch 4](#_Toc141813224)

[2.2.2. Sintaxis 5](#_Toc141813225)

[2.3. Logstash 6](#_Toc141813226)

[2.3.1. Ejemplo de uso 7](#_Toc141813227)

[2.4. Kibana 8](#_Toc141813228)

[2.4.1. Características 9](#_Toc141813229)

[2.4.2. Sintaxis 9](#_Toc141813230)

[2.4.3. uso 10](#_Toc141813231)

[2.5. Go 10](#_Toc141813232)

[2.5.1. Instalación 11](#_Toc141813233)

[2.5.2. Sintaxis 11](#_Toc141813234)

[2.6. RUST 16](#_Toc141813235)

[3. resultados 17](#_Toc141813236)

[4. conclusiones y líneas futuras 18](#_Toc141813237)

[4.1. Conclusiones 18](#_Toc141813238)

[4.2. Líneas futuras 18](#_Toc141813239)

[5. bibliografía 19](#_Toc141813240)

[anexo a: aspectos ÉTICOS, económicos, sociales y ambientales 20](#_Toc141813241)

[A.1 iNTRODUCCIÓN 20](#_Toc141813242)

[A.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO 20](#_Toc141813243)

[A.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS 20](#_Toc141813244)

[A.4 CONCLUSIONES 20](#_Toc141813245)

[anexo b: presupuesto económico 21](#_Toc141813246)

# introducción y objetivos

## Introducción

## Objetivos

El propósito de este trabajo de fin de máster es analizar la velocidad del os lenguajse de programación Go y Rust antes distintos tipos de operaciones y situaciones. Go se destaca por su eficiencia en el manejo de concurrencia y su capacidad para realizar tareas concurrentes de manera rápida y sencilla, mientras que Rust sobresale en el rendimiento y la seguridad de sistemas y aplicaciones de bajo nivel.

Para ello, se hará uso de lo que se conocen como los entornos automáticos de pruebas de rendimiento. Estos, son sistemas adaptados con los que se pueden evaluar el rendimiento y la capacidad del sujeto de prueba al que se sometan. Están diseñados para simular escenarios reales de uso y medir cómo responde el sistema ante un número determinado de usuarios, transacciones o solicitudes.

# desarrollo

En esta primera parte, se presentan de manera teórica todos los componentes que formarán parte del trabajo. Es por ello, que se expone en primera instancia lo que son los entornos automáticos de pruebas de rendimiento. Posteriormente, se detalla que es ELK y se entran en detalle en los tres componentes que lo forman, para finalizar con unas nociones básicas de Go y Rust.

## Entornos automáticos de pruebas de rendimiento

Los entornos automáticos de pruebas de rendimiento. son sistemas adaptados con los que se pueden evaluar el rendimiento y la capacidad del sujeto de prueba al que se sometan. Están diseñados para simular escenarios reales de uso y medir cómo responde el sistema ante un número determinado de usuarios, transacciones o solicitudes.

Las pruebas de rendimiento son cruciales para asegurar que una aplicación o sistema pueda manejar la carga esperada y proporcionar un rendimiento óptimo antes de ser puesto en producción o lanzado a un público más amplio.

### Características de los entornos automáticos

Algunas características comunes de los entornos automáticos de pruebas de rendimiento incluyen:

* **Escalabilidad**: Estos entornos permiten simular diferentes niveles de carga y tráfico, lo que ayuda a identificar los puntos de saturación y evaluar la capacidad de escalabilidad del sistema.
* **Automatización de pruebas**: Las pruebas de rendimiento se pueden diseñar y ejecutar automáticamente mediante scripts, lo que facilita la repetición de las pruebas y la comparación de resultados.
* **Análisis de resultados**: Los entornos automáticos suelen proporcionar herramientas para analizar y visualizar los datos recopilados durante las pruebas, lo que ayuda a identificar cuellos de botella, tiempos de respuesta lentos u otros problemas de rendimiento.
* **Carga realista**: Los entornos de pruebas pueden simular diferentes tipos de usuarios, patrones de tráfico y situaciones reales para evaluar el rendimiento en condiciones cercanas a la producción.
* **Monitoreo del sistema**: Durante las pruebas, se puede monitorear el rendimiento del sistema en tiempo real, lo que permite identificar problemas a medida que ocurren.
* **Generación de informes**: Los entornos automáticos suelen generar informes detallados con métricas y resultados de las pruebas, lo que facilita la comunicación y toma de decisiones.

### Curvas de caracterización

Para el caso de la generación de informes, se debe hacer especial énfasis en las curvas ede caracterización, estas son una representación gráfica que muestra cómo el rendimiento de un sistema, servicio o aplicación varía en función de la carga. La curva de caracterización es esencial a la hora de la generación de informes, ya que permite comprender de una manera sencilla el comportamiento del sujeto de pruebas ante distintos estímulos, como pueda ser la carga o el número de usuarios que disponen del servicio.

A medida que se aumenta la carga o el número de usuarios en las pruebas, se registran los resultados para cada nivel de carga y se representan gráficamente. La curva de caracterización resultante muestra cómo el rendimiento del sistema evoluciona en función de la carga. Generalmente, esta curva puede tener diferentes formas:

* **Lineal**: El rendimiento aumenta o disminuye de manera proporcional a la carga o el número de usuarios. Es ideal cuando el sistema es altamente escalable.
* **Plateau**: El rendimiento se mantiene constante durante un rango de carga, pero una vez que se alcanza un punto crítico, el rendimiento comienza a degradarse.
* **Descenso brusco**: El rendimiento cae bruscamente al llegar a cierto nivel de carga, lo que indica un punto de saturación en el sistema.
* **Descenso gradual**: El rendimiento se degrada progresivamente a medida que aumenta la carga, sin un punto de saturación claramente definido.

A través de estas curvas de caracterización, se podrán resaltar las conclusiones obtenidas de una manera más cómoda y visual, favoreciendo la presentación de los resultados.

## ELK

ELK hace referencia a las siglas de Elasticsearch (E), Logstash (L) y Kibana (K). Estos son tres proyectos de código libre, y juntos proporcionan un enfoque integral en la consolidación, gestión y análisis de registros de las aplicaciones en tiempo real [1]. A través de este sistema se pueden recopilar y centralizar todos los registros que generan las aplicaciones para ser analizados de una manera eficaz.

Imagen que contiene Esquemático

Descripción generada automáticamente

*Figura 1. Representación de ELK [2]*

A la hora de ejecutar estos servicios, se realizar de manera conjunta: Elasticsearch es un motor de búsqueda basado en Apache Lucene y diseñado para manejar grandes cantidades de datos y ser capaz de proporcionar una búsqueda rápida y escalable. Además, es capaz de analizar el texto. Por otro lado, Logstash se encarga de la recopilación de los datos para su posterior transformación y ser enviados a Kibana, una interfaz de usuario que nos permite visualizar estos datos y proporciona distintos métodos de representación como pueden ser gráficos, tablas o paneles de control. Además, cabe destacar Beats, una familia de programas o componentes software que consumen pocos recursos y se encargan de diferentes funciones que se explicarán en detalle.

Para la instalación de ELK, existen varios métodos distintos, en este caso, se va a usar una imagen de DockerHub. Para su uso, se necesita instalar Docker y registrarse en DockerHub, el repositorio de imágenes de Docker.

Una vez instalado Docker, se debe descargar la imagen de ELK de DockerHub con el siguiente comando en terminal: “*docker pull sebp/elk*”. Una vez descargada la imagen podemos crear un contenedor para su uso con el siguiente comando: “*sudo docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 -p 5000:5000 -it --name elk sebp/elk*”. Con esto se establecen los siguientes parámetros:

1. La interfaz JSON de Elasticsearch en el puerto 9200
2. La interfaz de Logstash en el puerto 5044
3. La interfaz web de Kibana en el puerto 5601
4. Logstash Lumberjack en el puerto 5000

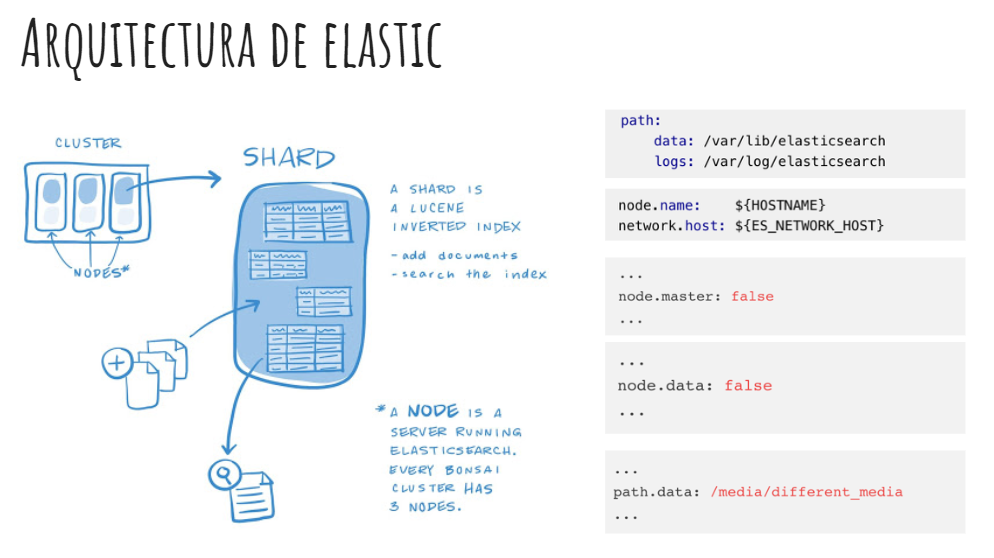
A continuación, vamos a ver los tres componentes básicos que componen ELK en detalle.

### Elasticsearch

Como se ha comentado, Elasticsearch es una base de datos NoSQL basada en Apache Lucene que incluye un motor de búsqueda, y está basada en Java [3]. Al ser una base de datos NoSQL, destaca por su escalabilidad y tiene la particular característica de ser capaz de realizar avanzadas búsquedas mediante expresiones regulares. Además, debido a su uso para analizar datos, permite reconocer lexemas, abreviaturas o frecuencias de aparición de palabras o cadenas de texto.

Para la implementación de estas características, se necesita de una arquitectura capaz de soportar todas estas funciones. En particular, la base de datos Elasticsearch tiene la siguiente arquitectura [4]:

* **Índices**: esta es la asignación de nombres lógicos que asigna la plataforma a los datos de manera automática una vez se agrega la información.
* **Shards**: son las instancias de índices que maneja el motor Lucene en la plataforma. Por defecto, existen cinco shards por cada índice
* **Réplicas**: este elemento hace referencias a las shards replicadas en el proceso de indexación y de búsqueda. Esto permite tener backups de los datos. Existe como mínimo, una réplica por índice.
* **Segmentos y commit points**: estos elementos pertenecen a cierta división de shards que consiste en acelerar el tiempo de búsqueda y generar estabilidad en la estructura de los datos.
* **Nodo**: se refiere al servidor de búsqueda que ejecuta ElasticSearch. Cada clúster posee varios nodos.
* **Clúster**: en este elemento se almacenan los datos gracias a un conjunto de nodos, los cuales pueden ser: máster node, data note, ingest node o coordinating node. Los nodos máster gestionan los índices y su distribución en los nodos de datos, donde se almacenan los shards y se realizan los procesos de búsqueda. Los Ingest Node se encargan de escuchar las entradas de datos y almacenarlas. Finalmente, los nodos coordinating balancean la carga y preprocesan los datos en futuras fases de agregación

*Figura 2. Arquitectura de Elasticsearch [4]*

Además, existen varias bibliotecas para interaccionar con Elasticsearch en lenguajes como Python y Java, y se pueden realizar consultas SQL a través de herramientas como ODBC o JDBC, ya que está basada en JSON.

### Sintaxis

Es por ello, que para interactuar con Elasticsearch de manera básica, se tiene la siguiente sintaxis:

* Indexar un documento:

PUT /nombre\_indice/\_doc/id\_del\_documento

{

"campo1": "valor1",

"campo2": "valor2",

...

}

* Obtener un documento:

GET /nombre\_indice/\_doc/id\_del\_documento

* Buscar documentos:

GET /nombre\_indice/\_search

{

"query": {

"term": {

"campo": "valor"

}

}

}

* Actualizar un documento:

POST /nombre\_indice/\_update/id\_del\_documento

{

"doc": {

"campo": "nuevo\_valor"

}

}

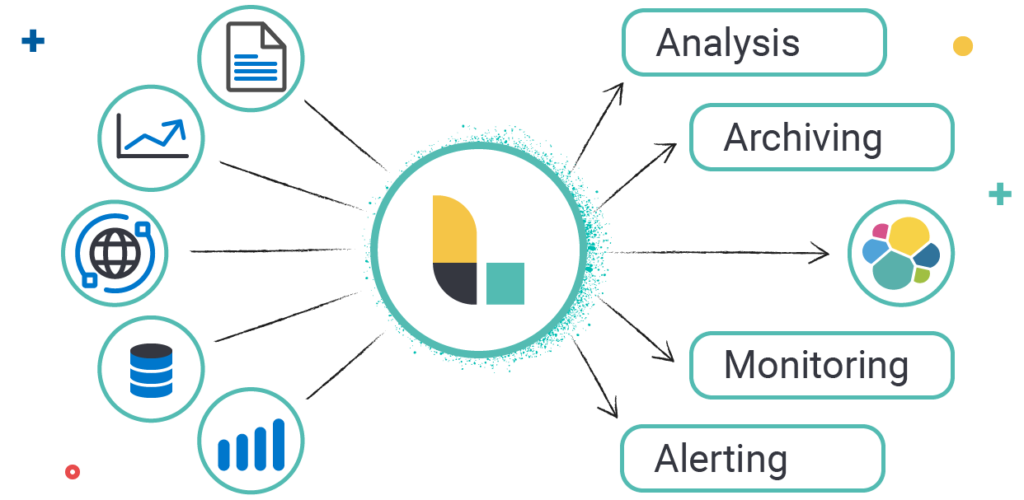
* Eliminar un documento:

DELETE /nombre\_indice/\_doc/id\_del\_documento

Además, existen funcionalidades más avanzadas, como pueden ser consultas avanzadas, donde se puede filtrar por rango, coincidencia de frases, prefijos o proximidad entre otras, y también tiene la opción de realizar agregaciones, que nos permiten realizar cálculos y analizar los datos.

## Logstash

Logstash es la parte de ELK que se encarga de centralizar en tiempo real toda la información para normalizarla a y distribuirla. Al igual que el resto de las herramientas, puede funcionar de manera independiente. Sin embargo, su integración con Beats, Elasticsearch y Kibana es nativa, ofreciendo así gran valor para la transformación avanzada de la información [5]. Esta herramienta al igual que la anterior, fue desarrollada por la Empresa Elastic y está escrito en JRuby.

*Figura 3. Ilustración de Logstash [5]*

Al igual que cualquier otro pipeline, Logstash está formado por tres pasos que se ordenan de manera secuencial:

* **Inputs**: En este primer proceso, se centraliza toda la información. En el caso de Logstash, es capaz de recibir eventos de múltiples clases, entre los que se incluyen TCP y UDP.
* **Filters**: En este segundo proceso, se transforman los datos, adaptándolos a las necesidades y características del cliente.
* Output: Finalmente, en este último paso, se envían los datos a destino, en este caso, a Elasticsearch. Para ello, Logstash cuenta con multitud de plugins creados por la comunidad. Estos plugins pueden agregar conectividad con fuentes de datos específicas, ofrecer filtros adicionales o permitir la salida de datos a destinos personalizados.

### Ejemplo de uso

Para establecer un pipeline simple, se deben designar dos rutas que hagan de input y output. En la primera de ellas se almacenarán los documentos sin procesar, y en la segunda se almacenará el resultado del procesamiento de estos. A la hora de procesar los eventos, se aplican filtros y transformaciones para darle formato y estructura a los datos. Algunos de los filtros y transformaciones disponibles incluyen análisis de patrones, enriquecimiento de datos, eliminación o adición de campos, entre otros. Una vez que los datos han sido procesados en Logstash, se pueden enviar a otros sistemas para su almacenamiento y análisis. Elasticsearch es uno de los destinos más comunes. Kibana, por su parte, proporciona una interfaz de visualización para explorar y analizar los datos.

Un ejemplo de filter podría ser el siguiente, suele tomar el nombre de “logstash.conf”:

Texto

Descripción generada automáticamente

*Figura 5. Ejemplo de un “Logstash.conf”.*

En el input, hemos definido, la ruta de donde Logstash va a tomar en este caso los logs, y mediante el “filter” se analiza el formato de los logs. Finalmente, mediante la sentencia output, se establece el lugar donde la ruta donde se almacenan los resultados, en este caso, un servidor local de Elasticsearch. Para ejecutar Logstash, se utilizaría la siguiente sentencia en terminal: *logstash -f logstash.conf*. Además, gracias al funcionamiento en tiempo real, cada vez que se actualicen los datos en la ruta input, los cambios se reflejarán en la cola output.

Para la gestión de las colas, Logstash las gestiona todas en memoria, por lo que un posible error en el equipo que tenga afectación al servicio puede afectar al correcto procesamiento de los eventos en proceso. Es por ello, que Logstash cuenta con la opción de establecer una cola persistente, reduciendo los riesgos de pérdida de información. Con esta opción, además, Logstash es capaz de asumir picos puntuales en el número de eventos recibidos, que de otra manera podría suponer una pérdida de información según la arquitectura que esté establecida [5]. Además, también se pueden tratar los eventos de error en una “DeadLetterQueue”, separándolos de los eventos correctos para que se puedan analizar por separado.

## Kibana

Como ya hemos comentado anteriormente, Kibana es una aplicación que proporciona un soporte frontend de visualización de datos específico para ELK. Además, actúa como la interfaz de usuario para monitorear, gestionar y asegurar un clúster del Elastic Stack; además de como concentrador centralizado de las soluciones integradas desarrolladas en el Elastic Stack [7]. La aplicación se desarrolló en 2013 y es el último paso dentro de la cadena de ELK.

### Características

Las principales características de Kibana son las siguientes [7]:

1. Buscar, ver y visualizar datos indexados en Elasticsearch y analizar los datos a través de la creación de gráficos de barras, gráficos circulares, tablas, histogramas y mapas. Una vista de dashboard combina estos elementos visuales para luego puedan ser compartidos a través del navegador y brindar vistas analíticas en tiempo real de grandes volúmenes de datos para dar soporte a casos de uso como los siguientes:

* logging y analíticas de logs
* métricas de infraestructura y monitoreo de contenedores
* monitoreo de rendimiento de aplicaciones
* análisis y visualización de datos geoespaciales

1. Monitorear, administrar y asegurar una instancia del Elastic Stack a través de interfaz web.
2. Centralizar el acceso para soluciones integradas que se desarrollan en el Elastic Stack para aplicaciones de seguridad y búsqueda empresarial.

### Sintaxis

Para la búsqueda de datos, Kibana implementa el análisis de los datos de los índices que han sido previamente obtenidos cuando Logstash ingesta todos los datos obtenidos y les da formato a través de Elasticsearch, y a través de sus distintas opciones, visualizar los datos obtenidos. Esto se hace en el “dashboard” de Kibana, el panel que recopila todas las métricas personalizables para visualizar los resultados. Kibana cuenta con tres formas para la petición de datos: el uso de KQL (Kibana Query Language), DSL (el lenguaje que utiliza Elasticsearch para realizar peticiones) y Lucene. Se explica a continuación, la sintaxis de los das dos primeras opciones:

* KQL se utiliza para realizar búsquedas sencillas dentro de la interfaz de Kibana, ya que tiene más limitaciones que DSL. Algunas operaciones que se pueden realizar son:
  1. **Consultas de igualdad:** field\_name: value
  2. **Consultas de rango:** field\_name: >= value (>, <, >=, <=)
  3. **Consultas con comodines:** field\_name: \*value\*
  4. **Consultas de coincidencia de frases:** "frase"
  5. **Consultas booleanas:** field\_name: value OR field\_name: value (AND, OR, NOT)
  6. **Consultas con múltiples campos:** field\_name1: value1 OR field\_name2: value2
* DSL tiene una sintaxis en forma de JSON. Una consulta básica suele tener la siguiente forma:

GET /<índice>/\_search

{

"query": {

"match": {

"<campo>": "<valor>"

}

}

}

También, existe la posibilidad de utilizar la agregación a la hora de realizar consultas:

GET /<índice>/\_search

{

"aggs": {

"<nombre\_agregación>": {

"<tipo\_agregación>": {

"field": "<campo>"

}

}

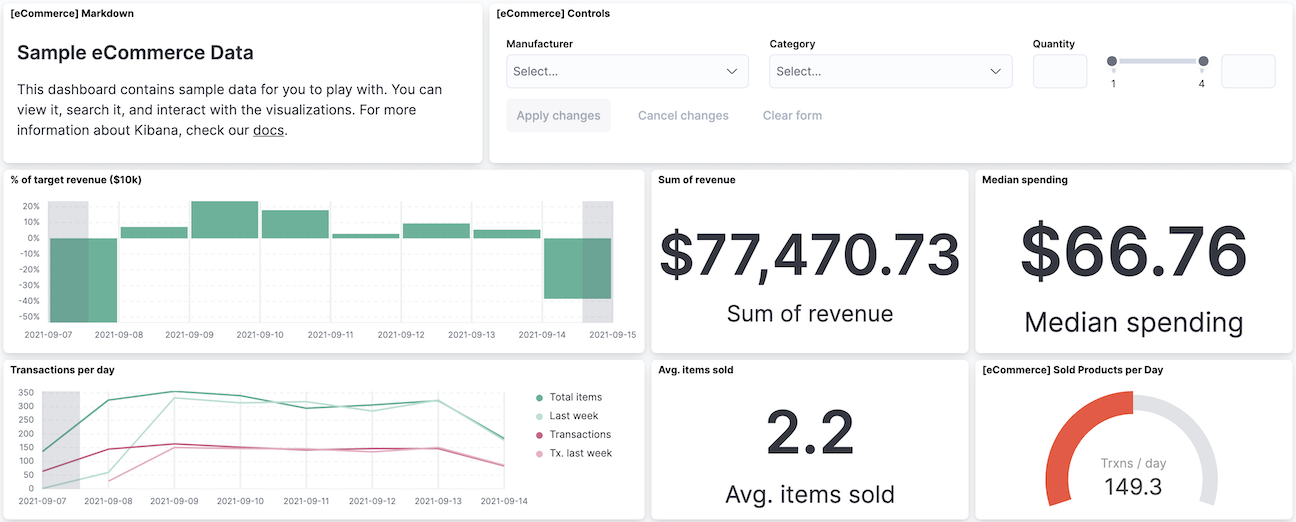
}

}

### uso

Para la creación de un dashboard, una vez se tengan los datos, se debe acceder a la aplicación de Kibana y seguir los siguientes pasos:

1. En el panel de navegación lateral, hacer clic en Dashboard
2. Create new dashboard
3. Add
4. Usa Add Panels para agregar visualizaciones y búsquedas guardadas al dashboard.



*Figura 6. Ejemplo de un Dashboard de Kibana [8]*

Además, Kibana cuenta con distintos plugins para mejorar las prestaciones de Kibana, donde se destacan los siguientes:

* Kibana Canvas: para la personalización de colores, imágenes, texto y pantallas dinámicas
* Kibana Visualize: implementa gráficos y grados estandarizados
* Kibana Maps: para el análisis de datos geoespaciales

## Go

Go es un lenguaje de programación desarrollado por Google, Ken Thompson, Rob Pike y Robert Griesemer, y creado en 2009, aunque no sería hasta 2012 cuando saldría su primera versión estable. Go es un lenguaje de programación de tipado estático y compilado, lo que implica que todas las variables, expresiones y operaciones tengan un tipo definido y que el código escrito se traduce a un código ejecutable mediante un archivo binario a través del compilador, y una vez compilado no es necesario que vuelva a ser interpretado. En el caso de Go, este compilador es “statically typed”, lo que significa que el código se pasa directamente a código máquina en el proceso de compilación. Además, la compilación de Go, es cruzada, y, por lo tanto, facilita la creación de ejecutables para diferentes sistemas operativos y arquitecturas.

Otras de sus características es que es concurrente, lo que implica que proporciona la posibilidad de trabajar con concurrencia, es decir, ser capaz de ejecutar varias tareas o procesos de manera independiente y parcialmente simultánea a través de hilos, que se ejecutan en paralelo.

Además, destaca por su simplicidad frente a otros lenguajes compilados como C o Java: Go utiliza inferencia implícita de tipos para así poder evitar la declaración explícita de variables que tienen lugar en C. [9].

Por otro lado, tiene también implementando un recolector de basura, este se encarga de administrar la memoria del programa, liberando la ya ocupada por objetos o datos que ya no son accesibles o utilizados por el programa, evitando así las fugas de memoria y simplificando la gestión de la memoria para los desarrolladores. Cuando un programa se ejecuta, utiliza la memoria para almacenar datos y objetos temporales. En algunos lenguajes de programación, como C o C++, los desarrolladores son los encargados de asignar y liberar manualmente la memoria utilizada por sus programas. Sin embargo, esto puede llevar a errores comunes como la asignación de memoria insuficiente o la liberación incorrecta de la memoria, lo que puede provocar errores de tiempo de ejecución o fugas de memoria.

En cambio, los lenguajes de programación que cuentan con un recolector de basura se encargan automáticamente de rastrear y administrar el uso de la memoria durante la ejecución del programa. El recolector de basura realiza un seguimiento de los objetos o datos que se han asignado dinámicamente en la memoria y determina cuándo un objeto ya no es accesible desde ninguna parte del programa. En ese momento, el recolector de basura libera automáticamente la memoria ocupada por ese objeto, marcándola como disponible para futuras asignaciones. Este proceso se realiza o de forma periódica o en momentos específicos durante la ejecución del programa. En el caso de Go, el recolector de basura está focalizado especialmente en reducir la latencia. Otra de sus características a destacar son las funciones de reflexión, que permiten a Go examinar y manipular la estructura durante el tiempo de ejecución.

También, es un lenguaje de programación orientado a objetos. Sin embargo, al contrario que la mayoría de estos, Go no dispone de herencia de tipos, clases y constructor. Los archivos escritos en go, tienen la extensión “.go”. Y estos se ejecutan con la sentencia “go run” y especificando como parámetro el nombre del archivo.

### Instalación

Para instalar Go, podemos siguiendo los siguientes pasos:

1. Extraer el archivo de instalación desde la web oficial: <https://go.dev/dl/>
2. Si es en un sistema Windows:
   1. Extraer el archivo en /usr/local/
   2. Agregar /usr/local/go/bin a las variables de entorno PATH:

### Sintaxis

Como se ha comentado previamente, una de las características que caben destacar de Go es la simplicidad de su sintaxis, para destacar por encima de otros lenguajes de programación como C.

En el caso de Go, contamos con las siguientes funcionalidades para escribir código:

#### Comentarios

Los comentarios son líneas de código que no compilan y sirven para dar contexto y describir el código. En Go, deben ir entre los símbolos “/\*” y “\*/”. Como por ejemplo: *\*/Esto es un comentario escrito en Go./\**.

#### Tipos de datos

En Go, existen varios tipos de datos que se utilizan para declarar variables, especificar tipos de parámetros y definir tipos de retorno en funciones. Para declarar una variable, se utiliza la palabra reservada var, seguida del nombre de la variable, y se acaba con el tipo de dato. Para asignarles un valor, después de la misma utilizamos un igual (=) y el valor que se le quiere asignar. Los tipos de variables más importantes son los siguientes [13]:

* **Enteros**: Representan números enteros sin parte decimal.
  + **int**: Entero con tamaño dependiente de la arquitectura (puede ser 32 o 64 bits).
  + **int8**: Entero de 8 bits con signo.
  + **int16**: Entero de 16 bits con signo.
  + **int32**: Entero de 32 bits con signo.
  + **int64**: Entero de 64 bits con signo.
  + **uint**: Entero sin signo con tamaño dependiente de la arquitectura.
  + **uint8**: Entero de 8 bits sin signo (byte).
  + **uint16**: Entero de 16 bits sin signo.
  + **uint32**: Entero de 32 bits sin signo.
  + **uint64**: Entero de 64 bits sin signo.
* **Flotantes**: Representan números de punto flotante con parte decimal.
  + **float32**: Número de punto flotante de 32 bits.
  + **float64**: Número de punto flotante de 64 bits (valor por defecto para números decimales).
* **Complejos**: Representan números complejos con partes real e imaginaria.
  + **complex64**: Número complejo de 64 bits (32 bits para cada parte).
  + **complex128**: Número complejo de 128 bits (64 bits para cada parte).
* **Booleanos**: Representan los valores de verdadero (true) o falso (false).
  + **bool**: Tipo de dato booleano.
* **Cadenas** de texto: Representan secuencias de caracteres.
  + **string**: Tipo de dato para cadenas de texto.
* **Caracteres**: Representan un único carácter Unicode.
  + **rune**: Tipo de dato para caracteres Unicode (32 bits).
* **Arreglos**: Representan una secuencia de elementos del mismo tipo, con tamaño fijo.
  + **[n]tipo**: Tipo de dato para arreglos de longitud n, donde tipo es el tipo de los elementos.
* **Slices**: Representan una vista flexible de una secuencia de elementos de un tipo.
  + **[]tipo**: Tipo de dato para slices, donde tipo es el tipo de los elementos.
* **Mapas**: Representan una colección de pares clave-valor.
  + **map[clave]valor**: Tipo de dato para mapas, donde clave y valor son los tipos de las claves y valores, respectivamente.
* **Punteros**: Representan la dirección de memoria de un valor existente en el programa.
  + **\*tipo**: Tipo de dato para punteros, donde tipo es el tipo apuntado.
* **Estructuras**: Representan una colección de campos con diferentes tipos.
  + **struct**: Tipo de dato para estructuras personalizadas definidas por el usuario.
* **Interfaces**: Representan un conjunto de métodos requeridos por un tipo concreto.
  + **interface**: Tipo de dato para interfaces personalizadas definidas por el usuario.

#### Palabras reservadas

El listado de palabras reservadas de Go es el siguiente:

* **break**: Se utiliza para finalizar una sentencia de bucle o un caso en una declaración de conmutador (switch).
* **default**: Se utiliza en una declaración de conmutador (switch) para especificar una acción predeterminada cuando ningún caso coincide.
* **func**: Sirve para declarar una función.
* **interface**: Se utiliza para definir una interfaz, que es un conjunto de métodos requeridos por un tipo concreto.
* **select**: Se emplea para realizar una selección no bloqueante entre varias operaciones de comunicación.
* case: Se utiliza en una declaración de conmutador (switch) para definir los casos posibles.
* **defer**: Se utiliza para programar la ejecución de una función hasta que la función actual regrese.
* **var**: Se utiliza para declarar una variable con un tipo específico.
* **go**: Se utiliza para iniciar la ejecución de una función.
* **map**: Sirve para declarar y crear un mapa, que es una estructura de datos que asocia claves con valores.
* **struct**: Se utiliza para definir una estructura, que es una colección de campos con diferentes tipos de datos.
* **chan**: Se emplea para declarar y crear un canal, que es una tubería de comunicación utilizada para la sincronización y comunicación entre goroutines.
* **else**: Se utiliza en una declaración condicional para especificar un bloque de código que se ejecutará si la condición es falsa.
* **goto**: Sirve para saltar a una etiqueta específica en el programa.
* **package**: Se utiliza para declarar el paquete al que pertenece el archivo de código actual.
* **switch**: Se emplea para evaluar una expresión y ejecutar diferentes bloques de código en función de los casos que coincidan.
* **const**: Sirve para declarar una constante, es decir, una variable cuyo valor no puede cambiar durante la ejecución del programa.
* **if**: Se utiliza para realizar una comprobación condicional y ejecutar un bloque de código si la condición es verdadera.
* **range**: Se emplea para iterar sobre elementos de una estructura de datos, como un arreglo, un mapa o un canal.
* **type**: Sirve para declarar un nuevo tipo o para asignar un nombre a un tipo existente.
* **continue**: Se utiliza para saltar a la siguiente iteración de un bucle sin ejecutar el resto del código dentro del bucle en esa iteración.
* **import**: Se utiliza para importar paquetes externos necesarios para el programa.
* **return**: Sirve para finalizar la ejecución de una función y devolver un valor opcional.

#### Declaración de variables

Para declarar una variable en Go, se debe realizar utilizando primero la palabra reservada var, seguida del nombre que se le quiere dar a una variable y finalmente se le debe indicar el tipo de dato. Además, se puede también inicializar a un valor, como, por ejemplo: “var edad int = 7”.

#### Declaración de funciones

Para declarar una función en Go, se debe utilizar la palabra reservada “*func*”, estas siguen la siguiente estructura:

Texto

Descripción generada automáticamente*Figura 7. Estructura de una función en Go*

Donde la sintaxis correspondiente es la siguiente:

* **func**: Es la palabra reservada que indica que se está declarando una función.
* **nombreFuncion**: Es el nombre que se le asigna a la función. Debe seguir las convenciones de nombres de identificadores en Go.
* (parametro1 tipo, parametro2 tipo): Aquí especificas los parámetros de la función junto con sus tipos. Puedes tener cero o más parámetros, separados por comas.
* **tipoRetorno**: Es el tipo de dato que devuelve la función. Si la función no devuelve ningún valor, se puede omitir este tipo o utilizar *void* (que en Go se representa con “void”).
* **{}**: Entre las llaves se coloca el cuerpo de la función, donde se escribe el código que se ejecutará cuando se invoque la función.
* **return valorRetorno**: Si la función tiene un tipo de retorno especificado, se debe utilizar la palabra reservada return seguida del valor que deseas devolver. Si la función no tiene un tipo de retorno, se puede omitir esta línea o utilizar return.

#### Bucle For

El bucle “for” se utiliza para ejecutar un bloque de código repetidamente hasta que se cumpla una condición. En Go existen tres formas distintas de declarar el bucle for:

##### For “básico”

La estructura es la siguiente:

Texto

Descripción generada automáticamente*Figura 8. Estructura de un bucle for en Go*

Donde los parámetros son los siguientes:

* **inicialización**: Se inicializa una variable de control antes de que comience el bucle. Esta parte se ejecuta solo una vez antes de la primera iteración.
* **condición**: Es una expresión booleana que se evalúa antes de cada iteración. Si la condición es verdadera, el bucle continúa ejecutándose; si es falsa, el bucle se detiene.
* **post-iteración**: Aquí se actualiza la variable de control o se realiza cualquier otra acción después de que se completa cada iteración. Se ejecuta al final de cada iteración.

##### For Range

El bucle “for range” se utiliza para iterar sobre una estructura de datos, como un arreglo, un slice o un mapa.

La estructura es la siguiente:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media*Figura 9. Estructura de un bucle for-range en Go.*

Donde los parámetros son los siguientes:

* **índice**: Es el índice (o clave) de la posición actual en la estructura de datos.
* **valor**: Es el valor correspondiente al índice actual.
* **colección**: almacena la lista de valores sobre los que se iteran

##### For Continue

El bucle “for – continue” se utiliza para realizar iteraciones condicionales y saltar a la siguiente iteración sin ejecutar el resto del código dentro del bucle en esa iteración.

La estructura es la siguiente:

Texto

Descripción generada automáticamente*Figura 10. Estructura de un bucle for-continue en Go.*

Donde los parámetros que recibe son los siguientes:

* **condición**: Es una expresión booleana que se evalúa antes de cada iteración.
* **condiciónDeSalto**: Es una expresión booleana que se evalúa dentro del bucle. Si es verdadera, el bucle salta a la siguiente iteración sin ejecutar el resto del código en esa iteración.

#### Estructura de control If

Al igual que en la mayoría de los lenguajes de programación la sentencia “*if*” se utiliza para realizar evaluaciones condicionales y ejecutar bloques de código basados en el resultado de esas evaluaciones.

La estructura es la siguiente:

Texto

Descripción generada automáticamente*Figura 11. Estructura de una sentencia if en Go.*

## RUST

A continuación, se presenta el otro lenguaje de programación sobre el que se van a calcular las prestaciones sobre las que se basa la hipótesis de este trabajo. Una definición exacta de lo que presenta este lenguaje se encuentra en su propia web: “Rust es increíblemente rápido y eficiente con la memoria: sin run time ni recolector de basura, puede sustentar servicios de rendimiento crítico, ejecutarse en dispositivos integrados, y colaborar con otros lenguajes fácilmente.” [11]

Su desarrollo comenzó en 2006 por el ingeniero informático Graydon Hoare, y actualmente hay 2,8 millones de programadores escribiendo en Rust, y empresas como Microsoft o Amazon lo consideran clave para su futuro. La plataforma de chat Discord utilizó Rust para acelerar su sistema; Dropbox lo emplea para sincronizar archivos con tu ordenador; y Cloudflare lo utiliza para procesar más del 20% de todo el tráfico de Internet [12]. Su primera versión estable se lanzó en 2015.

Sus principales características radican en los siguientes puntos:

1. **Seguridad de memoria**: Rust se enorgullece de su enfoque en la seguridad de memoria sin sacrificar el rendimiento. El lenguaje incorpora conceptos como son los "prestamos") y "referencias" para garantizar que no haya errores de acceso a memoria, un ejemplo de ello puede ser eliminar la referencia de punteros nulos. Gracias a esto se evitan problemas comunes como fugas de memoria o corrupción de datos.
2. **Propiedad única**: En Rust, cada valor tiene un propietario único y solo puede haber un propietario a la vez. Cuando un valor sale del ámbito, se libera automáticamente. Esto permite prevenir errores comunes de gestión de memoria en tiempo de compilación y elimina la necesidad de un recolector de basura en tiempo de ejecución.
3. **Concurrencia segura**: Rust proporciona concurrencia sin problemas de seguridad. Su modelo de hilos seguro asegura que no haya problemas como carreras de datos (data races) al compartir datos entre hilos. Esto se logra mediante el uso del sistema de tipos y el concepto de "mutabilidad" controlada.
4. **Simplicidad en la sintaxis**: Rust tiene una sintaxis clara y concisa que facilita la lectura y escritura de código.
5. **Soporte multiplataforma**: Rust puede compilar para diferentes plataformas y arquitecturas, lo que lo hace adecuado para el desarrollo de aplicaciones de sistemas y software embebido.

Para ejecutar código en Rust, se debe crear primero un proyecto nuevo en el directorio deseado con el comando “cargo new” pasándole como parámetro el nombre del proyecto. Para crear el sistema de directorios necesarios para la creación del proyecto, entre los que se incluyen el fichero “Cargo.toml” que es el fichero de configuración del proyecto, donde se incluyen la información general sobre el proyecto, o dependencias entre otras cosas. A su vez, se genera también el directorio src con el proyecto “main.rs” para su edición. Para ejecutarlo, se debe hacer desde el directorio en el que se encuentre el fichero de configuración, y se hace con la sentencia “cargo run”

### Instalación

Para la instalación de Rust se debe acceder a la página oficial de Rust (<https://www.rust-lang.org/es>) para descargar el instalador en sistemas Windows y una vez instalado se debe ejecutar el siguiente comando por terminal: %USERPROFILE%\.cargo\env.

Para sistemas Linux o Mac se puede instalar con las siguientes dos sentencias:

1. curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
2. source $HOME/.cargo/env

### Sintaxis

Como se ha comentado previamente, la sintaxis de Rust está en gran parte inspirada por el lenguaje de programación C, pero también incorpora características modernas que lo hacen más expresivo.

#### Comentarios

Para escribir comentarios en Rust, existen dos maneras:

1. Comentarios en línea: Se precede el comentario con los caracteres *//*
2. Comentarios en bloque: Se agrupa el párrafo entre los caracteres */\** y  *\*/*

#### Tipos de datos

En Rust, los datos se dividen en primitivos y compuestos:

* **Tipos Primitivos:**
  + **Integer**: Representan números enteros y pueden ser con signo o sin signo. Los enteros sin signo se representan con la letra u (por ejemplo, u8, u16) y los enteros con signo se representan con la letra i (por ejemplo, i8, i16). También existen enteros de tamaño específico, como isize y usize, que dependen de la arquitectura del sistema.
  + **Float**: Representan números con decimales. Los tipos de punto flotante son f32 (punto flotante de precisión simple) y f64 (punto flotante de precisión doble).
  + **Boolean**: Representa un valor de verdad y puede tener dos posibles valores: true o false.
  + **Character**: Representa un solo carácter y se denota con comillas simples, por ejemplo, 'a' o '7'.
* **Tipos Compuestos:**
  + **Tuple**: Una colección ordenada de elementos con tipos diferentes. Las tuplas se definen entre paréntesis y sus elementos están separados por comas.
  + **Array**: Una colección de elementos del mismo tipo y con un tamaño fijo conocido en tiempo de compilación. Los arrays se definen con corchetes y su tamaño debe ser conocido en tiempo de compilación.
  + **Slice**: Una vista no poseedora de una sección de un arreglo. Los slices son una forma de referencia a una parte de un arreglo y permiten trabajar con porciones de este sin tener que copiar los datos.
  + **Struct**: Un tipo de datos personalizado que permite agrupar varios campos con diferentes tipos en una sola entidad.
  + **Enum**: Un tipo de datos que permite definir un conjunto de posibles variantes.
  + **Reference**: Permite hacer referencia a valores existentes sin tomar posesión de ellos. Se utilizan con el operador *&* y son esenciales para garantizar la seguridad de memoria en Rust.

#### Palabras Reservadas

El listado de palabras reservadas en Rust es el siguiente [14]:

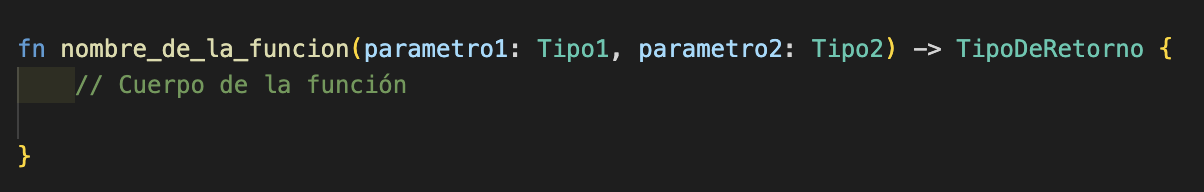
* **as**: Se utiliza para realizar conversiones de tipo explícitas. Por ejemplo, para convertir un valor de un tipo a otro compatible.
* **let**: Se utiliza para declarar una variable
* **async**: Se utiliza para definir funciones asincrónicas que pueden usar la sintaxis await para esperar que se resuelvan futuros.
* **await**: Se utiliza dentro de funciones asincrónicas para esperar que se complete la ejecución de un futuro antes de continuar.
* **break**: Se utiliza para salir de un bucle (for, while, loop) antes de que se complete su ejecución normal.
* **const**: Se utiliza para declarar constantes cuyos valores no cambian durante la ejecución del programa.
* continue: Se utiliza para interrumpir una iteración del bucle actual y pasar a la siguiente iteración.
* **crate**: Hace referencia al crate raíz actual, que es el conjunto de módulos que se compilan en un binario o librería.
* **dyn**: Se utiliza para especificar que un tipo es una referencia dinámica a un trait.
* **else**: Se utiliza en combinación con if para definir un bloque de código que se ejecuta cuando la condición de if no es verdadera.
* **enum**: Se utiliza para definir un tipo de dato enumerado que puede tener múltiples variantes.
* **extern**: Se utiliza para interactuar con código externo a Rust, como funciones escritas en otros lenguajes.
* false: Representa el valor booleano false.
* **fn**: Se utiliza para definir una función.
* **for**: Se utiliza para iterar sobre una colección (arreglo, rango, iterador, etc.).
* **if**: Se utiliza para definir una estructura condicional. Se ejecuta un bloque de código si la condición es verdadera.
* **impl**: Se utiliza para implementar métodos y funciones asociadas a un tipo.
* **in**: Se utiliza en algunos contextos para denotar una relación de pertenencia, como en un bucle for.
* **let**: Se utiliza para declarar y asignar variables.
* **loop**: Se utiliza para crear un bucle infinito.
* **match**: Se utiliza para realizar patrones y realizar diferentes acciones según el valor de una variable.
* **mod**: Se utiliza para definir un módulo en Rust.
* **move**: Se utiliza para capturar el contexto actual y moverlo a un closure.
* **mut**: Se utiliza para denotar que una variable es mutable, es decir, su valor puede cambiar.
* **pub**: Se utiliza para hacer públicos los elementos (funciones, estructuras, etc.) dentro de un módulo para que puedan ser utilizados desde otros módulos.
* **ref**: Se utiliza para crear referencias explícitas.
* **return**: Se utiliza para devolver un valor desde una función.
* **Self**: Se utiliza para hacer referencia al tipo actual dentro de una implementación.
* **self**: Se utiliza dentro de una implementación para referenciar el objeto en sí mismo.
* **static**: Se utiliza para declarar variables globales con tiempo de vida estático.
* **struct**: Se utiliza para definir una estructura de datos.
* super: Se utiliza para hacer referencia al módulo padre desde dentro de un módulo.
* **trait**: Se utiliza para definir un trait, que es una interfaz que define comportamientos que los tipos pueden implementar.
* **true**: Representa el valor booleano true.
* **type**: Se utiliza para definir un alias de tipo.
* **unsafe**: Se utiliza para indicar que un bloque de código contiene operaciones inseguras que el compilador no puede verificar en términos de seguridad de memoria.
* **use**: Se utiliza para importar módulos, funciones o elementos desde otros módulos para poder usarlos en el alcance actual.
* **where**: Se utiliza en asociación con impl para especificar condiciones o restricciones en las implementaciones de traits.
* **while**: Se utiliza para crear un bucle que se ejecuta mientras se cumpla una condición.

#### Declaración de variables

En Rust, tenemos las variables mutables e inmutables, y estas se declaran de la siguiente manera:

* **Variables inmutables**: Su valor no puede cambiar. Se utiliza la palabra reservada “let”, seguida del nombre de la variable. Adicionalmente, se le puede asignar un valor al declarar que se indicará después del “=”. Ejemplo: let edad = 7
* **Variables mutables**: Su valor puede cambiar. Se utiliza la palabra reservada “let”, seguida de la palabra reservada mut seguida del nombre de la variable. Adicionalmente, se le puede asignar un valor al declarar que se indicará después del “=”. Ejemplo: let mut edad = 7

#### Declaración de funciones



*Figura 12. Estructura de una declaración de función en Rust.*

#### BUCLE FOR

La estructura de un bucle for en Rust es la siguiente:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

*Figura 13. Estructura de un bucle for en Rust*

#### Bucle While

La estructura de bucle while es la siguiente. Este se ejecuta mientras la condición que se evalúa sea verdadera:

Texto

Descripción generada automáticamente

*Figura 14. Estructura de un bucle for en Rust*

#### Estrucura de control if

La sentencia de control if tiene la siguiente estructura:

Texto

Descripción generada automáticamente

*Figura 15. Estructura de un bucle for en Rust*

# resultados

Lo primero es presentar los tests que se han utilizado para los tiempos de ejecución de Go y Rust. Estos son los siguientes: una prueba que calcula la velocidad de ejecución de distintas operaciones matemáticas, una prueba que se encarga de desplegar contenedores y posteriormente eliminarlos, y finalmente un test que se encarga de realizar peticiones HTTP GET y POST a un servidor local.

Cabe destacar, que el resultado de las pruebas se debe procesar usando ELK para mostrar los resultados de una manera visualmente atractiva. Para ello, se deben configurar los sistemas involucrados en este proceso

## Operaciones matemáticas

A continuación, se presentan las pruebas planteadas en Go y Rust para medir los tiempos de cálculo de operaciones matemáticas. Estos tienen la misma funcionalidad, y se diferencian por la sintaxis empleada para su desarrollo. Estos se encuentran en el Anexo 1: “Tests de Operaciones Matemáticas”. La funcionalidad es la siguiente:

Primero, se crea un nuevo archivo “.log” para escribir los resultados obtenidos del procesamiento de las pruebas, posteriormente se ejecutan las pruebas de rendimiento para las distintas operaciones matemáticas (Suma, Resta, Raíz Cuadrada, Multiplicación, División, Seno, Coseno).

Para cada operación se fija el número de repeticiones para obtener la cantidad deseada de muestras y se crea un bucle para ejecutar la operación, midiendo el tiempo de ejecución en cada iteración. Se captura el instante de inicio con antes de realizar la operación y se ejecuta la función correspondiente, para a continuación calcular el tiempo transcurrido y almacenándolo en una lista. Luego, se registra en el archivo de registro el nombre de la operación, el número de repetición y el tiempo transcurrido en formato de cadena de caracteres.

Todos estos datos se almacenan en el fichero mencionado anteriormente para que luego puedan ser procesados con ELK a través de los ficheros “logstash.conf” correspondientes, donde se incluye la información necesaria para enviar los datos al servidor de ElasticSearch. Estos se incluyen también en el anexo 1.

CONTINUAR CON ELK

## DESPLIEGUE DE CONTENEDORES

A continuación, se presentan las pruebas planteadas en Go y Rust. Estos tienen la misma funcionalidad, y se diferencian por la sintaxis empleada para su desarrollo. Estos se encuentran en el Anexo 2: “Tests de despliegue de contenedores”. La funcionalidad es la siguiente:

Primero se crea el archivo “.log” correspondiente para almacenar los resultados de la ejecución los tests diseñados en los lenguajes de Go y Rust.

Posteriormente se inicializa un bucle indicando el número de contenedores deseados para desplegar, y tras entrar en el bucle, se captura el tiempo actual. A continuación, se ejecuta el comando de ejecución de creación de un contenedor Docker con la imagen "nginx:latest", y se evalúa si la ejecución del comando ha sido correcta o no. De haberlo sido, se obtiene el ID correspondiente de la creación y se calcula el tiempo transcurrido durante su creación, mostrándolo por consola y a su vez almacenándolo en el fichero “.log” correspondiente. Posteriormente, se detiene el contenedor creado. Si hubo algún error durante la ejecución del comando, se captura el error y se maneja.

Finalmente, se procesa el contenido de estos ficheros en ELK a través de logstash por mediación del fichero de configuración “logstash.conf”, enviando los resultados al servidor de ElasticSearch para que estos pueden ser procesados. Estos ficheros de configuración se encuentran también en el Anexo 2.

CONTINUAR CON ELK

## Peticiones HTTP

Las dos peticiones que sobre las cuales se van a medir los tiempos son GET y POST. Para ello, en este caso se va a hacer uso de Apache JMeter, donde se lanzarán peticiones HTTP y se recogerán los resultados, para posteriormente procesarlos con ELK.

# conclusiones y líneas futuras

## Conclusiones

…

## Líneas futuras

…

# bibliografía

[1] Pérez, J. (2018, August 7). ELK Stack: ¿Qué es y cómo implementarlo fácilmente mediante DOCKER? Encora. [https://www.encora.com/es/blog/elk-stack-que-es-y-como-implementarlo-facilmente-mediante-docker#:~:text=ELK%20es%20la%20combinación%20de,de%20los%20datos%20de%20registro.](https://www.encora.com/es/blog/elk-stack-que-es-y-como-implementarlo-facilmente-mediante-docker#:~:text=ELK%20es%20la%20combinaci%C3%B3n%20de,de%20los%20datos%20de%20registro.)

[2] El ELK Stack: de los creadores de Elasticsearch. (n.d.). Elasticsearch Platform — Find real-time answers at scale | Elastic. <https://www.elastic.co/es/what-is/elk-stack>

[3] Introducción a Elasticsearch. (2022, December 30). aprenderbigdata.com. <https://aprenderbigdata.com/elasticsearch/>

[4] Arquitectura de ElasticSearch | KeepCoding Bootcamps. (2022, April 20). KeepCoding Bootcamps. <https://keepcoding.io/blog/arquitectura-elasticsearch/>

[5] ¿Qué es Logstash? + Ejemplo práctico de uso. (2020, May 20). davinciitii. <https://davinciti.com/que-es-logstash-ejemplo-practico-de-uso/>

[6] LT DATA CHANNEL. (2021, January 8). ✅ [LOGSTASH] - QUÉ es LOGSTASH, CÓMO INSTALARLO | DOS EJEMPLOS PRACTICOS 🥇 [Video]. YouTube. <https://www.youtube.com/watch?v=ENWa_KnJahY>

[7] ELK Team. (n.d.). ¿Qué es Kibana? Elastic. <https://www.elastic.co/es/what-is/kibana>

[8] Dashboard and visualizations | Kibana Guide [8.8] | Elastic. (n.d.). Elasticsearch Platform — Find real-time answers at scale | Elastic. <https://www.elastic.co/guide/en/kibana/current/dashboard.html>

[9] Lenguaje de Programación Go y sus Características. (2023, February 15). KeepCoding Bootcamps. <https://keepcoding.io/blog/lenguaje-de-programacion-go-caracteristicas/>

[10] Ruelas, Uriel. “01. Go to Go: Iniciando Go for the Lulz.” 01. Go to Go: Iniciando Go for the Lulz, 29 Apr. 2016, codingornot.com/01-go-to-go-iniciando-go-for-the-lulz.

[11]*Rust, el lenguaje de programación*. (n.d.). Rust Programming Language. <https://www.rust-lang.org/es>

[12] JINHWA JANG. (2023, March 3). *Breve historia de Rust, el lenguaje de programación que ha destronado a C*. MIT Technology Review. [https://www.technologyreview.es//s/15106/breve-historia-de-rust-el-lenguaje-de-programacion-que-ha-destronado-c](https://www.technologyreview.es/s/15106/breve-historia-de-rust-el-lenguaje-de-programacion-que-ha-destronado-c)

[13] Uriel Ruelas. (2016, May 13). *02. Go to go: sintaxis, tipo de datos y palabras reservadas*. Blog de tecnologías de la información. <https://codingornot.com/02-go-to-go-sintaxis-tipo-de-datos-y-palabras-reservadas>

[14] *Rust - Apéndice A:Palabras clave La siguiente lista contiene palabras clave que están reservadas para su uso futuro por el lenguaje Rust.* (n.d.). Runebook.dev. <https://runebook.dev/es/docs/rust/book/appendix-01-keywords>

# anexo a: aspectos ÉTICOS, económicos, sociales y ambientales

El apartado “Requisitos de las acreditaciones internacionales EUR-ACE y ABET” de la Normativa de TFT de la ETSIT-UPM establece que “*La memoria del TFT del GITST, GIB y MUIT, y en general la de aquellas titulaciones que hayan obtenido o para las que se desee solicitar una acreditación internacional EUR-ACE o ABET, debe mostrar conciencia de la responsabilidad de la aplicación práctica de la ingeniería, el impacto social y ambiental, el compromiso con la ética profesional, la responsabilidad y las normas de la aplicación práctica de la ingeniería, así como sobre las prácticas de gestión de proyectos, gestión, y control de riesgos, entendiendo sus limitaciones*”.

Este anexo obligatorio del TFT tendrá un carácter sintético con los siguientes apartados:

## A.1 iNTRODUCCIÓN

Breve descripción del contexto del proyecto, objetivos, necesidades que pretende cubrir o problemas que pretende resolver, centrándose en su relación con los temas sociales, económicos, éticos, legales y/o ambientales que se hayan identificado.

## A.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

Síntesis del trabajo realizado en la fase 2, de selección y descripción de impactos. Presentar y justificar las conclusiones a las que se haya llegado sobre cuáles son los asuntos más relevantes relacionados con la sostenibilidad social, económica o ambiental, así como los principales grupos de interés identificados y que se han considerado en los análisis posteriores.

## A.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS

Síntesis del trabajo de análisis realizado.

## A.4 CONCLUSIONES

Valorar el proyecto desde un punto de vista ético, social, económico y medioambiental y justificar si el uso de criterios de sostenibilidad ha aportado o puede aportar valor añadido al proyecto.

# anexo b: presupuesto económico

El apartado “Requisitos de las acreditaciones internacionales EUR-ACE y ABET” de la Normativa de TFT de la ETSIT-UPM establece que “*La memoria del TFT del GITST, GIB y MUIT, y en general la de aquellas titulaciones que hayan obtenido o para las que se desee solicitar una acreditación internacional EUR-ACE o ABET, … debe incluir un presupuesto económico*”. A modo de ejemplo, la tabla del presupuesto de un proyecto podría ser la siguiente:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **COSTE DE MANO DE OBRA (coste directo)** | | | | | **Horas** | | **Precio/hora** | **Total** |
|  | | | | | 300 | | 15 € | **4.500 €** |
|  | | | |  |  | |  |  |
| **COSTE DE RECURSOS MATERIALES (coste directo)** | | | | **Precio de compra** | **Uso en meses** | | **Amortización (en años)** | **Total** |
| Ordenador personal (Software incluido)....... | | | | 1.500,00 € | 6 | | 5 | 150,00 € |
| Impresora láser | | | | 500,00 € | 6 | | 5 | 50,00 € |
| Otro equipamiento | | | |  |  | |  |  |
|  | | | |  |  | |  |  |
| **COSTE TOTAL DE RECURSOS MATERIALES** | | | | | | | | **200,00 €** |
|  | |  | |  | |  | |  |
| **GASTOS GENERALES (costes indirectos)** | | 15% | | sobre CD | | | | **705,00 €** |
| **BENEFICIO INDUSTRIAL** | | 6% | | sobre CD+CI | | | | **324,30 €** |
|  | |  | |  | |  | |  |
| **MATERIAL FUNGIBLE** | |  | |  | |  | |  |
| Impresión | | | | | | | | **100,00 €** |
| Encuadernación | | | | | | | | **300,00 €** |
|  |  | |  | | |  | |  |
| **SUBTOTAL PRESUPUESTO** | | | | | | | | **6.129,30 €** |
| **IVA APLICABLE** | | | | | | | 21% | **1.287,15 €** |
|  |  | |  | | | |  |  |
| **TOTAL PRESUPUESTO** | | | | | | | | **7.416,45 €** |

Esta tabla podría ser rellenada mediante una sencilla hoja de cálculo como la siguiente:



# anexo C: Elementos del desarrollo

## Anexo 1. Tests de Operaciones Matemáticas

Texto

Descripción generada automáticamente con confianza baja Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

*Figura 16. Test de operaciones matemáticas en Go*

Texto

Descripción generada automáticamente

*Figura 17. Tests de operaciones matemáticas en Rust*

*AÑADIR LOGSTASH.CONF DE AMBOS*

## Anexo 2. Tests de despliegues de contenedores

Texto

Descripción generada automáticamente

*Figura 20. Tests de despliegue de contenedores en Go*

*Texto

Descripción generada automáticamente*

*Figura 21. Tests de despliegue de contenedores en Rust*

## Anexo 3. Tests de peticiones http