INTELIGENCIA SOBRE BLOCKCHAIN USANDO ELASTIC STACK

Autor:
Mª Vanesa García Tubío

Nombre del tutor:
Rafael Pérez Reyes
Profesor responsable de la asignatura:
Victor García Font

Máster en Ciberseguridad y Privacidad

Sistemas de Blockchain

Enero 2023

FICHA DEL TRABAJO FINAL

Título del trabajo:	Inteligencia sobre Blockchain usando Elastic Stack
Nombre del autor:	M.ª Vanesa García Tubío
Nombre del consultor/a:	Rafael Páez Reyes
Nombre del PRA:	Victor García Font
Fecha de entrega:	01/2023
Titulación o programa:	Máster en Ciberseguridad y Privacidad
Área del Trabajo Final:	Sistemas de Blockchain
Idioma del trabajo:	Castellano
Palabras clave:	Inteligencia Blockchain ElasticStack

Resumen del Trabajo:

El aumento en la cantidad de datos que se generan actualmente hace que sea más necesario que nunca buscar estrategias de análisis de esos datos para obtener información de la misma. Este trabajo se centra en la creación de los elementos necesarios para ELK Stack para el tratamiento de transacciones Bitcoin.

Se ha visto que existen herramientas que permiten trabajar con datos de Blockchain, pero hay un sesgo importante, o bien son herramientas muy complejas y costosas o bien se centran en pequeñas áreas. Además, entre las gratuitas, se han visto muchas herramientas orientadas al tratamiento de transacciones Ethereum, pero menos y más sencillas para Bitcoin.

Se propone la creación de los elementos necesarios para trabajar con los datos de la red de Bitcoin en ELK Stack. Para llevar a cabo este objetivo será necesario analizar que elementos son necesarios y como deben realizarse para integrarse en el entorno de ELK Stack.

El proyecto cumplirá su objetivo cuando se consiga cargar datos, visualizarlos y analizarlos para obtener la información necesaria para responder a preguntas.

Abstract:

The increase in the amount of data that is currently generated makes it more necessary than ever to seek data analysis strategies to obtain information from it. This work focuses on the creation of the necessary elements for the ELK Stack for the treatment of Bitcoin transactions.

It has been seen that there are tools that allow working with Blockchain data, but there is an important bias, either they are very complex and expensive tools or they focus on small areas. In addition, among the free ones, many tools oriented to the treatment of

Ethereum transactions have been seen, but fewer and simpler for Bitcoin.

The creation of the necessary elements to work with the data of the Bitcoin network in ELK Stack is proposed. To carry out this objective, it will be necessary to analyze what elements are necessary and how they must be carried out to integrate into the ELK Stack environment.

The project will achieve its goal when it is possible to load data, visualize it and analyze it to obtain the information necessary to answer questions.

Índice

Sumario

1 Introducción....10
1.1 Contexto y justificación del trabajo....10
1.2 Conceptos previos....11
1.3 Estado del arte....12
1.4 Herramientas y tecnologías....13
1.5 Objetivos del trabajo....14
1.6 Impacto en sostenibilidad, ético-social y de diversidad....15
1.7 Enfoque y método seguro....15
1.8 Breve sumario de productos obtenidos....16
1.9 Breve descripción de los otros capítulos de la memoria....16
2 Planificación y presupuestos....17
2.1 Planificación temporal....17
2.2 Planificación en horas....18
2.3 Plan de costes....18
2.3.1. Introducción....18
2.3.2. Estimación de costes....19
2.4 Riesgos del proyecto....22
2.4.1. Catálogo de riesgos....22
3 Creación del entorno de trabajo....25
3.1 Herramientas y tecnologías para instalaciones....25
3.2 Docker y Docker Compose....26
3.3 Elastic Stack....29
3.3.1. Elasticsearch....30
3.3.2. Logstash....30
3.3.3. Kibana....31
3.3.4. Beats....31
3.4 Crear entorno....31
3.4.1. Pasos previos....31
3.4.2. Ejecutar Elasticsearch....32
3.4.3. Carga de datos de pruebas....37
3.4.4. Carga de datos de Ethereum....40
4 Análisis de carga de datos....46
4.1 Introducción....46
4.2 Análisis de Beat de Ethereum....46
4.2.1. Herramientas para el desarrollo del Beat....46
4.2.2 Análisis del Beat de Ethereum....47
4.3 Carga de datos en ELK. Alternativas al Beat....51
4.3.1. Logstash....51
4.3.2. Parsear BD de Bitcoin....51
4.3.3. Crear herramienta propia....52
4.3.4. Utilizar API de Bitcoin....52
4.4 Crear red local de Bitcoin....53

4.4.1. Testnet de Bitcoin....54
4.4.2. Crear nodo de Bitcoin....55
5 Carga y tratamiento de datos....62
5.1 Carga de datos de Bitcoin en ELK....62
5.1.1. Añadir Logstash al entorno....62
5.1.2. Creación de pipeline de prueba....63
5.1.3. Cargar datos de Bitcoin....64
5.2 Tratamiento de datos....70
5.2.1. Filtrado de datos....70
5.2.2. Consultar información en varios índices....71
6 Conclusiones y trabajos futuros....79
6.1 Conclusiones....79
6.1.1. Introducción....79
6.1.2. Resultados....80
6.2 Trabajos futuros....80
7 Glosario....82
8 Bibliografía....83
9 Anexos....88
9.1 Anexo I....88
9.2 Anexo II: Docker-compose utilizado en fase de creación del entorno de trabajo...89
9.3 Anexo III: Docker-compose utilizado en fase de análisis....90
9.4 Anexo IV: Pasos a seguir para ejecutar el entorno final....91
9.4.1 Hardware empleado....91
9.4.2. Crear nodo de local de Bitcoin....91
9.4.3. Instalar Docker Desktop....92
9.4.4. Ejecutar contenedores ELK....93
9.4.5. Instalar el plugin de logstash para blockchain....93
9.4.6. Configurar el plugin de blockchain....94

Índice de figuras

Figura 1: Blockchain: cadena de bloques.....11
Figura 2: Arquitectura ELK.....14
Figura 3: Planificación temporal.....17
Figura 4: Salario medio anual en España para analista programador y jefe de proyecto.. 19
Figura 5: Máquinas virtuales vs contenedores Docker.....27
Figura 6: Ejemplo de yaml para docker-compose.....28
Figura 7: Arquitectura ELK.....30
Figura 8: Docker desktop.....32
Figura 9: Primera ejecución de Elasticsearch en Docker Desktop.....33
Figura 10: Pagina de inicio de Kibana.....33
Figura 11: Kibana: Analytics.....34
Figura 12: Kibana Observability.....35
Figura 13: Kibana Security.....36
Figura 14: Kibana. Sample_data_flights. Index.....37
Figura 15: Kibana. Sample_data_flights. Data View.....37
Figura 16: Kibana. Sample_data_flights. Saved Objects.....38
Figura 17: Kibana. Sample_data_flights. Dashboards.....38
Figura 18: Kibana. Sample_data_flights. Dashboards vista.....39
Figura 19: Ganache. Datos de Servidor.....40
Figura 20: Ganache. Vista de Bloques.....41
Figura 21: *EtherBeat*: ejecución de contenedor.....42
Figura 22: EtherBeat: logs de ejecución.....42
Figura 23: Ejecución de docker-elk con Etherbeat.....43
Figura 24: Docker-elk. Etherbeat logs.....43
Figura 25: Docker-elk. Kibana.....44
Figura 26: Kibana. Carga de datos. Index Management.....45
Figura 27: Configuración de EtherBeat.....47
Figura 28: Código fuente de EtherBeat.....47
Figura 29: Custom Beats Deprecation.....48
Figura 30: Custom Beat generator code no disponible.....48
Figura 31: Crear un Beat. Actualización de deprecated.....49
Figura 32: Crear un Beat. Problemas con la información oficial.....49
Figura 33: Limitaciones API Bitcoin.....52
Figura 34: Tamaño de la BD de Bitcoin.....53
Figura 35: Datos de la red de Tesnet.....54
Figura 36: Bicoin.conf.....55
Figura 37: Ejecutables de BitcoinCore.....55
Figura 38: Ejecucion de BitcoinCore.....56
Figura 39: Log de sincronización de BitcoinCore.....56
Figura 40: Resultado de getblockchaininfo.....57
Figura 41: Resultado de getblockcount.....57
Figura 42: Resultado de crear dirección en Testnet.....57
Figura 43: Resultado de getunconfirmedbalance.....58
Figura 44: Obtener saldo en la address.....58
Figura 45: Resultado de consultar saldo tras transferencia.....58

Figura 46: Resultado de obtener la lista de transacciones.....59
Figura 47: Resultado de obtener el balance confirmado.....59
Figura 48: Contenido del fichero .env de btc-rpc-explorer.....60
Figura 49: Creación de imagen docker de btc-rpc-explorer.....60
Figura 50: Ejecución de contenedor docker de btc-rpc-explorer.....61
Figura 51: Web de btc-rpc-explorer.....61
Figura 52: Servicio de Logstash en docker-compose.....62
Figura 53: Servicios de ELK en Docker Desktop.....63
Figura 54: Pipeline de logstash.....63
Figura 55: Logs de ejecución de pipeline de Logstash.....64
Figura 56: Kibana. Índices tras cargar fichero de logs.....64
Figura 57: Logstash. Plugins instalados.....65
Figura 58: Fichero pipeline de logstash para cargar datos de Bitcoin.....66
Figura 59: Log de Blockchain. Inicia pipeline de blockchain.....67
Figura 60: Log de Logstash. Se conecta a elasticsearch.....67
Figura 61: Kibana. Listado de índices.....68
Figura 62: Kibana. Data view.....68
Figura 63: Kibana: apartado Discover.....69
Figura 64: Fichero de configuración de pipeline incluyendo filtro.....70
Figura 65: Comparativa de documentos tras insertar filtro.....71
Figura 66: Datos de índices con usernames.....72
Figura 67: Discover de Bitcoin_address_v2.....72
Figura 68: Kibana: Visualize Library.....73
Figura 69: Kibana: Visualización de las 10 cuentas más comunes.....73
Figura 70: Kibana: Visualización Lens Número de apariciones.....74
Figura 71: Kibana: Lista de direcciones de bitcoin_address.....74
Figura 72: Kibana: Dashboard con visualizaciones de varios índices.....75
Figura 73: Dashboard *con datos filtrados*.....76
Figura 74: Dashboard: Crear drilldown.....77
Figura 75: Dashboard: vista de drilldown activado.....77
Figura 76: Dashboard: Usar el drilldown.....78
Figura 77: Dashboard enlazado con drilldown.....78
Figura 78: Bicoin.conf.....91
Figura 79: Ejecutables de BitcoinCore.....92
Figura 80: Ejecucion de BitcoinCore.....92
Figura 81: Plugins de Logstash.....94
Figura 82: Fichero de logstash para cargar datos de Bitcoin.....94

Índice de tablas

Tabla 1: Sumario de productos.....16
Tabla 2: Descripción de capítulos.....16
Tabla 3: Descomposición del proyecto en fases.....18
Tabla 4: Unidades de medida.....19
Tabla 5: Cálculo de coste/hora.....20
Tabla 6: Coste en RRHH.....20
Tabla 7: Coste total de gastos indirectos.....21
Tabla 8: Coste total del proyecto.....21
Tabla 9: Algoritmo de exposición de riesgos.....22

1 Introducción

1.1 Contexto y justificación del trabajo

El presente trabajo de fin de máster está orientado a la obtención de respuestas a raíz del análisis de datos de transacciones de Blockchain. El resultado del proyecto será el esperado cuando se haya podido generar inteligencia analizando las transacciones de Blockchain y sea posible responder preguntas complejas para ayudar a las personas a tomar decisiones.

Actualmente en el mercado es posible encontrar herramientas que proporcionen esa inteligencia que se busca en el proyecto, sin embargo, las herramientas más completas tienen costes muy elevados y las herramientas gratuitas o con costes asequibles tienen funciones limitadas.

Hoy en día estamos rodeados de sistemas que generan y obtienen datos de muchas formas distintas, desde nuestro móvil hasta los asistentes virtuales (Alexa, Google Assistant), pasando por navegadores web, tarjetas de crédito y todo un abanico de elementos. Y me gustaría resaltar el término dato, que, según la primera acepción de la RAE es “Información sobre algo concreto que permite su conocimiento exacto o sirve para deducir las consecuencias derivadas de un hecho”. Entonces, tenemos muchos datos, pero si esos datos no se tratan, agrupan, comparan... son inútiles. Esos datos deben ser analizados para obtener la información necesaria.

¿Y como obtenemos esa información? Pues podemos buscar información concreta analizando datos o podemos usar técnicas para obtener información de contexto. Por ejemplo, si hablamos de datos del móvil, podemos responder preguntas concretas como ¿Cuántas llamadas ha hecho este número este mes? O ¿Cuántos minutos ha hablado? Pero también podemos obtener otro tipo de información como ¿Cuales son las horas en las que suele llamar? Si es un teléfono de empresa, podríamos obtener cual es el horario de trabajo aproximado de esa persona.

Veamos ahora qué conocemos por Inteligencia, volviendo a la RAE, las definiciones de la primera y segunda acepción son “capacidad de entender o comprender” y “capacidad de resolver problemas”. Por lo tanto podríamos decir que cuando hablamos de inteligencia, hablamos de obtener algo más elaborado que una estadística, hablamos de obtener respuestas para resolver un problema.

Y si hablamos de sistemas que generan muchos datos, tenemos que hablar de Blockchain. Blockchain es la tecnología que se creó originalmente para albergar a las criptomonedas como Bitcoin o Ethereum, pero que actualmente ha ampliado su uso gracias a las Aplicaciones descentralizadas y los SmartContract.

En cuanto al entorno de explotación de datos, al igual que hay muchos datos en el mundo actual, hay muchas herramientas para tratarlos, en este caso nos valdremos de una herramienta existente en el mercado, Elastic Stack. Elastic Stack(ELK) es una herramienta compuesta de tres tecnologías open source: Elasticsearch, Logstash y Kibana. Es un sistema estable, muy completo y fácilmente configurable usada por

1.2 Conceptos previos

Antes de seguir hablando de Blockchain creo que es necesario que veamos una serie de conceptos previos:

- **Address**: cuando creamos una cuenta en una Blockchain, se crea una clave privada y la pública correspondiente, a partir de esta última se crea la dirección o address, que es un código en hexadecimal que identifica la cuenta:

  0xF5d7968eDfAF9A746E71928969569ba41a7dbA53

- **Wallet**: es un elemento software o hardware que permite guardar las cuentas, en él se guarda la Address y la clave privada.
- **Transacción**: mensajes que contienen información y que van firmados por el que los emite
- **Bloque**: conjunto de transacciones.
- **Hash de bloque**: cadena alfanumérica que identifica al bloque.

Y ahora que ya sabemos lo que es un bloque, podemos definir que es Blockchain, que como su nombre indica, es una cadena de bloques.

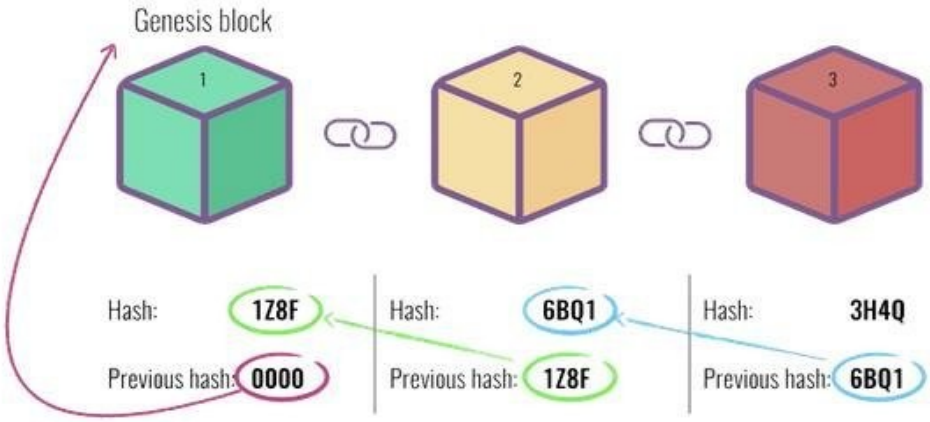


Figura 1: Blockchain: cadena de bloques.

Cada bloque de la cadena está formado por:

- El hash del bloque anterior
- El paquete de transacciones
- El hash del bloque actual.

El hecho de que cada bloque incluya el hash del bloque anterior es lo que lo convierte en una cadena.

El hash del bloque debe cumplir una serie de condiciones que son complicadas de conseguir, encontrar el hash que cumpla esas condiciones es lo que conocemos como Minar un bloque.

1.3 Estado del arte

Analizando el mercado en búsqueda de herramientas existentes que nos ayuden a obtener inteligencia, se ven dos vertientes:

- Herramientas comerciales sobre las que no hay precios públicos, en todos los casos es necesario solicitar información comercial, eso suele indicar que los precios no son asequibles.

  Además, en algunos casos incluyen certificaciones o programas de entrenamiento, por lo que parecen herramientas bastantes complejas.

  Algunos ejemplos podrían ser:

  - ChainAnalisys[8]. Más que una herramienta es un servicio personalizado en el que ayudan a generar inteligencia dependiendo del problema a resolver. Dividen estos problemas en cuatro áreas distintas: Compliance(cumplir con Leyes), Investigación (detectar e investigar Crypto crimen), DeFi(ayudar a participar en soluciones DeFi), NFTs (asegurar acceso y control seguro para NFTs)
  - Elliptic.co. Como en el caso anterior, se entiende más como un servicio que como herramientas.
  - Blockchain Intelligence Group[11]. Dispone de varias soluciones: BitRank Verified (Monitorización del riesgo de criptomonedas), QLUE (ayuda a investigar, analizar e identificar actividades maliciosas), Crypto Investigations(es un hub para conectar víctimas de estafas con investigadores)

- Por otro lado nos encontramos con herramientas con costes más moderados o gratuitas. En este caso las herramientas sólo tienen un parte gratuita, o están muy centradas en una funcionalidad o son muy sencillas
  - DuneAnalytics: Orientada a la red de Ethereum, permite investigar usando consultas simples a la BD.
  - DappRadar, herramienta muy centrada en Dapps. Es gratuita
  - DEXTools es una herramienta de análisis que permite a los comerciantes tomar sus decisiones más fácilmente al proporcionar información completa sobre criptomonedas en tiempo real.
  - Osint-toolkit. Es una recopilación de herramientas para análisis de Blockchain, la mayoría son herramientas sencillas que permiten obtener información, pero no proporcionan la inteligencia que se busca en este proyecto.

1.4 Herramientas y tecnologías

En la introducción se decía que el objetivo principal del trabajo era generar inteligencia con una herramienta gratuita.

Se propone el uso de ELK para el tratamiento de esa información. ELK es un conjunto de herramientas destinado al almacenamiento, indexación y tratamiento de datos. En una herramienta que tiene una parte gratuita[1] muy completa y que será suficiente para el propósito de este proyecto.

Elastic se compone de varios elementos:

- Beats: agentes de datos. Envían datos de clientes a Logstash para su procesado
- Logstash: se usa básicamente para la creación de pipelines de dato. Usando Logstash, se puede limpiar el dato usando filtros antes de enviarlo a Elastic.
- Elasticsearh: es un motor de búsqueda de texto que se puede utilizar como base de datos NoSQL y como motor de análisis.
- Kibana: Es la herramienta de visualización de datos. También permite la configuración de elementos(índices, gráficos, dashboards)

1 Pueden consultarse más información sobre las licencias en su página web: https://www.elastic.co/es/subscriptions

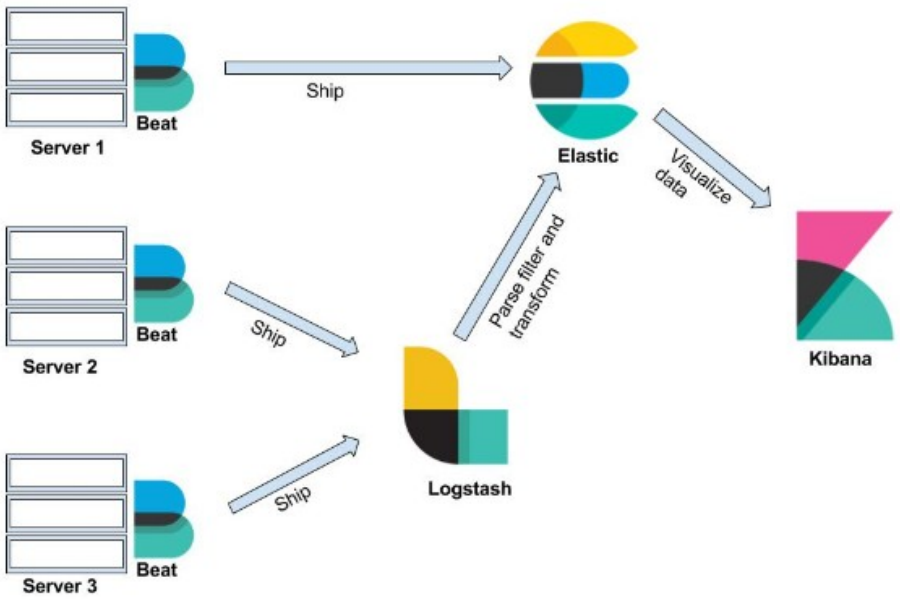


Figura 2: Arquitectura ELK.

Es una herramienta ampliamente extendida y que parte con una serie de escenarios predefinidos que se pueden seleccionar al iniciar la herramienta(APM, SIEM, observabilidad...).

Por otra parte es fácilmente escalable al poder añadir nodos para aumentar su capacidad, además, esos nodos pueden ser de distintos tipos(cluster, node, index, type, document, shard) para que el sistema se adecúe a las necesidades del proyecto.

En cuanto a la infraestructura, para la creación del proyecto se hará uso de Docker y Docker Compose. El uso de contenedores ligeros permite crear entornos de forma rápida y sencilla y que estos sean fácilmente exportables a nuevos entornos sin la necesidad de instalaciones complicadas.

1.5 Objetivos del trabajo

Tras el análisis inicial del mercado(se incluye más información en el punto *1.3 Estado del arte*) se han obtenido las siguientes conclusiones:

- En el mercado hay herramientas comerciales que proporcionan la inteligencia que se busca en este proyecto, pero no son asequibles.
- Las herramientas gratuitas o con coste asequible están muy sesgadas, suelen centrarse o en Dapps, en NFTs.
- Muchas de estas herramientas gratuitas sólo aportan estadísticas sobre los datos.

- La mayoría de estas herramientas se centran en la red de Ethereum.

Vistas las conclusiones, puesto que la mayor parte de herramientas que se han encontrado se centran en Ethereum y que para la red de Bitcoin parece haber menos posibilidades, el propósito principal del proyecto será conseguir un entorno sobre ELK que permita trabajar con datos de la red de Bitcoin.

Para obtener ese propósito se han fijado los siguientes objetivos

- OBJ-01: Crear un entorno de ELK para la carga de datos de una blockchain.
- OBJ-02: Investigar la forma de cargar datos de Bitcoin.
- OBJ-03: Cargar y tratar datos de Bitcoin.

1.6 Impacto en sostenibilidad, ético-social y de diversidad

El resultado de este TFM no tiene ningún impacto ni positivo ni negativo en aspectos de sostenibilidad medioambiental y/o huella ecológica.

El resultado de este TFM es tan técnico que no tiene ningún impacto ni positivo ni negativo en aspectos de género, diversidad o derechos humanos.

En cuánto al impacto en aspectos ético-sociales, el proyecto trata de facilitar el acceso a datos de Bitcoin. El proyecto en sí no tiene impactos en este tema, pero el uso que se puede hacer del mismo, puede influir en decisiones que se tomarán que influyan en esos ámbitos.

1.7 Enfoque y método seguro

Se aplicará un método de trabajo iterativo, con 4 iteraciones, la primera para la gestión del proyecto y las tres siguientes para la obtención de los objetivos definidos.

La primera parte consiste en ver que objetivos se van a intentar obtener y ver de que manera se organizará el trabajo para llegar a los mismos.

Para el OBJ-01, se analizarán distintas aproximaciones para crear el entorno en el que se ejecutará el sistema, será necesario ver que elementos existen para la carga de datos de las redes Blockchain

Para el OBJ-02, se investigará que tipo de elementos es posible utilizar para cargar los datos de Bitcoin en ELK.

En cuanto al OBJ-03, se implementarán los elementos definidos en el OBJ-02.

1.8 Breve sumario de productos obtenidos

Producto	Descripción
Ficheros de despliegue de infraestructura	La infraestructura se creará como IAC (Infraestructura As Code), es decir, se creará un conjunto de fichero que desplegará la infraestructura necesaria.
Ficheros de configuración	Ficheros de configuración para cargar el entorno.

Tabla 1: Sumario de productos.

1.9 Breve descripción de los otros capítulos de la memoria

Capítulo1: Introducción	Contextualización del proyecto
Capítulo2: Planificación y presupuestos	Plan de proyecto: • Planificación temporal • Planificación presupuestaria • Análisis de riesgos y medidas preventivas/paliativas
Capítulo 3: Creación entorno de trabajo	Se analizará el sistema más adecuado para desplegar el entorno de Elastic Stack.
Capítulo 4: Investigación para la carga de datos	Se indicarán los resultados de la investigación para el método elegido para la carga de datos.
Capítulo 5: Carga de datos y tratamiento de la información.	Se especificará que pasos se han llevado a cabo para realizar la carga de datos de Bitcoin en ELK
Capítulo 6: Conclusiones y trabajos futuros	Incluye una serie de conclusiones derivadas de todo el proceso de elaboración, así como una serie de posibles líneas de futuras de evoluciones
Anexos	Ficheros de configuración, scripts de ejecución.

Tabla 2: Descripción de capítulos.

2 Planificación y presupuestos

2.1 Planificación temporal

Para la planificación temporal se ha partido de la siguiente descomposición de tareas por fases:

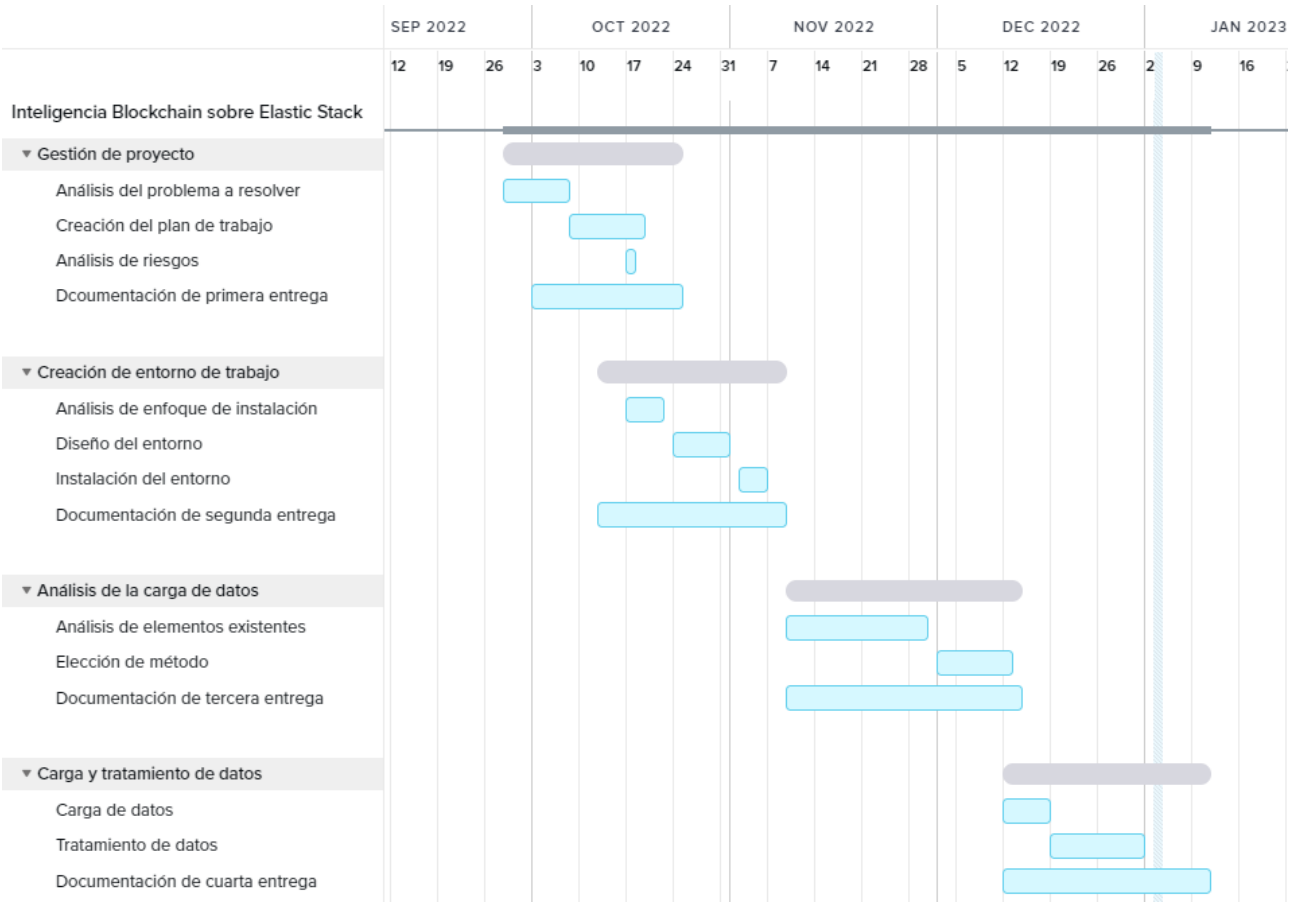


Figura 3: Planificación temporal.

Es un proyecto unipersonal, por lo que las tareas se realizarán casi siempre de forma secuencial, se produce algún solapamiento porque hay partes del proyecto que aplican a varias tareas. La documentación, para la cual se ha creado una tarea por fase, se extiende a lo largo de la misma y será paralela a las demás tareas.

En la imagen se observa que la gestión se hace toda al principio. En un proyecto real, es necesario realizar procesos de gestión durante toda la duración del proyecto, pero en este caso será suficiente con las tareas indicadas.

2.2 Planificación en horas

Una vez identificadas las tareas y estimada la duración de las mismas, se obtiene un total de 315 horas divididas de la siguiente forma:

Fase	Horas
Gestión del proyecto	70
Creación de entorno de trabajo	75
Análisis de carga de datos	120
Carga y tratamiento de datos	50
Total:	315

Tabla 3: Descomposición del proyecto en fases.

2.3 Plan de costes

2.3.1. Introducción

Para el cálculo de costes se va a emular una situación real, es decir, se calculará el coste del proyecto en un entorno real, con distintos perfiles(Jefe de proyecto, Analistas y Programadores)...

2.3.1.1 Nivel de exactitud

Se realizará un redondeo en las cifras obtenidas al centenar superior más cercano para el cálculo final del coste del proyecto. En relación a las cuentas intermedias, se utilizarán todas las cifras decimales que permite la herramienta software utilizada, en este caso el programa Calc de OpenOffice.

Se contará con una reserva de contingencia de aproximadamente el 5% del presupuesto del proyecto, en el capítulo 2.3.1.6 se calcula esta reserva.

2.3.1.2 Unidades de medida

Se utilizan las siguientes unidades de medida:

Medida	Unidad
Trabajo	horas/persona
Salario	eu/año
Coste	eu

Tabla 4: Unidades de medida.

2.3.2. Estimación de costes

2.3.1.3 Horas anuales de trabajo

El número de horas de trabajo anual es un dato que se puede encontrar en los convenios laborales. El XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública [78], es el aplicable en este caso. El último fue publicado en el BOE el 22 de febrero de 2018 y abarca los 2017, 2018 y 2019, pero como es el último, sigue en vigor. En este convenio se establece que la jornada máxima será de 1800 horas. Se tomará este dato como referencia para los cálculos.

2.3.1.4 Porcentaje que la empresa pagará a Seguridad Social

El porcentaje que paga la empresa a la Seguridad Social por trabajador es de un 29,9% del sueldo bruto del trabajador.

2.3.1.5 Sueldos medios por perfil

Según Talent.com[79] el sueldo medio para los perfiles que se necesitan en España es el siguiente:

Figura 4: Salario medio anual en España para analista programador y jefe de proyecto.

Se tomarán como datos:

- Jefe de proyecto: 35000 eu/año
- Analista programador: 31500 eu/año

Se calcula ahora el coste por hora de cada perfil, para esto, al sueldo bruto de la tabla anterior hay que sumarle un 30% que se corresponde con la parte de Seguridad Social que paga la empresa. Este sueldo se dividirá entre las horas anuales de trabajo(1800).

Perfil	sueldo bruto	con Seguridad social	coste/hora
Jefe de proyecto	35000 eu	45500 eu	25,27 eu hora
Analista programador	31500 eu	40950 eu	22,75 eu hora

Tabla 5: Cálculo de coste/hora.

2.3.1.6 Coste total

Coste en RRHH

Entonces, el proyecto constará de 315 horas que se distribuyen de la siguiente manera entre los perfiles. El jefe de proyecto se encarga de la gestión del proyecto, el analista programador se encargará de las demás tareas, se ha propuesto la tarea como una prueba de concepto, por lo que se encargará del análisis, documentación e implementación del sistema base. Se redondeará a la hora.

Perfil	porcentaje	número de horas	coste/hora	total
Jefe de proyecto	6%	19 horas	25,27 eu hora	480,13 eu
Analista programador	94%	296 horas	22,75 eu hora	6734,00 eu
Total	**100%%**	**315 horas**		**7214,13**

Tabla 6: Coste en RRHH.

Total gastos en RRHH: 7214,13

Gastos indirectos

Se van a suponer los siguientes gastos mensuales en servicios básicos para una empresa de 20 empleados:

Servicio	**Coste(eu/mes)**
Limpieza	150,00 eu
Electricidad	200,00 eu
Gas	60,00 eu
Material de oficina	150,00 eu
Agua	40,00 eu
Personal de gestión, RRHH, administración(2 personas)	3000 eu
Alquiler	5.000,00 eu
Otros gastos	500,00 eu
Total	**9100 eu**

Tabla 7: Coste total de gastos indirectos.

Aproximadamente las horas de trabajo mensuales corresponden a 1800/30=150 horas. Por lo tanto, el gasto en servicios básicos por persona y hora es de 9100/20/150=3,04. Así que si el proyecto se compone de 315 horas son 3,04*315=957,6 eu

Total gastos indirectos= 958 eu.

Reserva

Hasta ahora el coste es de 8171,73 eu, se pondrá como reserva un 5%, es decir, 408,60 eu.

Resumen

Gastos	**coste**
RRHH	7214,13 eu
Gastos indirectos	957,6 eu
Reserva	408,60 eu
Total	**8580,32 eu**

Tabla 8: Coste total del proyecto.

Redondeando a la centena más cercana, **el coste total del proyecto será de 8600 eu**

2.4 Riesgos del proyecto

Dentro de la planificación de proyectos, una das partes más importantes es la gestión de riesgos del proyecto. La finalidad de crear un plan de proyecto es conseguir que el proceso se lleve a su fin, es decir, se alcancen los objetivos propuestos, en los plazos definidos y en los costes calculados. Para ayudar a que esto se cumpla se deben detectar y controlar los posibles riesgos que afecten el proceso. Se puede definir un riesgo como cualquier situación que pueda influir en el proyecto, tanto positiva como negativamente.

Dentro del plan de proyecto, sólo se tendrán en cuenta los riesgos negativos, que son los que amenazan el éxito del proyecto, para esto:

- Se hará un catálogo de riesgos del proyecto a partir de la información del catálogo de requisitos y del alcance del proyecto.
- Una vez detectados los riesgos, utilizando el algoritmo de Exposición, que se basa en la probabilidad de ocurrencia y en el impacto en el proyecto, ver Cuadro 2.3, se les otorgar´a un nivel de exposición.
- Para los riesgos que tengan una exposición alta, se especificará una plan de prevención (acciones que prevengan que el riesgo se produzca) y un plan de contingencias (acciones que se aplicarán si el riesgo se produce).

		Probabilidad		
		Alta	Media	Baja
Impacto	Alto	Alta	Alta	Media
	Medio	Alta	Media	Baja
	Bajo	Media	Baja	Baja

Tabla 9: Algoritmo de exposición de riesgos.

A la hora de gestionar un plan de riesgos, es necesario ser cauto, si algún riesgo importante no es detectado a tiempo, puede producir que el proyecto tenga retrasos o pérdidas económicas importantes. Sin embargo, tampoco es recomendable tratar con una gran lista de riesgos, ya que podría llevarnos a que el esfuerzo dedicado a gestión de cada uno de ellos no sea acorde a las necesidades del proyecto.

2.4.1. Catálogo de riesgos

- **RIE01:** Retraso respecto a la planificación temporal.
  - *Descripción*: Debido a alguno de los demás riesgos o a riesgos no detectados, se está produciendo un retraso en la planificación.

- *Probabilidad de suceso*: media
- *Impacto*: alto
- *Exposición*: alta
- *Plan de prevención*: cada dos semanas se evaluará el retraso acumulado.
- *Plan de contingencia*: redimensionar el alcance del producto eliminando requisitos secundarios.

- **RIE02**: El uso de tecnologías muy nuevas supone un atraso en el desarrollo.
  - *Descripción*: la utilización de modelos y tecnologías nuevas puede suponer un problema, ya que es más difícil estimar correctamente la duración de las tareas.
  - *Probabilidad de suceso*: media
  - *Impacto*: alto
  - Exposición: alta
  - *Plan de prevención*: Se crear´a una tarea dentro de la planificación para que el desarrollador se forme en las nuevas tecnologías.
  - *Plan de contingencia*: Contactar con un experto en esas nuevas tecnologías que ayude a solucionar los problemas.
- **RIE03**: Dificultad a la hora de cargar datos.
  - *Descripción*: se va a trabajar con una gran cantidad de datos y puede ser difícil acotar/filtrar los datos para que sean suficientes para resolver las cuestiones.
  - *Probabilidad de suceso*: media
  - *Impacto*: alto
  - *Exposición*: alta
  - *Plan de prevención*: Probar las cargas de datos lo antes posible.
  - *Plan de contingencia*: Contactar con un experto en datos Blockchain que asesore en los filtros para carga de datos.
- **RIE04**: Dificultad o imposibilidad de crear los elementos necesarios
  - *Descripción*: se van a crear ampliaciones y plugins para ELK, es posible que esto no sea sencillo o que sea posible implementar los elementos necesario para conseguir el objetivo definido
  - *Probabilidad de suceso*: media
  - *Impacto*: alto

- *Exposición*: alta
- *Plan de prevención*: Investigar la extensión de ELK lo antes posible.
- *Plan de contingencia*: Depende de la dificultad podría ser desde prescindir de algún elemento hasta replantearse el objetivo final del proyecto.

- **RIE05**: Problemas personales o enfermad grave propia o de familiares directos.
  - *Descripción*: pueden darse problemas personales que impidan o retrasen el trabajo del proyecto, pueden ser enfermedades personales o de familiares directo.
  - *Probabilidad de suceso*: media
  - *Impacto*: alto
  - *Exposición*: alta
  - *Plan de prevención*: Tratar de adelantar el trabajo lo máximo posible y no ajustar a las fechas de entrega.
  - *Plan de contingencia*: Es un caso muy difícil de evaluar por las diferentes casuísticas y duraciones. El plan de contingencia podría ser desde pedir un aplazamiento de una entrega hasta tener que llegar a posponer la entrega del TFM en otro semestre.

3 Creación del entorno de trabajo

Desde el mismo título del proyecto queda claro que la herramienta principal que se va a utilizar en este TFM es Elastic Stack, pero no puede ser la única.

Para utilizar esta herramienta es preciso instalarla y a eso se va a dedicar este capítulo.

3.1 Herramientas y tecnologías para instalaciones

En el momento técnico en el que nos encontramos donde una de las grandes preguntas que se debe hacer al inicio de cada proyecto es si sería conveniente desplegarlo en cloud, es imprescindible que cualquier instalación que se plantee sea fácil de replicar. Para esto podríamos decantarnos por distintos sistemas:

- **Máquinas virtuales**: sistema empleado durante muchos años para la instalación de entornos sin necesidad de que se instale directamente sobre el sistema de la máquina. Existen muchas herramientas que permiten este tipo de virtualización. Como ventaja, tienes un sistema operativo completo. Como inconveniente, es necesario asociarse a una tecnología en concreto, no funciona correctamente en todos los sistemas. Es una tecnología que en la actualidad se sigue usando para entornos on premise. La migración entre entornos de este tipo de sistemas no siempre es sencillo.
- **Vagrant**[20]: Herramienta de HashiCorp que con una aplicación de línea de comandos permite definir entornos que se levantan sobre máquinas virtuales. Para ello se utiliza un fichero donde se van definiendo que tipo de máquina se necesita, que recursos(cpu, RAM) y que paquetes se instalan en la máquina. Como ventaja, en un fichero de texto defines el entorno completo y puedes levantarlos de forma bastante sencilla. Como inconveniente, sigues dependiendo de máquinas virtuales clásicas y de instalar herramientas para virtualización.
- **Terraform**[21]: Herramienta de HashiCorp para IaC(Infraestructura as Code), es una herramienta de codificación declarativa que permite utilizar un lenguaje de configuración de alto nivel llamado HCL (HashiCorp Configuration Language) para describir la infraestructura en local o cloud para ejecutar una aplicación. Como ventaja, toda la definición del entorno está en ficheros de texto y permite desplegar en distintos entornos(habría que adaptar esos ficheros a esos entornos). Como desventaja, la curva de aprendizaje de Terraform es superior a Docker.

- **Docker y Docker Compose**. Docker es un sistema de virtualización ligero. Se crean imágenes que contienen sistemas operativos ligeros que sólo incluyen lo imprescindible para ejecutar. Como ventajas, es fácil de configurar y muy portable. Como desventajas, es necesario tener o generar las imágenes para ejecutar las herramientas. Con Docker Compose se puede gestionar el conjunto de contenedores así como la interconexión entre ellos.
- **Kubernetes**[25]. Sistema para automatizar los despliegues, escalar y gestionar aplicaciones en contenedores. Es un sistema de contenerización más orientado a pequeños servicios que a gestionar grandes aplicaciones. La ventaja de Kubernetes es que permite gestionar el escalado de forma sencilla. La desventaja es la curva de aprendizaje y la necesidad de mayores recursos que otros sistemas.

Una vez revisadas los principales sistemas de instalación, se ha decidido que se va a utilizar Docker y Docker Compose. La decisión se toma en base a:

- El entorno a montar es un entorno de demo/trabajo por lo que no se necesitan grandes recursos.
- Facilidad de portar la instalación a otros entornos una vez definidos los ficheros del entorno.
- Permite montar entornos abstrayéndose del sistema operativo.
- La curva de aprendizaje es adecuada al ámbito del proyecto.

3.2 Docker y Docker Compose

Como se ha visto en el apartado anterior, se va a utilizar Docker y Docker Compose para la instalación del entorno.

Para empezar, se detallarán algunas de sus características más destacadas:

- Flexibilidad: la utilización de contenedores es cada día más popular y ya es posible encontrar imágenes de Docker para la mayoría de la herramientas, incluidas el entorno de ELK.
- Ligereza: los contenedores aprovechan y comparten el kernel del host, haciéndolos más eficientes en términos de recursos que las máquinas virtuales.
- Portabilidad: es posible crear las imágenes localmente y llevarlas a cualquier entorno.

- Sistema mínimamente acoplado: cada contenedor es autosuficiente y encapsulado, lo que permite reemplazar o actualizar uno sin interrumpir a los demás.
- Escalable: es muy fácil escalar de forma vertical, es decir, aumentar los recursos que utiliza el contenedor y de forma horizontal, si la aplicación lo permite, es posible añadir más nodos al sistema.
- Seguridad: los contenedores pueden tener restricciones y aislamientos sin configuración por parte del usuario.

Los contenedores, al contrario que las máquinas virtuales, en lugar de ejecutar un sistema operativo completo lo que hacen es compartir los recursos del propio sistema operativo "host" sobre el que se ejecutan.

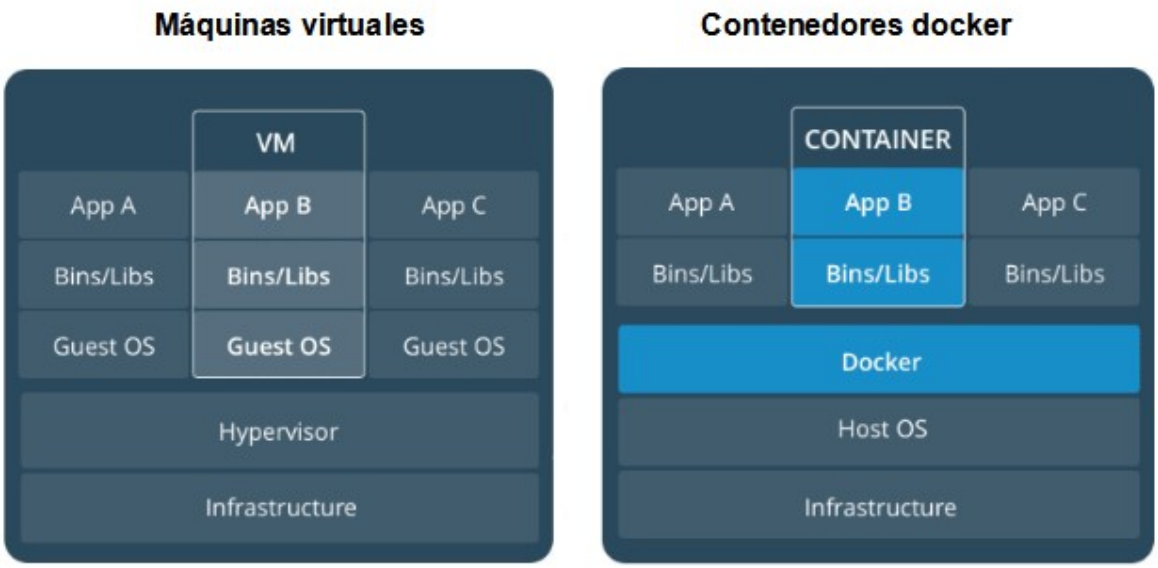


Figura 5: Máquinas virtuales vs contenedores Docker.

En la imagen se aprecia que desaparece la capa del sistema operativo huésped (Guest OS), y se sustituye el hipervisor por el motor de Docker, sin embargo hay más diferencias.

Docker se encarga de ejecutar y gestionar los contenedores, pero en lugar de exponer los diferentes recursos de hardware del host, lo que hace es compartir entre todos los contenedores ese hardware, optimizando su uso y eliminando la necesidad de tener una cantidad de sistemas operativos separados para conseguir el aislamiento y garantizar el mismo comportamiento de las aplicaciones contenerizadas en diferentes ambientes.

Otra de las grandes ventajas de Docker es su funcionamiento en base a capas. Cada imagen se compone de varias capas que se pueden reutilizar entre varias aplicaciones y/o contenedores. Cada imagen de Docker se puede asimilar como una "capa" que se puede superponer a otras para formar un sistema de archivos que combina todas las capas necesarias.

Con Docker, se puede ejecutar las imágenes pasándole los parámetros necesarios para que funcione. Por ejemplo:

```
> docker run -dp 3000:3000 getting-started
```

Con ese comando se ejecuta un contenedor basado en la imagen “getting-started” que expone el puerto 3000 del contenedor al host.

Pero aquí el problema es que cada vez que se quiera iniciar ese contenedor, es necesario ejecutar esa instrucción y esa forma de trabajar no es operativa. Es necesario automatizar esas ejecuciones y al mimo tiempo tener un sistema que permita persistir esas instalaciones, es aquí donde entra Docker Compose.

Docker Compose es una herramienta para definir y ejecutar aplicaciones Docker multicontenedor que permite simplificar el uso de Docker a partir de archivos YAML, de está forma es mas sencillo crear contenedores que se relacionen entre sí, conectarlos, habilitar puertos, volúmenes, etc.

Con un sólo comando permite crear e iniciar todos los servicios desde un fichero yaml, esto significa que puedes crear diferentes contenedores al mismo tiempo, permite iniciarlos y/o apagarlos, etc. También es posible parar y reiniciar servicios concretos.

Con Docker Compose se genera un fichero que se puede ejecutar en distintos entornos(desarrollo, preproducción, producción, demo…). De esta forma, si se actualiza un sistema sólo es necesario llevarnos el fichero de creación del entorno y ejecutarlo en los demás.

Siguiendo con el ejemplo anterior, se necesita un fichero como el siguiente:

```
1  version: '3.8'
2  services:
3      getting-started:
4          image: getting-started:latest
5          restart:unless-stopped
6          ports:
7              - "3000:3000"
8
```

Figura 6: Ejemplo de yaml para docker-compose.

Con el siguiente comando se levanta el entorno:

```
> docker-compose up -d
```

Se obtiene el mismo resultado que con el comando Docker anterior. En este caso, el ejemplo es muy sencillo, pero si hay que desplegar 4 o 5 contenedores, se haría con un sólo comando. Además, queda documentado todo lo necesario para crear el entorno.

Ya por último y para afianzar la ventaja de llevar el sistema a distintos entorno. En Docker Compose es posible tener ficheros con variables y que esas variables sean distintas para cada entorno. De esta forma hay un fichero con el despliegue, que es el mismo para todos los entornos y otro fichero por entorno que es el que incluye las variables del propio entorno.

3.3 Elastic Stack

Una vez decidido el sistema que se utilizará para la instalación, se hará una descripción detallada de la herramienta principal del TFM.

ELK Stack es un sistema con tres componentes de código abierto diferentes: Elasticsearch, Logstash y Kibana.

- Elasticsearch es un motor de búsqueda desarrollado sobre Apache Lucene.
- Logstash se usa básicamente para la canalización de datos donde se podrá obtener datos de cualquier fuente de datos como entrada, transformarlos si es necesario y enviarlos a cualquier destino como salida. En general, se usa Logstash para enviar los datos a Elasticsearch.
- Kibana es una herramienta de visualización, que se puede configurar con Elasticsearch para generar tablas, gráficos y dashboards usando los datos.

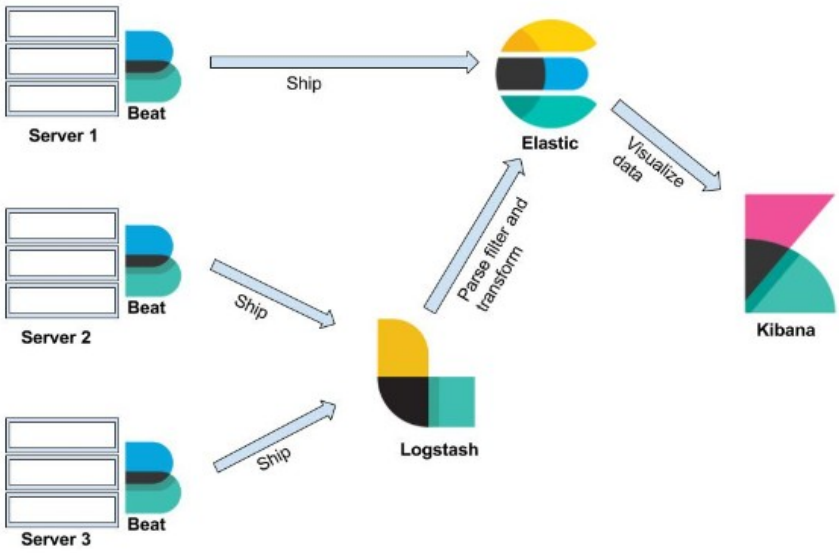


Figura 7: Arquitectura ELK.

3.3.1. Elasticsearch

Elasticsearch es un motor de búsqueda de texto completo que se puede utilizar como base de datos NoSQL y como motor de análisis. Es fácil de escalar, sin esquemas y casi en tiempo real, y proporciona una interfaz REST para diferentes operaciones. No tiene esquemas y utiliza índices invertidos para el almacenamiento de datos.

3.3.2. Logstash

Logstash se usa para la canalización de datos, se pueden usar entradas de diferentes fuentes y salidas a diferentes fuentes de datos. Usando Logstash, se puede limpiar los datos a través de opciones de filtro y mutar los datos de entrada antes de enviarlos a la fuente de salida. Logstash tiene diferentes adaptadores para manejar diferentes aplicaciones, como MySQL o cualquier otra conexión de base de datos relacional.

Para ejecutar Logstash, será necesario tener un contenedor de Logstash y editar el archivo de configuración logstash.conf, que consta de secciones de entrada, salida y filtro. Es necesario decirle a Logstash de dónde debe obtener la entrada a través del bloque de entrada, qué debe hacer con la entrada a través del bloque de filtro y dónde debe enviar la salida a través del bloque de salida.

3.3.3. Kibana

Kibana es un software open source de dashboards de ELK Stack, y es una buena herramienta para crear diferentes visualizaciones, gráficos, mapas e histogramas, y al integrar diferentes visualizaciones juntas, se puede crear dashboards. Es parte de ELK Stack; por lo tanto, es bastante fácil leer los datos de Elasticsearch. Esto no requiere ninguna habilidad de programación.

Proporciona diferentes paneles integrados con múltiples visualizaciones cuando se usan Beats, ya que crea automáticamente múltiples visualizaciones que se pueden personalizar para crear un dashboards útiles.

3.3.4. Beats

Los Beats son cargadores de datos. Se pueden instalar como agentes en diferentes servidores para enviar datos desde diferentes fuentes a un clúster central de Logstash o Elasticsearch. Están escritos en Go; trabajan en un entorno multiplataforma; y son de diseño ligero. Antes de Beats, era muy difícil obtener datos de diferentes máquinas, ya que no había un remitente de datos de un solo propósito, y había que hacer ajustes para obtener los datos.

3.4 Crear entorno

3.4.1. Pasos previos

El entorno de trabajo se ha creado en una máquina personal con las siguientes características:

- Intel(R) core™ i5-4300U CPU @ 1.90GHz 2.50 GHz
- RAM instalada: 8.00 GB
- Sistema operativo: Windows 10 Pro

Para ejecutar Docker en la máquina ha sido necesario instalar el Docker Desktop[27]. Esta aplicación, además de instalar el servicio que permite ejecutar imágenes Docker, dispone de un entorno gráfico sencillo que permite ver que contenedores se están ejecutando, ver los logs, acceder a los contenedores. Es una herramienta con licencia, pero que se puede usar en entornos personales.

La instalación de Docker incluye el Docker Compose.

Versiones de:

- Docker: 20.10.14, build a224086
- Docker-compose: 1.29.2, build 5becea4c

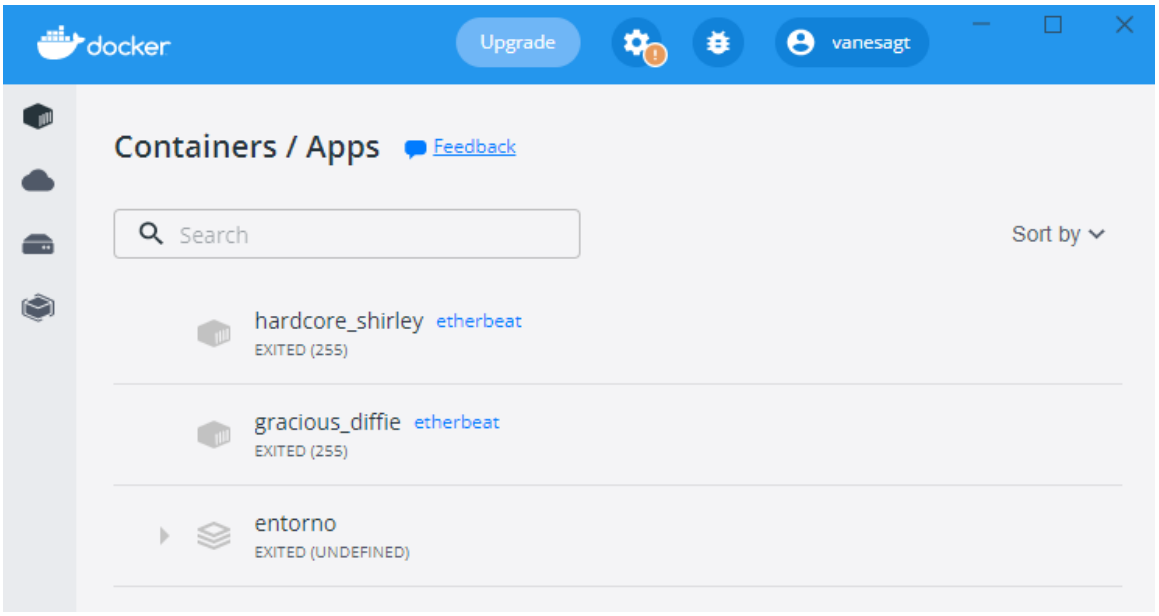


Figura 8: Docker desktop.

3.4.2. Ejecutar Elasticsearch

Para crear el entorno base se ha hecho una búsqueda de imágenes y ficheros docker-compose. Las primeras pruebas se han hecho con la imagen oficial de ElasticSearch[28]. Es un un docker-compose.yml en el que se crea 1 nodo de configuración, 3 nodos de Elasticsearch y 1 nodo de Kibana.

En la primera ejecución se ve que hay un error que se soluciona modificando un parámetro de sistema. Se incluye más información sobre este parámetro en los anexos

```
> wsl -d docker-desktop

> sysctl -w vm.max_map_count=262144
```

Se procede ahora a ejecutar el sistema:

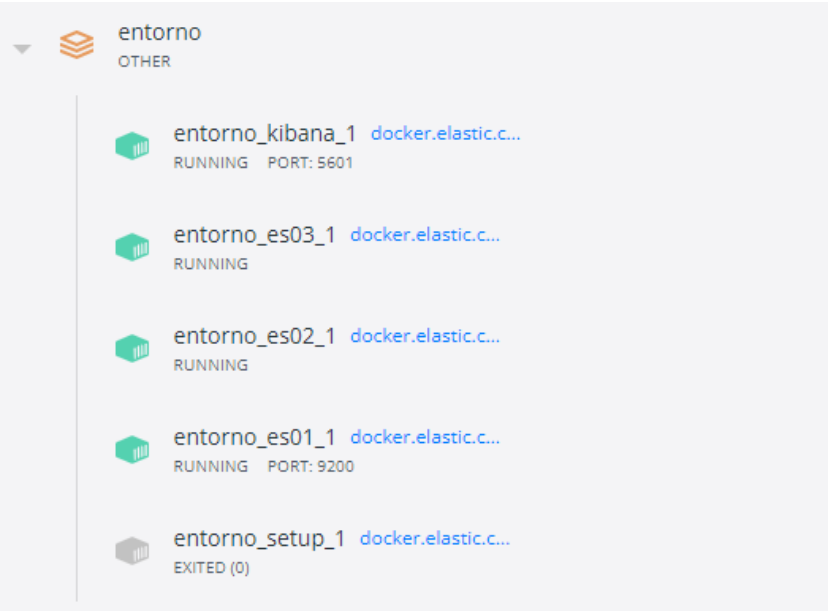


Figura 9: Primera ejecución de Elasticsearch en Docker Desktop.

Y a la primera carga de Kibana:

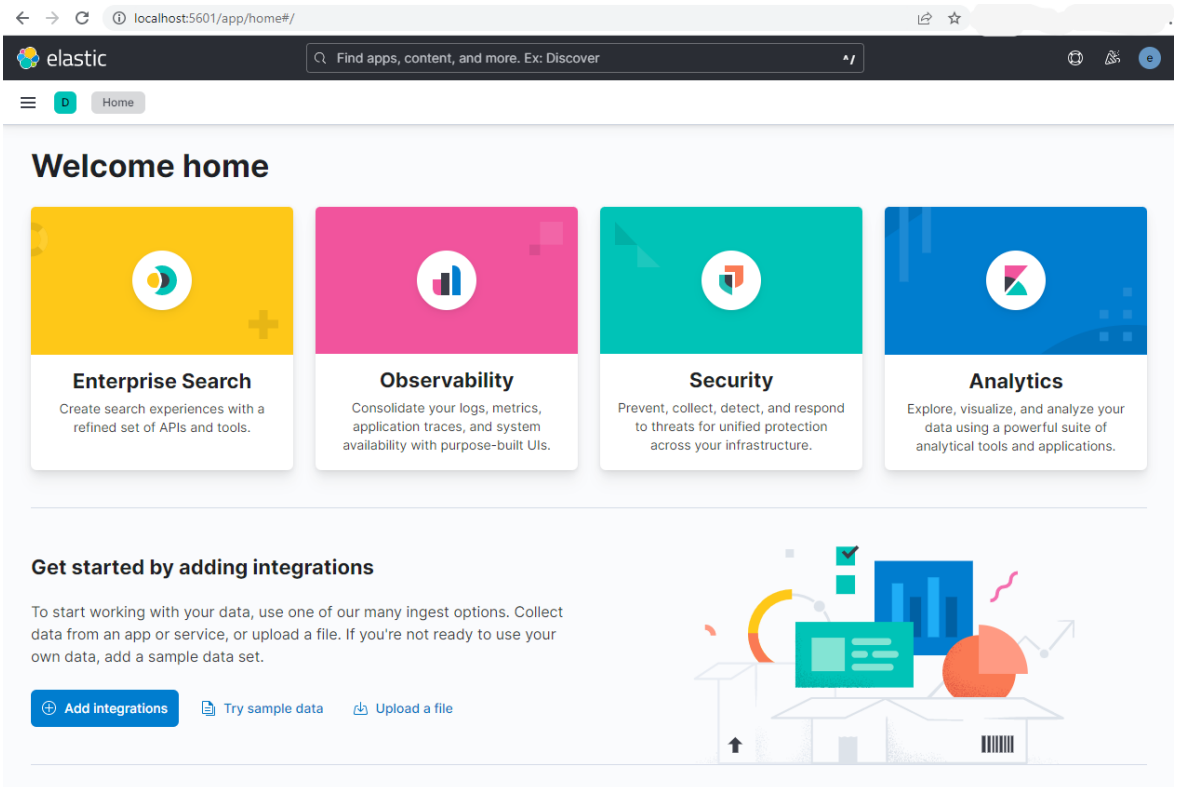


Figura 10: Pagina de inicio de Kibana.

Tras el primer arranque, la aplicación invita a instalar integraciones. Echando un vistazo general se ve que en este momento dispone de 303 elementos divididas en varias categorías.

Si se revisan las categorías se puede observar que no hay ningún elemento relacionado con ninguna BlockChain.

Aunque explicar las funcionalidades de Kibana no es uno de los objetivos del TFM, sí me que creo que es recomendable hacer un resumen general.

Navegando por la aplicación se observa que hay 5 áreas diferenciadas:

- Analytics: Área de visualización de datos. Se ven los datos brutos en el apartado de Discover o elementos de agrupación de datos en visualizaciones, mapas, dashboards.

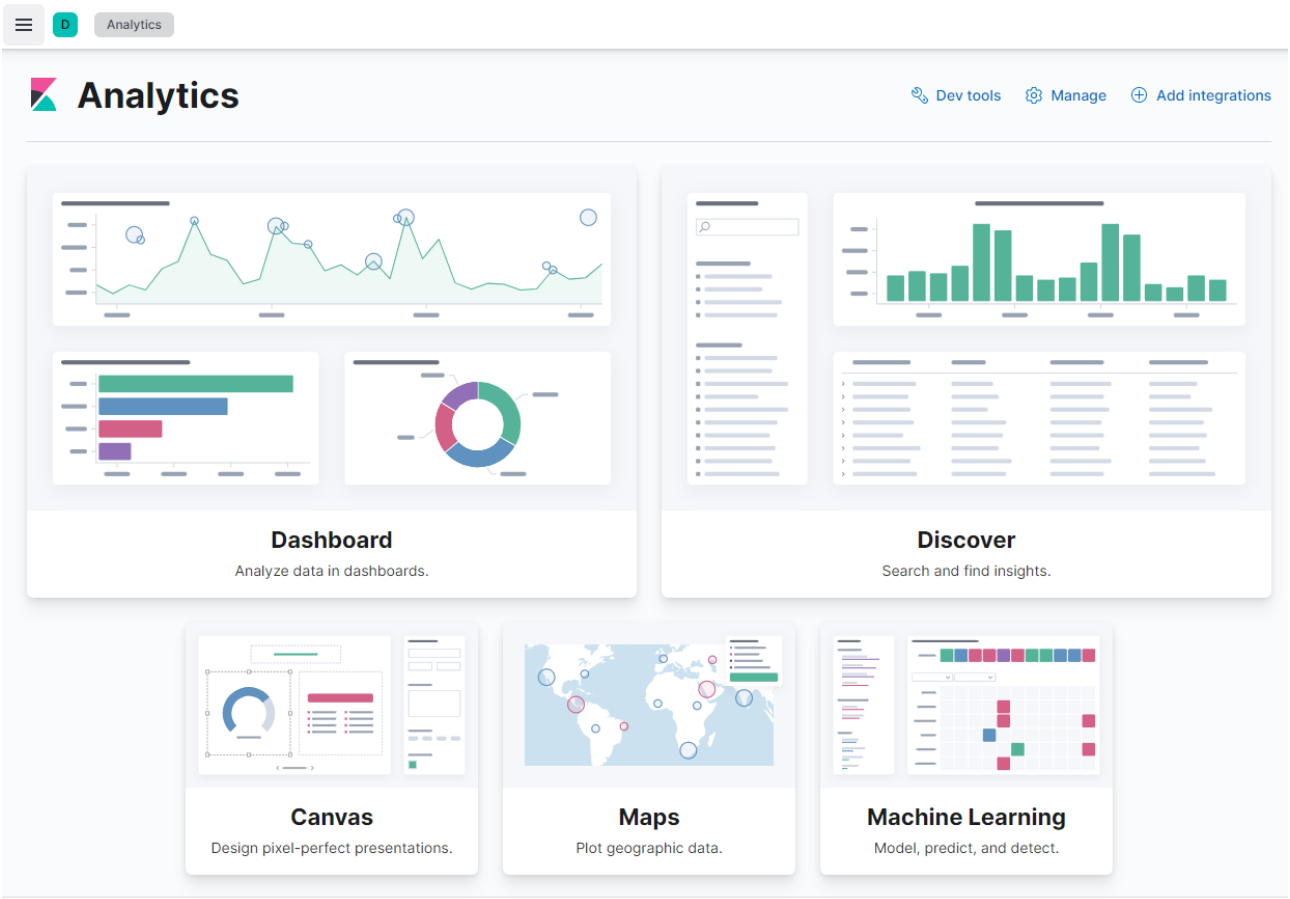


Figura 11: Kibana: Analytics.

- Enterprise Search: Conjunto de herramientas que permite de forma fácil y simple construir búsquedas que van más allá de los casos de uso típicos. Funcionalidad no soportada por el producto en licencia Basic[30].
- Observability[31]: Proporciona un único stack para unificar logs, métricas de infraestructura, trazas de aplicaciones. Permite observar una aplicación o sistema en conjunto y da la capacidad de detectar y corregir problemas.

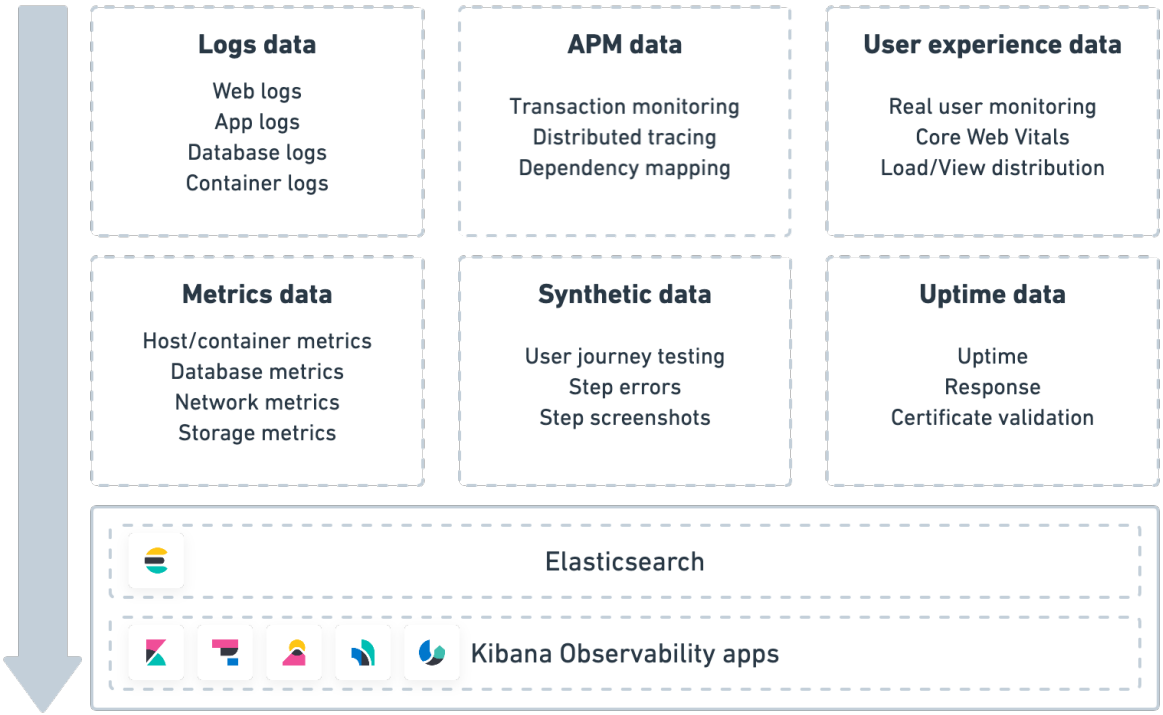


Figura 12: Kibana Observability.

- Security: Es la solución de seguridad que ofrece Elastic para prevenir, detectar y responder a posibles amenazas obteniendo una visión unificada de toda la estructura. Incluye SIEM, seguridad de Endpoint y seguridad del cloud.

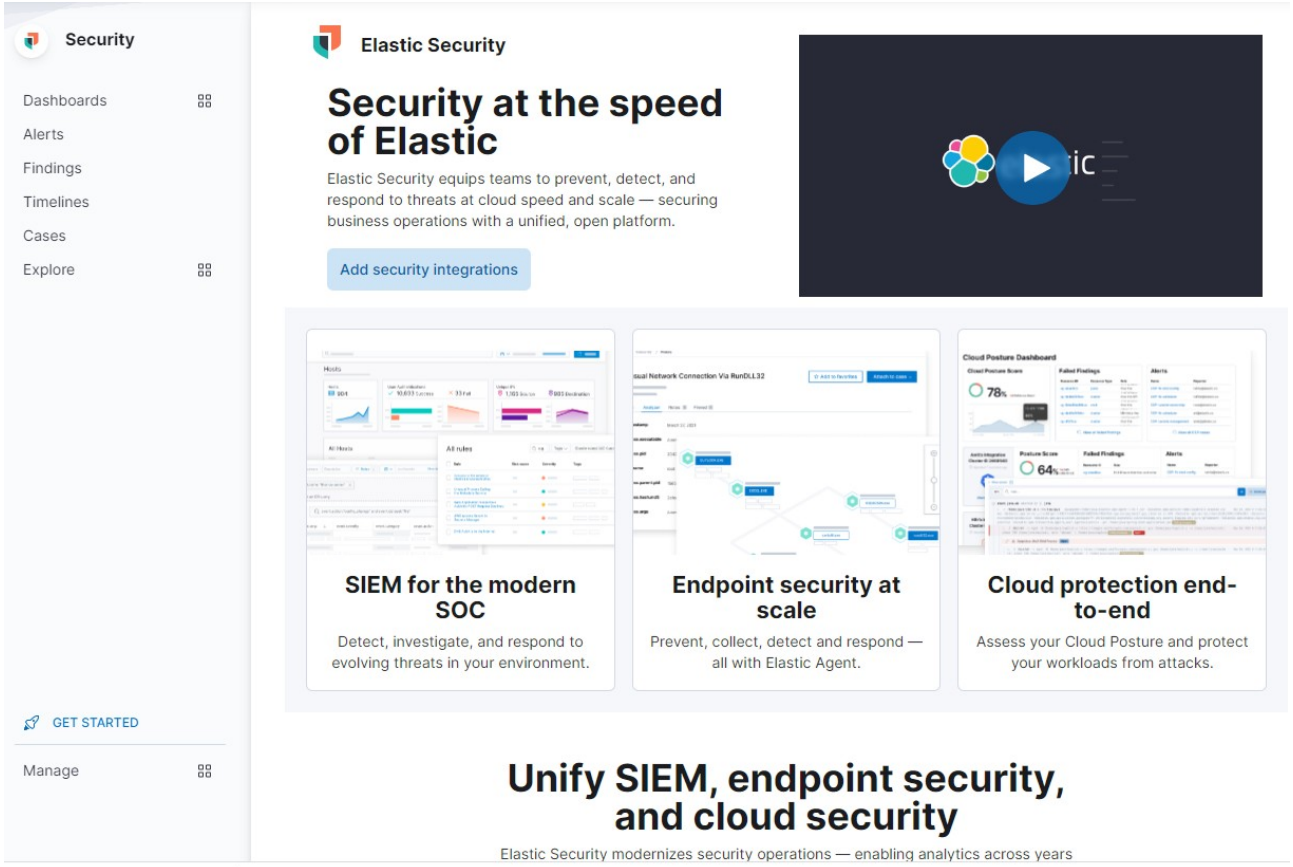


Figura 13: Kibana Security.

- Management: Área de gestión de Kibana, permite definir índices, alertas, usuarios y roles….

3.4.3. Carga de datos de pruebas

Una vez se ha instalado la herramienta, se hace una carga de datos. La primera carga se he hecho con datos de ejemplo que proporciona Kibana. Esta carda de datos crea un Index, un Data view Saved Objects y un Dashboad:

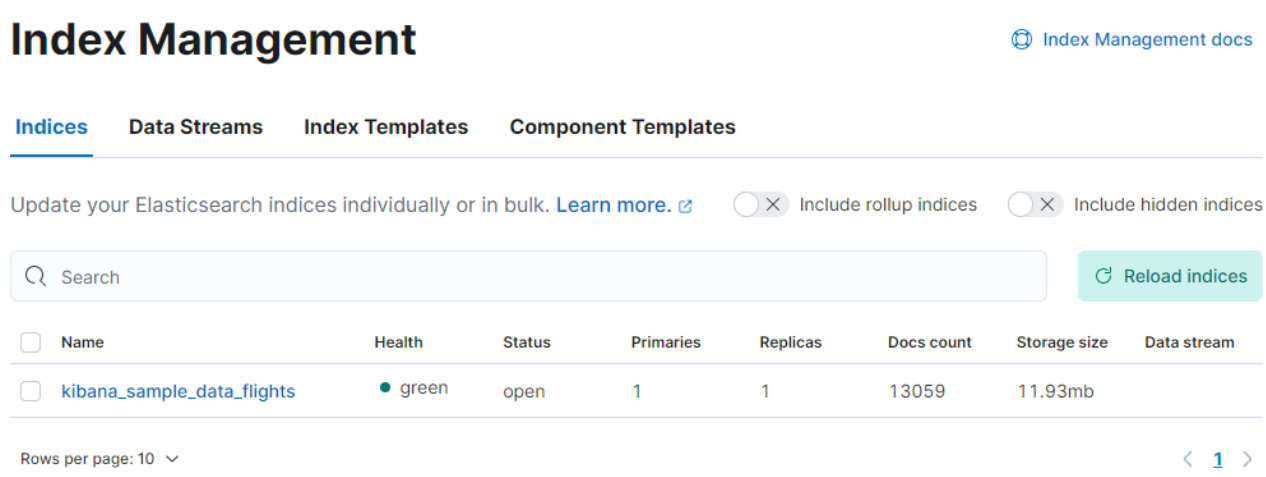


Figura 14: Kibana. Sample_data_flights. Index.

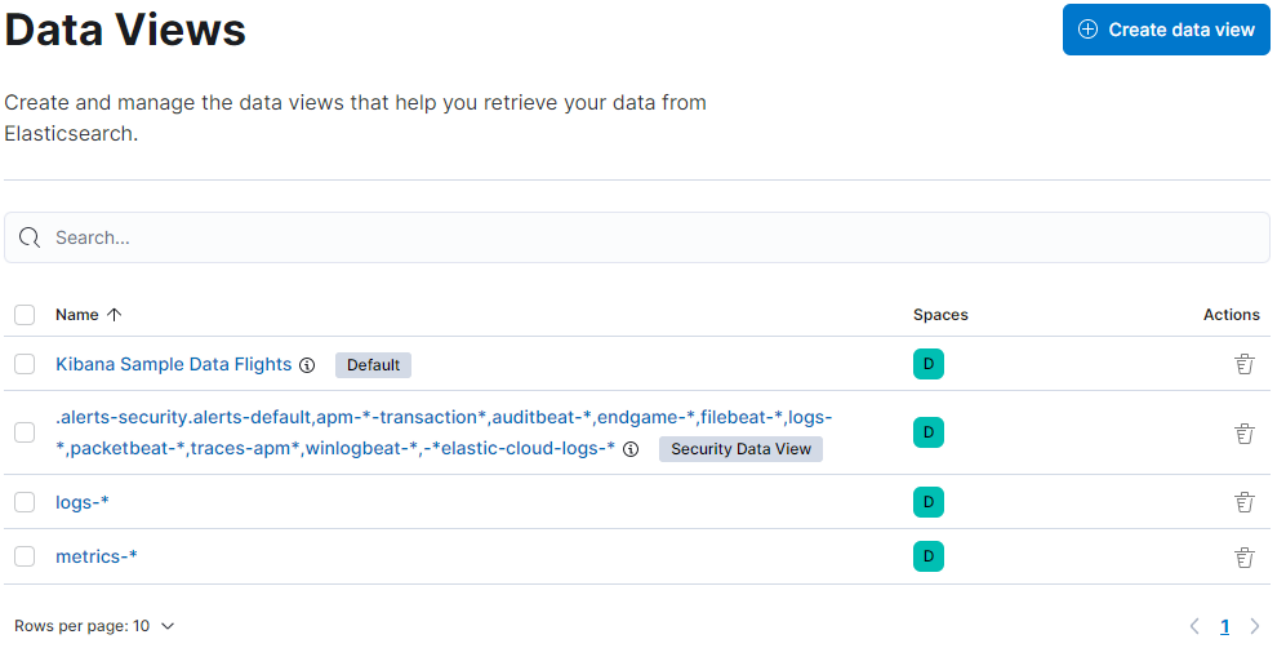


Figura 15: Kibana. Sample_data_flights. Data View.

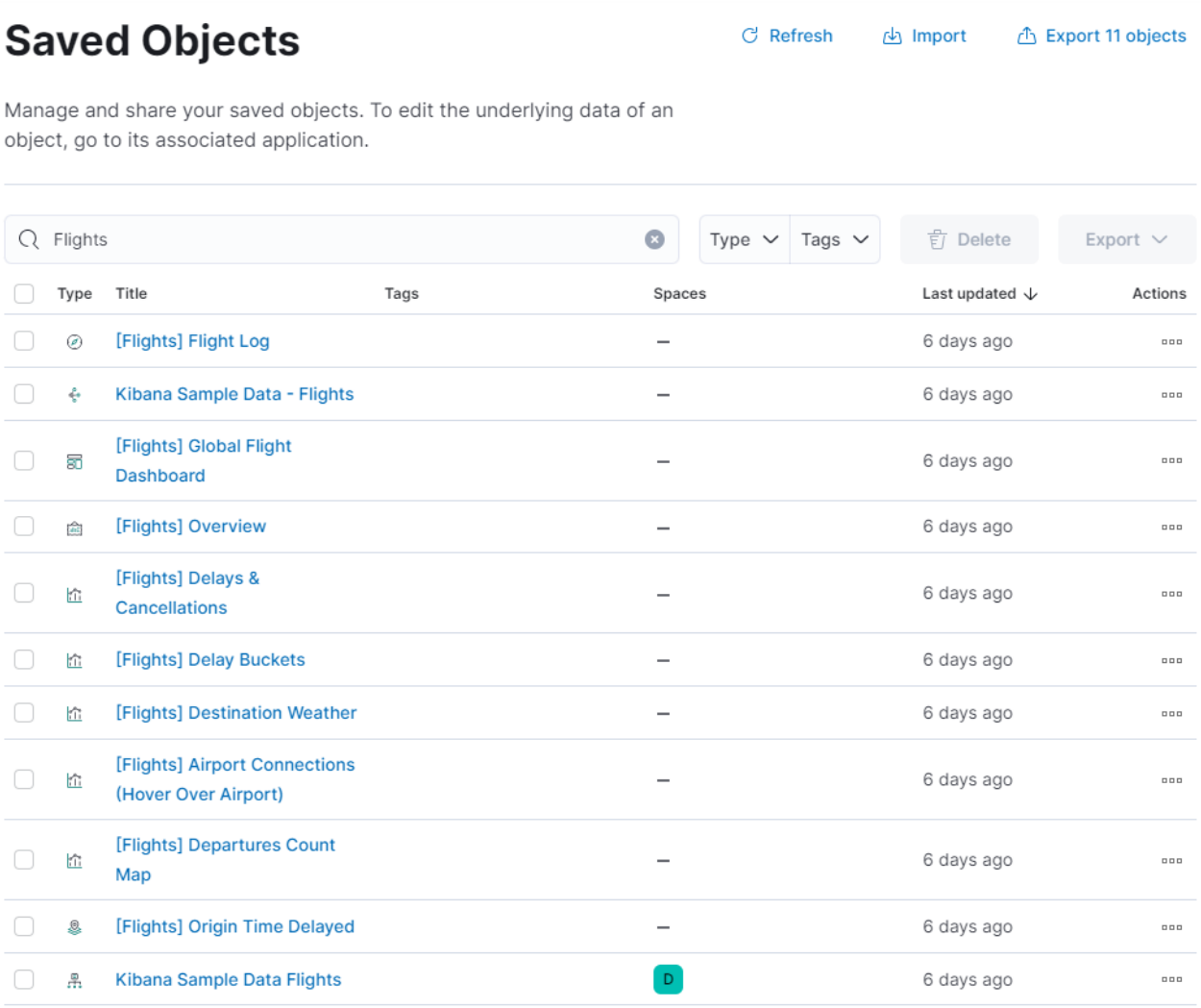


Figura 16: Kibana. Sample_data_flights. Saved Objects.

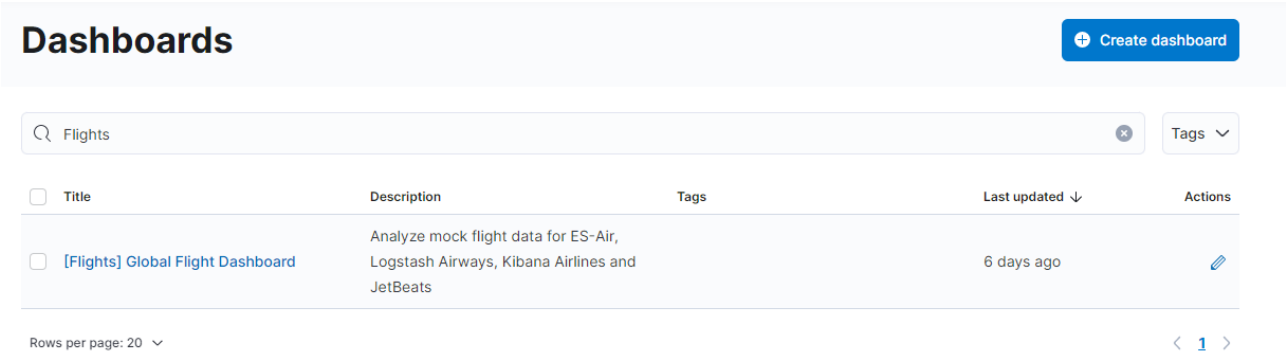


Figura 17: Kibana. Sample_data_flights. Dashboards.

Y aquí se puede ver el Dashboad que genera:

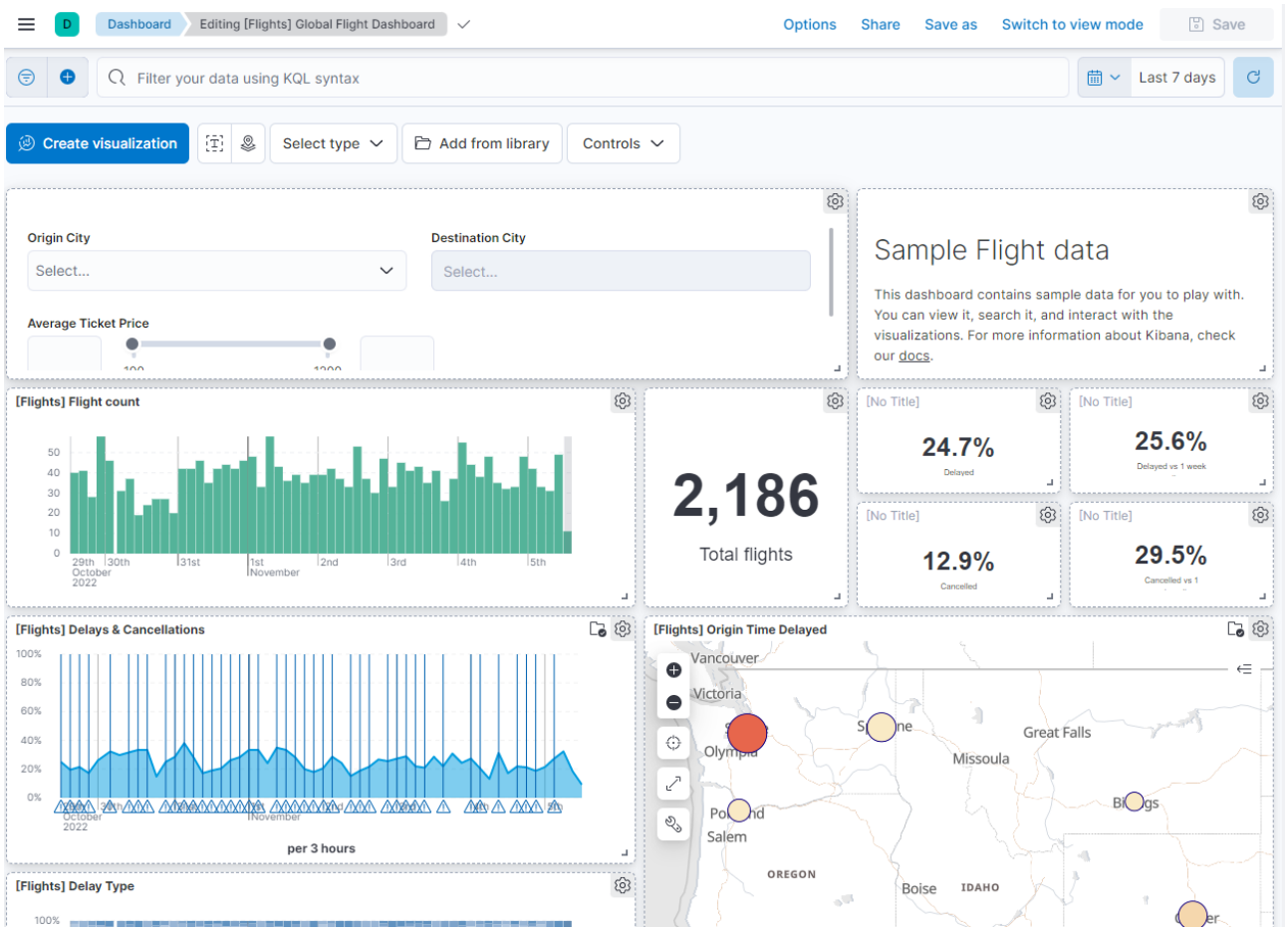


Figura 18: Kibana. Sample_data_flights. Dashboards vista.

3.4.4. Carga de datos de Ethereum

Como se ha comentado en la introducción, ahora mismo no hay ningún elemento que permita cargar datos de Bitcoin, pero si se puede encontrar en la web de Elastic un Beat para cargar datos de la red de Ethereum, así que creo que es conveniente hacer una prueba con ese Beat para ver como funciona y que datos proporciona.

En la web de Community Beats de Elastic[32] se ve que hay un Beat de Ethereum[36] llamado EtherBeat. Esta web lleva a un repositorio de Gitlab en el que está el código fuente del Beat. En las instrucciones indica la forma de ejecutarlo tanto desde sistema como desde docker. La prueba se realizará con la segunda opción, siguiendo la filosofía de que todo el sistema sea portable.

Pero antes de ejecutar este contenedor, es necesario tener los datos de acceso a la red de Ethereum que se quiere utilizar. Para las pruebas, se usa una aplicación que permite crear una red local de Ethereum. Esa aplicación es Ganache[33].

Ganache es una aplicación de código abierto desarrollada por Truffle que permite crear una red local de Ethereum para desarrollar y probar Dapps y SmartContracts en un entorno seguro y determinista. Está disponible en versión escritorio y como herramienta de línea de comando

En este caso se usará la versión para Windows y se creará una red con algunos movimientos que serán los que se cargarían en Elastic.

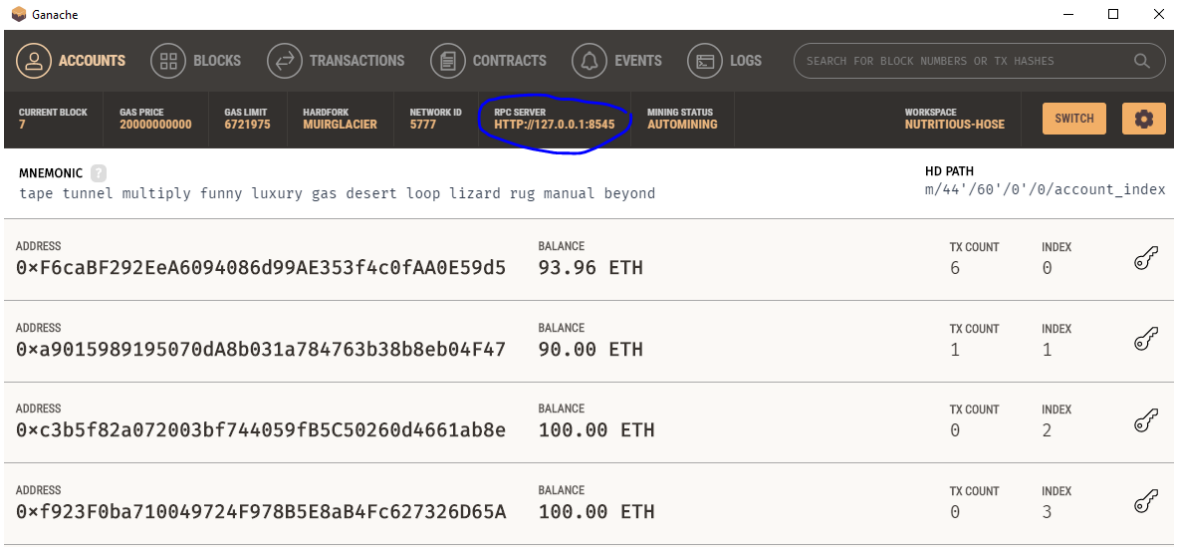


Figura 19: Ganache. Datos de Servidor.

También es posible ver que bloques se han generado.

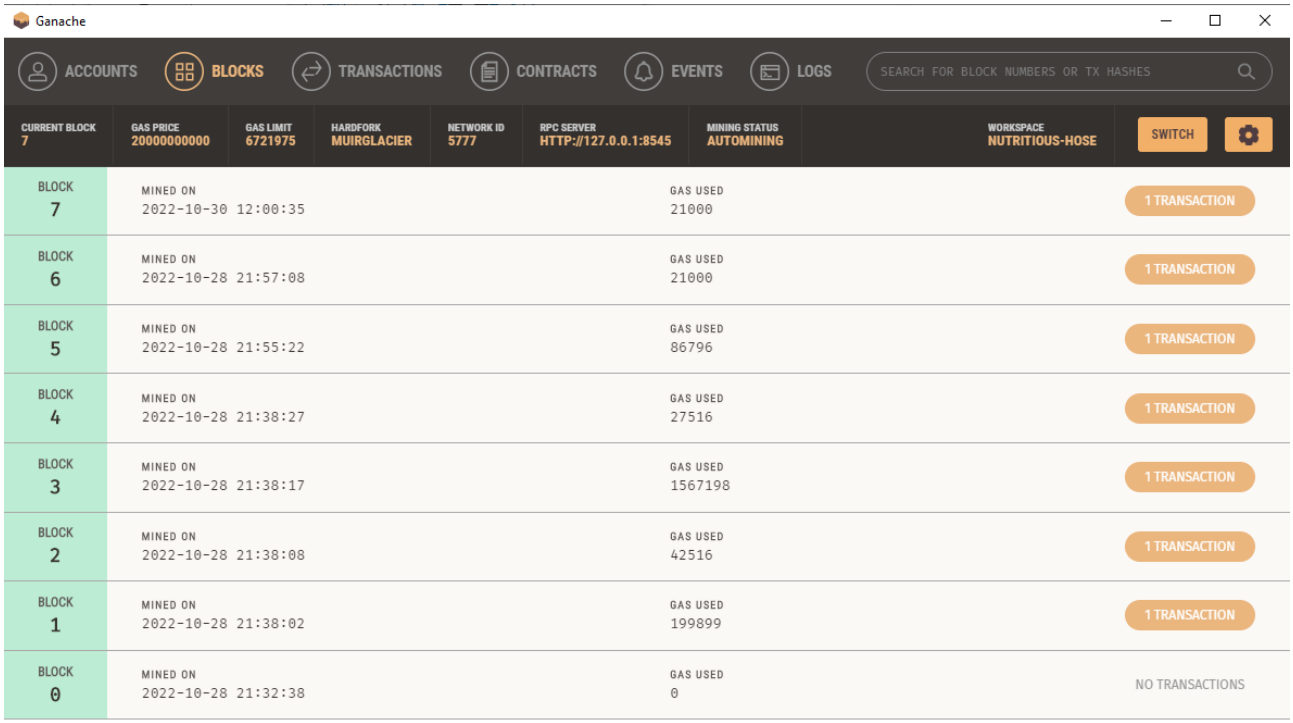


Figura 20: Ganache. Vista de Bloques.

Los datos que solicita EtherBeat son:

- Datos de acceso a Elastic: Host, usuario, contraseña
- Datos de acceso a Kibana: Host
- Datos de acceso a red Ethereum. Se solicita el RPC endpoint.

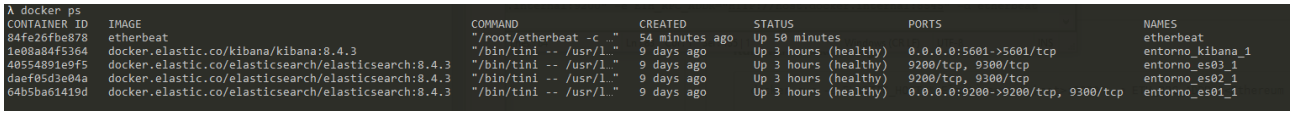
Al revisar Ganache, es posible obtener de forma sencilla el RPC endpoint, se ha resaltado en la Figura 20.

Tras descargar el EtherBeat y configurar correctamente etherbeat.yml, se procede a lanzar el contenedor:

```
> docker build -t etherbeat:latest .

> docker run --name=etherbeat -e ELASTIC_HOST="host.docker.internal:9200"
-e ETH_RPC_ADDR="HTTP://host.docker.internal:8545" -d etherbeat
```

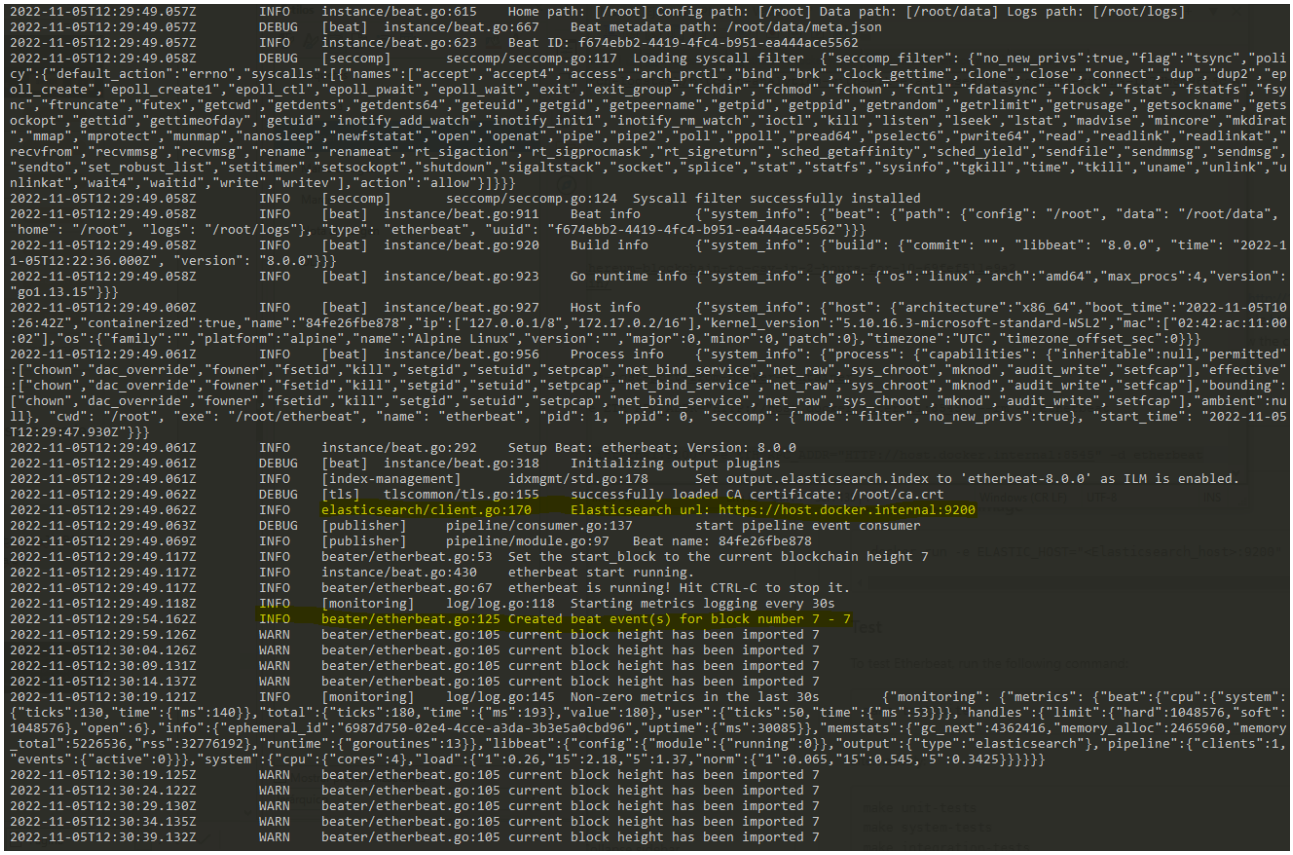
Se puede comprobar que se ha ejecutado el contenedor correctamente:



```
λ docker ps
CONTAINER ID   IMAGE                                                 COMMAND                  CREATED          STATUS                    PORTS                                       NAMES
84fe26fbe878   etherbeat                                             "/root/etherbeat -c …"   54 minutes ago   Up 50 minutes                                                         etherbeat
1e08a84f5364   docker.elastic.co/kibana/kibana:8.4.3                 "/bin/tini -- /usr/l…"   9 days ago       Up 3 hours (healthy)      0.0.0.0:5601->5601/tcp                      entorno_kibana_1
40554891e9f5   docker.elastic.co/elasticsearch/elasticsearch:8.4.3   "/bin/tini -- /usr/l…"   9 days ago       Up 3 hours (healthy)      9200/tcp, 9300/tcp                          entorno_es03_1
daef05d3e04a   docker.elastic.co/elasticsearch/elasticsearch:8.4.3   "/bin/tini -- /usr/l…"   9 days ago       Up 3 hours (healthy)      9200/tcp, 9300/tcp                          entorno_es02_1
64b5ba61419d   docker.elastic.co/elasticsearch/elasticsearch:8.4.3   "/bin/tini -- /usr/l…"   9 days ago       Up 3 hours (healthy)      0.0.0.0:9200->9200/tcp, 9300/tcp            entorno_es01_1
```

Figura 21: EtherBeat: ejecución de contenedor.

Si se revisan los logs del contenedor, es posible ver que se conecta correctamente a la red de Ganache y a Elastic:



```
2022-11-05T12:29:49.057Z        INFO    instance/beat.go:615    Home path: [/root] Config path: [/root] Data path: [/root/data] Logs path: [/root/logs]
2022-11-05T12:29:49.057Z        DEBUG   [beat]  instance/beat.go:667    Beat metadata path: /root/data/meta.json
2022-11-05T12:29:49.057Z        INFO    instance/beat.go:623    Beat ID: f674ebb2-4419-4fc4-b951-ea444ace5562
2022-11-05T12:29:49.058Z        DEBUG   [seccomp]       seccomp/seccomp.go:117  Loading syscall filter  {"seccomp_filter": {"no_new_privs":true,"flag":"tsync","poli
cy":{"default_action":"errno","syscalls":[{"names":["accept","accept4","access","arch_prctl","bind","brk","clock_gettime","clone","close","connect","dup","dup2","ep
oll_create","epoll_create1","epoll_ctl","epoll_pwait","epoll_wait","exit","exit_group","fchdir","fchmod","fchown","fcntl","fdatasync","flock","fstat","fstatfs","fsy
nc","ftruncate","futex","getcwd","getdents","getdents64","geteuid","getgid","getpeername","getpid","getppid","getrandom","getrlimit","getrusage","getsockname","gets
ockopt","gettid","gettimeofday","getuid","inotify_add_watch","inotify_init1","inotify_rm_watch","ioctl","kill","listen","lseek","lstat","madvise","mincore","mkdirat
","mmap","mprotect","munmap","nanosleep","newfstatat","open","openat","pipe","pipe2","poll","ppoll","pread64","pselect6","pwrite64","read","readlink","readlinkat","
recvfrom","recvmmsg","recvmsg","rename","renameat","rt_sigaction","rt_sigprocmask","rt_sigreturn","sched_getaffinity","sched_yield","sendfile","sendmmsg","sendmsg",
"sendto","set_robust_list","setitimer","setsockopt","shutdown","sigaltstack","socket","splice","stat","statfs","sysinfo","tgkill","time","tkill","uname","unlink","u
nlinkat","wait4","waitid","write","writev"],"action":"allow"}]}}}
2022-11-05T12:29:49.058Z        INFO    [seccomp]       seccomp/seccomp.go:124  Syscall filter successfully installed
2022-11-05T12:29:49.058Z        INFO    [beat]  instance/beat.go:911    Beat info       {"system_info": {"beat": {"path": {"config": "/root", "data": "/root/data",
"home": "/root", "logs": "/root/logs"}, "type": "etherbeat", "uuid": "f674ebb2-4419-4fc4-b951-ea444ace5562"}}}
2022-11-05T12:29:49.058Z        INFO    [beat]  instance/beat.go:920    Build info      {"system_info": {"build": {"commit": "", "libbeat": "8.0.0", "time": "2022-1
1-05T12:22:36.000Z", "version": "8.0.0"}}}
2022-11-05T12:29:49.058Z        INFO    [beat]  instance/beat.go:923    Go runtime info {"system_info": {"go": {"os":"linux","arch":"amd64","max_procs":4,"version":
"go1.13.15"}}}
2022-11-05T12:29:49.060Z        INFO    [beat]  instance/beat.go:927    Host info       {"system_info": {"host": {"architecture":"x86_64","boot_time":"2022-11-05T10
:26:42Z","containerized":true,"name":"84fe26fbe878","ip":["127.0.0.1/8","172.17.0.2/16"],"kernel_version":"5.10.16.3-microsoft-standard-WSL2","mac":["02:42:ac:11:00
:02"],"os":{"family":"","platform":"alpine","name":"Alpine Linux","version":"","major":0,"minor":0,"patch":0},"timezone":"UTC","timezone_offset_sec":0}}}
2022-11-05T12:29:49.061Z        INFO    [beat]  instance/beat.go:956    Process info    {"system_info": {"process": {"capabilities": {"inheritable":null,"permitted"
:["chown","dac_override","fowner","fsetid","kill","setgid","setuid","setpcap","net_bind_service","net_raw","sys_chroot","mknod","audit_write","setfcap"],"effective"
:["chown","dac_override","fowner","fsetid","kill","setgid","setuid","setpcap","net_bind_service","net_raw","sys_chroot","mknod","audit_write","setfcap"],"bounding":
["chown","dac_override","fowner","fsetid","kill","setgid","setuid","setpcap","net_bind_service","net_raw","sys_chroot","mknod","audit_write","setfcap"],"ambient":nu
ll}, "cwd": "/root", "exe": "/root/etherbeat", "name": "etherbeat", "pid": 1, "ppid": 0, "seccomp": {"mode":"filter","no_new_privs":true}, "start_time": "2022-11-05
T12:29:47.930Z"}}}
2022-11-05T12:29:49.061Z        INFO    instance/beat.go:292    Setup Beat: etherbeat; Version: 8.0.0
2022-11-05T12:29:49.061Z        DEBUG   [beat]  instance/beat.go:318    Initializing output plugins
2022-11-05T12:29:49.061Z        INFO    [index-management]      idxmgmt/std.go:178      Set output.elasticsearch.index to 'etherbeat-8.0.0' as ILM is enabled.
2022-11-05T12:29:49.062Z        DEBUG   [tls]   tlscommon/tls.go:155    successfully loaded CA certificate: /root/ca.crt
2022-11-05T12:29:49.062Z        INFO    elasticsearch/client.go:170     Elasticsearch url: https://host.docker.internal:9200
2022-11-05T12:29:49.063Z        DEBUG   [publisher]     pipeline/consumer.go:137        start pipeline event consumer
2022-11-05T12:29:49.069Z        INFO    [publisher]     pipeline/module.go:97   Beat name: 84fe26fbe878
2022-11-05T12:29:49.117Z        INFO    beater/etherbeat.go:53  Set the start_block to the current blockchain height 7
2022-11-05T12:29:49.117Z        INFO    instance/beat.go:430    etherbeat start running.
2022-11-05T12:29:49.117Z        INFO    beater/etherbeat.go:67  etherbeat is running! Hit CTRL-C to stop it.
2022-11-05T12:29:49.118Z        INFO    [monitoring]    log/log.go:118  Starting metrics logging every 30s
2022-11-05T12:29:54.162Z        INFO    beater/etherbeat.go:125 Created beat event(s) for block number 7 - 7
2022-11-05T12:29:59.126Z        WARN    beater/etherbeat.go:105 current block height has been imported 7
2022-11-05T12:30:04.126Z        WARN    beater/etherbeat.go:105 current block height has been imported 7
2022-11-05T12:30:09.131Z        WARN    beater/etherbeat.go:105 current block height has been imported 7
2022-11-05T12:30:14.137Z        WARN    beater/etherbeat.go:105 current block height has been imported 7
2022-11-05T12:30:19.121Z        INFO    [monitoring]    log/log.go:145  Non-zero metrics in the last 30s        {"monitoring": {"metrics": {"beat":{"cpu":{"system":
{"ticks":130,"time":{"ms":140}},"total":{"ticks":180,"time":{"ms":193},"value":180},"user":{"ticks":50,"time":{"ms":53}}},"handles":{"limit":{"hard":1048576,"soft":
1048576},"open":6},"info":{"ephemeral_id":"6987d750-02e4-4cce-a3da-3b3e5a0cbd96","uptime":{"ms":30085}},"memstats":{"gc_next":4362416,"memory_alloc":2465960,"memory
_total":5226536,"rss":32776192},"runtime":{"goroutines":13}},"libbeat":{"config":{"module":{"running":0}},"output":{"type":"elasticsearch"},"pipeline":{"clients":1,
"events":{"active":0}}},"system":{"cpu":{"cores":4},"load":{"1":0.26,"15":2.18,"5":1.37,"norm":{"1":0.065,"15":0.545,"5":0.3425}}}}}}
2022-11-05T12:30:19.125Z        WARN    beater/etherbeat.go:105 current block height has been imported 7
2022-11-05T12:30:24.122Z        WARN    beater/etherbeat.go:105 current block height has been imported 7
2022-11-05T12:30:29.130Z        WARN    beater/etherbeat.go:105 current block height has been imported 7
2022-11-05T12:30:34.135Z        WARN    beater/etherbeat.go:105 current block height has been imported 7
2022-11-05T12:30:39.132Z        WARN    beater/etherbeat.go:105 current block height has been imported 7
```

Figura 22: EtherBeat: logs de ejecución.

Y ahora es necesario ir a Kibana para ver estos bloques. Sin embargo, al entrar en Kibana, no se encuentra el índice por ninguna parte. Parece que si se están recolectando correctamente, pero no se muestra ninguna información en Kibana.

Se comprueba que el plugin tiene 3 años, por lo que es posible que no sea compatible con la versión de Elastic que se está usando, que es la última, la 8.4.3.

Al seguir investigando, se descubre que hay un proyecto de docker-compose que incluye Elastic, Kibana y Etherbeat[35] [36]. En este caso se usa la versión 7.3.1 de Elastic

Siguiendo las instrucciones de la página, se lanza este proyecto. Aparentemente se crea todo correctamente. Se levanta un Elastic, un Kibana y el contenedor con EtherBeat.

Figura 23: Ejecución de docker-elk con Etherbeat.

Al revisar los los logs del contenedor de Etherbeat, de nuevo parece que se está conectando correctamente tanto a Elastic como a la red de Ethereum.

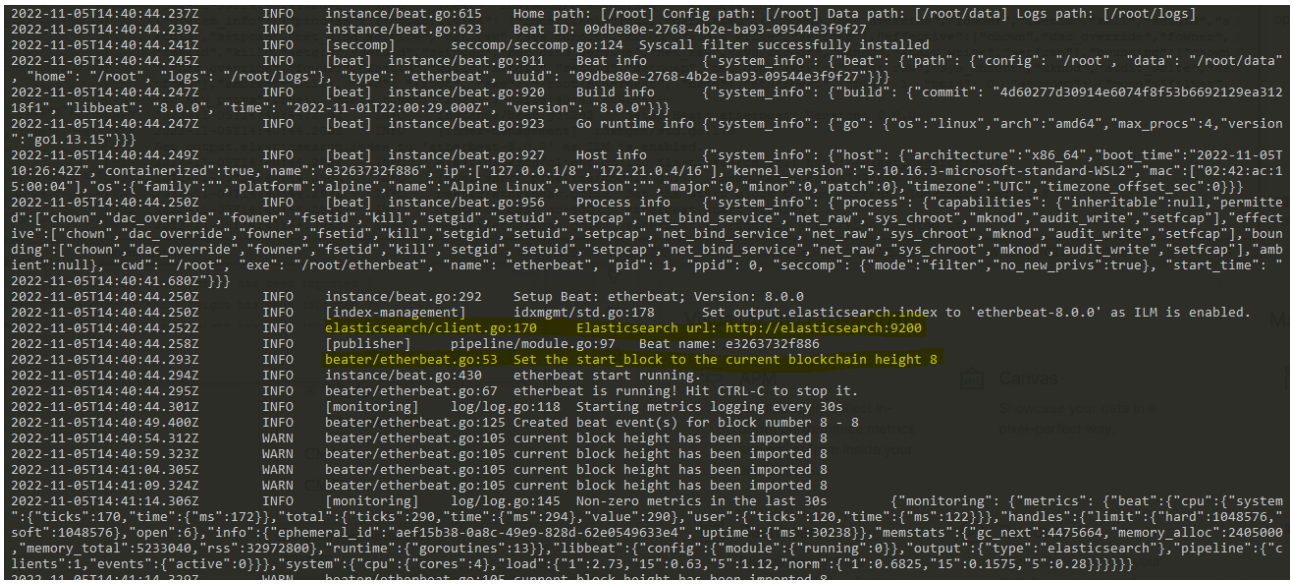


Figura 24: Docker-elk. Etherbeat logs.

Al cargar Kibana en el navegador, ya se aprecia que el aspecto es distinto, como se ha dicho, pertenece a la versión 7.3.1. De todas formas, sigue sin aparecer ningún elemento relacionado con Etherbeat.

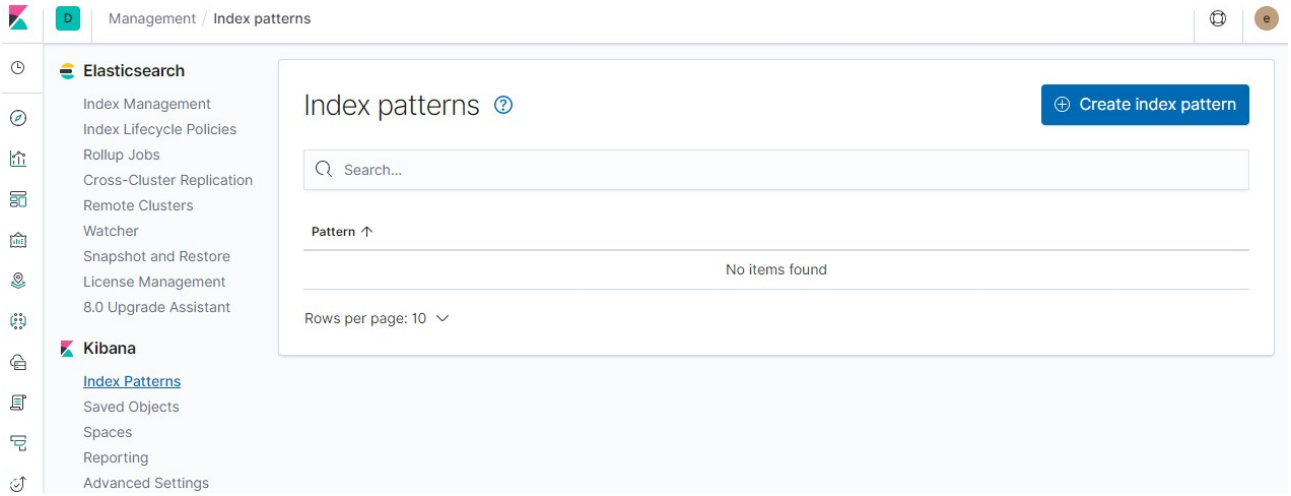


Figura 25: Docker-elk. Kibana.

Tras dos pruebas fallidas, se decide hacer un análisis del código del Beat EtherBeat. Además, se ha probado a cargar el MetricBeat para ver donde podía estar el fallo. Se obtienen las siguientes conclusiones:

- No se estaba configurando correctamente las rutas que se llaman desde el contenedor de Etherbeat. Al usar contenedores las rutas que se emplean deben ser relativos a los contenedores.
- El uso de certificados digitales en el despliegue de Elastic estaba impactando en la carga de datos. Es necesario revisar como se crean esos certificados para que funcione correctamente el Beat.
- El log de EtherBeat es muy mejorable, puesto que no indica todos los errores de conexión a Kibana o Elastic, dando la falsa impresión de que está funcionando.
- El Beat es muy sencillo, sólo recolecta y carga la información de bloques de Ethereum, sería interesante cargar también la información de transacciones.
- Algunas de las funcionalidades de EtherBeat no están correctas o son confusas.
- La documentación debería ser más completa.

Tras algunos cambios de configuración, creando un docker-compose propio, se consigue que se cree en Elastic el index de EtherBeat. Se usa la última versión de Elastic(8.5). El docker-compose.yml utilizado se incluye en el ANEXO II.

Sin embargo no se ha conseguido que EtherBeat envíe ningún documento al index.

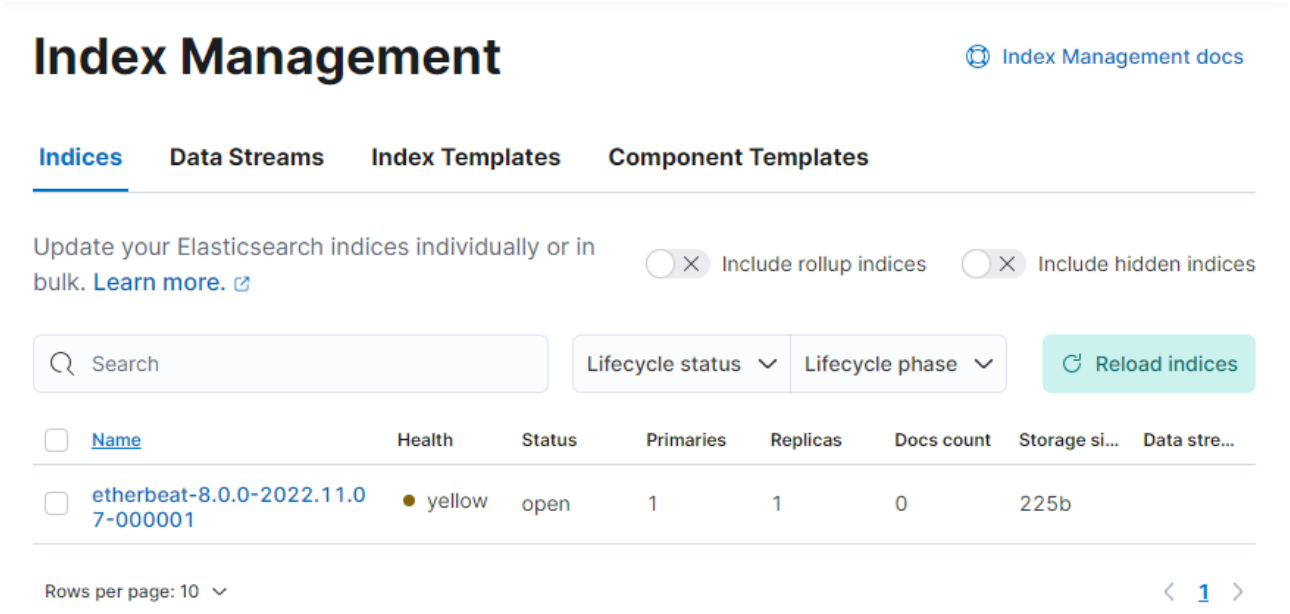


Figura 26: Kibana. Carga de datos. Index Management.

Con esto se da por finalizada la etapa de creación de entorno de trabajo. Durante las siguiente etapas se irán puliendo detalles en este entorno, pero se considera que con lo creado hasta ahora hay información suficiente para abordar la fase de análisis y diseño de elementos para la creación del entorno de Bitcoin.

4 Análisis de carga de datos

4.1 Introducción

En este apartado se analizará el proceso que se llevará a cabo para la carga de datos de Bitcoin en ELK. Se empezará analizando el Beat de Ethereum para tratar de crear uno similar para los datos de Bitcoin. También será necesario conectarse a una red de pruebas de Bitcoin o crear una red/nodo local para realizar el proceso de acceso a datos y carga en Elastic. A continuación se analizará que elementos es posible crear en ELK para el tratamiento de datos de Bitcoin.

4.2 Análisis de Beat de Ethereum

En el apartado anterior, se había llegado a ejecutar el Beat de Ethereum[36] descargado de la web de Community Beats de Elastic[32]. Se parte en este apartado de este punto y el primer paso es conseguir cargar datos en Elastic, ya que se había creado el índice pero no se habían cargado datos.

Se procede a instalar el entorno de trabajo para el análisis del Beat de Ethereum para partir del mismo y cargar datos de Bitcoin.

4.2.1. Herramientas para el desarrollo del Beat

4.2.1.1 Visual Studio

Herramienta de desarrollo de software para Windows. Una de las herramientas más completas del mercado. Incluye extensiones para adaptarse a distintos lenguajes, entre ellos el lenguaje Go que es en el que se escriben los Beat.

4.2.1.2 Lenguaje Go

Es necesario instalar el lenguaje Go[51] ya que es lenguaje en el que se escriben los Beats. Para su instalación se han seguido las instrucciones proporcionadas por el equipo de Go[52].

4.2.2 Análisis del Beat de Ethereum

Durante el análisis del Beat se encuentra la razón por la que no se están cargando datos en el índice. En la configuración del beat EtherBeat hay un parámetro en el que se indica en que bloque se debe empezar a cargar los datos:

```
################### Etherbeat Configuration Example #########################

############################# Etherbeat ######################################

etherbeat:
  # Defines how often an event is sent to the output
  period: 5s
  # Ethereum JSON RPC endpoint
  eth_rpc_addr: ${ETH_RPC_ADDR:"http://host.docker.internal:8545"}
  # Ethereum Block number to start with. If the value is larger than the latest
  # block, the beat will log an error and exit
  start_block: ${START_BLOCK:-1}
#================================ General =====================================
```

Figura 27: Configuración de EtherBeat.

Sin embargo, en el código fuente del Beat, al crear el beat se hace una consulta para obtener el número del último bloque y empieza a cargar desde ahí.

```go
// New creates an instance of etherbeat.
func New(b *beat.Beat, cfg *common.Config) (beat.Beater, error) {
    c := config.DefaultConfig
    if err := cfg.Unpack(&c); err != nil {
        return nil, fmt.Errorf("Error reading config file: %v", err)
    }
    chain, err := chain.Init(c.EthRPCAddr)
    if err != nil {
        return nil, err
    }

    currBlockNum, err := chain.GetCurrentBlock()
    if err != nil {
        return nil, err
    }
    if currBlockNum < c.StartBlock {
        return nil, ErrStartBlockTooLarge
    }
    if c.StartBlock < 0 {
        c.StartBlock = currBlockNum
        logp.Info("Set the start_block to the current blockchain height %d", currBlockNum)
    }
```

Figura 28: Código fuente de EtherBeat.

Se procede entonces a crear un Beat nuevo siguiendo las instrucciones de Elastic de la web “Build your own Beat”[49]. Tras instalar las herramientas necesarias, empiezan a aparecer problemas, las últimas versiones de las herramientas a instalar no son compatibles con el código que se indica en esa web. Se busca más información y se descubre que el proceso de creación de nuevos Beats ha sido deprecado por ELK.

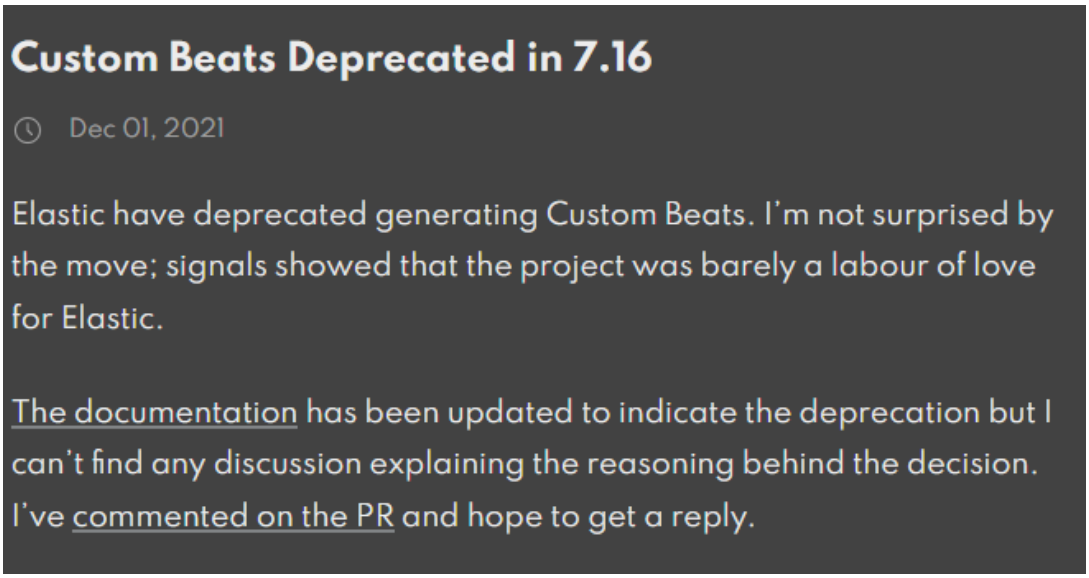


Figura 29: Custom Beats Deprecation.

En la Web de George Bridgeman[54] se indica esta información, que se confirma al consultar la web oficial de Elastic[32]:

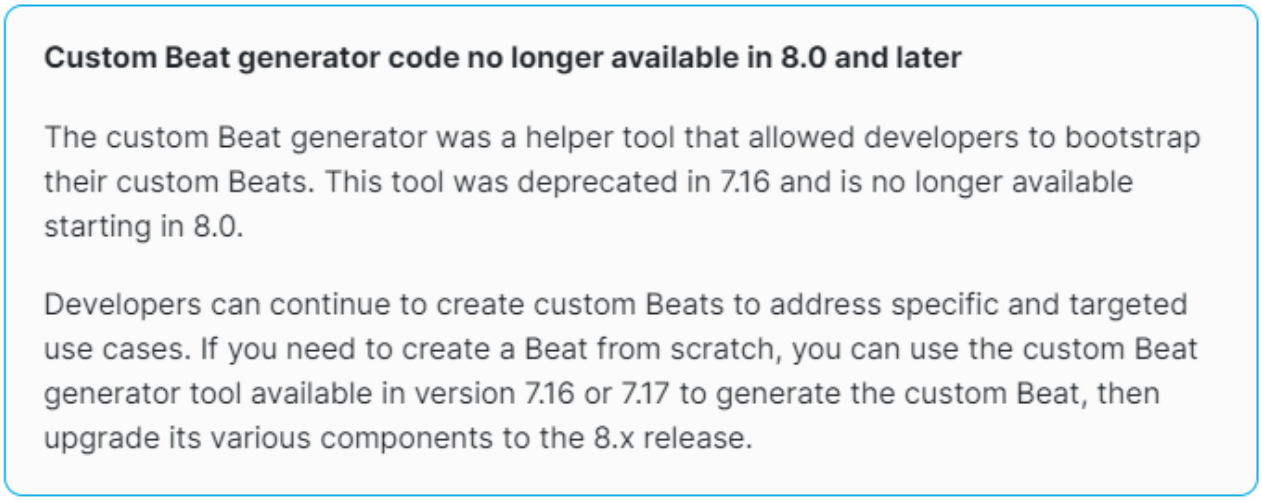


Figura 30: Custom Beat generator code no disponible

George Bridgeman tiene una web[55] donde se indica como proceder para usar la última versión disponible y crear esos Beats.

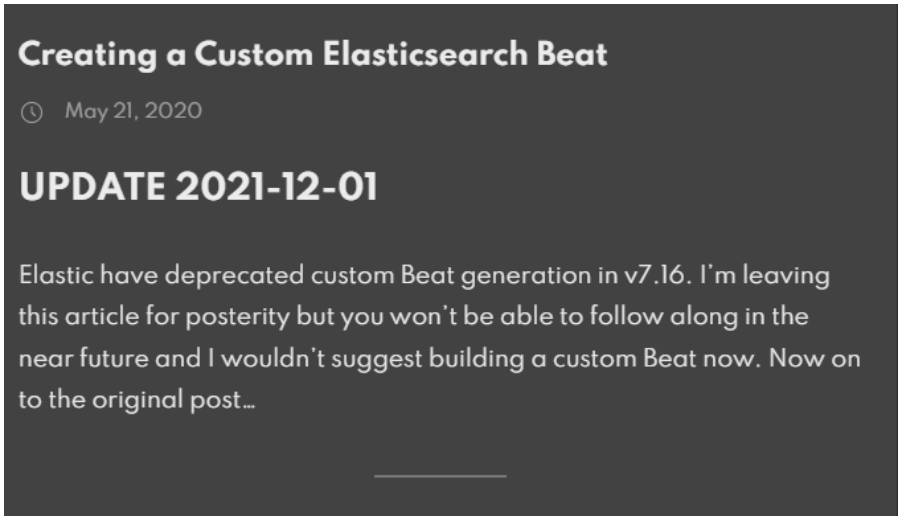


Figura 31: Crear un Beat. Actualización de deprecated.

En esta misma web también indica que la documentación para crear esos Beats no es correcta y da información de algunas de las versiones que ha utilizado.

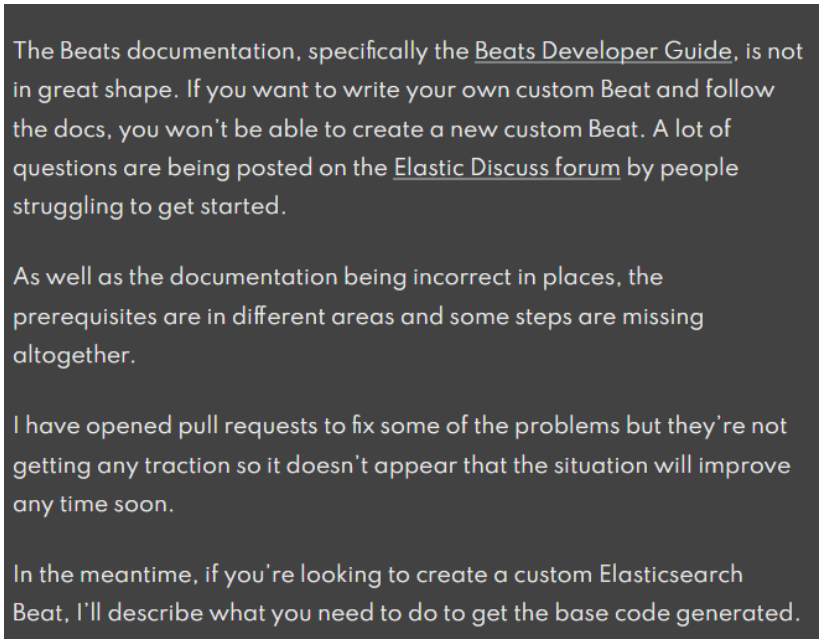


Figura 32: Crear un Beat. Problemas con la información oficial.

Al llegar a este punto, es necesario hacer una reflexión de si es conveniente seguir con el plan inicial de creación del Beat para Bitcoin.

Se exponen algunos hechos:

- En diciembre de 2021 ELK decidió dejar de mantener el Custom Beat Generator, que era la herramienta que se utilizaba para crear Beats.
- La versión de Go que debe instalarse para trabajar con la última versión es la 1.13 que fue liberada el 3 de octubre de 2019[56].
- Sería necesario hacer un análisis de que versiones de otras herramientas es necesario utilizar para que sea compatible con esa versión de Beat (gcc, Mage, Python, cookiecutter).
- La documentación para la creación y ejecución de Beats es insuficiente y confusa.
- El Beat de EtherBeat no funciona correctamente.
- El Beat de EtherBeat sólo está cargando los bloques, es necesario mejorarlo para cargar las transacciones.
- El Beat no está volcando los bloques correctamente en ELK.

Vistos los hechos, se concluye que no se creará un Beat para cargar datos en ELK. No parece muy lógico empezar a crear un elemento usando tecnologías que están discontinuadas. Se analizarán otras alternativas.

4.3 Carga de datos en ELK. Alternativas al Beat

Llegada a la conclusión de que no se creará un Beat para ELK, se analizarán otras opciones para la carga de datos.

4.3.1. Logstash

Tras descartar el Beat, Logstash parece la opción más evidente, ya que, como se ha comentado en otros apartados, Logstash es uno de los elementos de ELK para el manejo de datos.

Logstash es una herramienta que usando pipelines (que son altamente configurables) consigue

- obtener datos de un origen
- aplicar o no filtros sobre esos datos
- depositar esos datos en algún lugar.

Para aplicar estos pasos Logstash tiene una serie de plugins que se pueden aplicar. En la web oficial[58] es posible ver la lista, se observa que no hay ninguna entrada específica para Blockchain o Bitcoin.

En cuanto a la salida, en este caso se usará la salida que escribe directamente los datos en Elasticsearh[59].

4.3.2. Parsear BD de Bitcoin

Otra de las posibilidades consiste en acceder a la BD de Bitcoin y parsear los datos. Se ha encontrado un ejemplo que realiza esa acción[57]. Es una aplicación sencilla, sólo es necesario indicar el directorio donde están los fichero de Bitcoin blkXXXX.dat y el directorio donde se depositarán los resultados. Se generan ficheros de texto. En la web se indica que está realizada sólo con Python y que no requiere de librerías de terceros.

- A favor, parece sencilla de instalar y sencilla de ejecutar. Puede ser muy útil para hacer una única carga de datos para un caso concreto.
- En contra, sería necesario programar cargas recurrentes y que no hubiese solape. Es necesario otro elemento que procese los ficheros y los cargue en ELK. Además, habría que analizar el rendimiento de la misma.

4.3.3. Crear herramienta propia

Probablemente la opción más abierta y también la más costosa de implementar es usar una herramienta externa para leer y cargar datos. Desarrollar una herramienta propia permite que sea todo lo configurable que se desee. Podría permitir que se ejecutase cada cierto tiempo o que tenga algún tipo de polling para obtener los datos.

Sería posible utilizar cualquier tipo de lenguaje, pero es importante tener en cuenta los siguientes datos:

- Para leer los datos será necesario acceder a un nodo de Bitcoin. Generalmente este acceso se hace a través de RPC, por lo que sería conveniente que ese lenguaje tuviese alguna librería que facilitase esta conexión.
- Por otro lado, el objetivo final es que estos datos se carguen en ELK, por lo que podría ser interesante que ese lenguaje ya tuviese alguna librería que permitiese trabajar con ElasticSearch.

Tras una búsqueda en Internet parece que el lenguaje que más se suele emplear en este caso es Python. Es un lenguaje muy extendido y muy adaptable ya que dispone de un gran cantidad de librerías. Además:

- dispone de una librería para conectarse a Bitcoin [60] y existen muchos ejemplos de uso[61]
- dispone de una librería[62] muy completa para trabajar con Elasticsearch
- se ha encontrado un ejemplo completo de como conectarse a un nodo de Bitcoin y cargarlo en Elasticsearch[63].

4.3.4. Utilizar API de Bitcoin

Otra de las opciones podría ser el uso de la API de Blockchain para acceder a los datos. Blockchain.com dispone de varias APIs[64] para acceder a los datos y obtener la información que se necesita. Es necesario tener en cuenta el límite que recomienda Blockchain, 1 query cada 10 segundos[65]

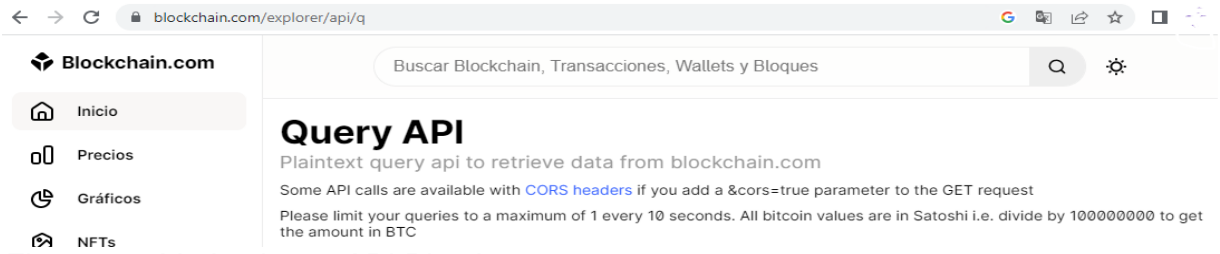


Figura 33: Limitaciones API Bitcoin.

Esta opción puede ser muy interesante porque se puede acceder a toda la información usando otro de los lenguajes más utilizados actualmente, javascript y alguno de los múltiples frameworks que lo usan.

Por lo tanto, se ha visto que hay múltiples opciones para obtener la información sin la necesidad de utilizar los Beats. Tras analizar algunas de la posibilidades, se elegirá la opción de Logstash porque parece ser la más estándar y la que más se ajusta al esquema ELK.

4.4 Crear red local de Bitcoin

Antes de proceder a obtener los datos, será necesario tener un origen de datos al que conectarse. Se ha decidido montar un nodo local de Bitcoin.

Debido a los recursos de máquina y disco duro que necesita Bitcoin, se ha optado por crear una red de Testnet de Bitcoin. Según Ycharts[66], el tamaño actual de la BD es de 442GB.

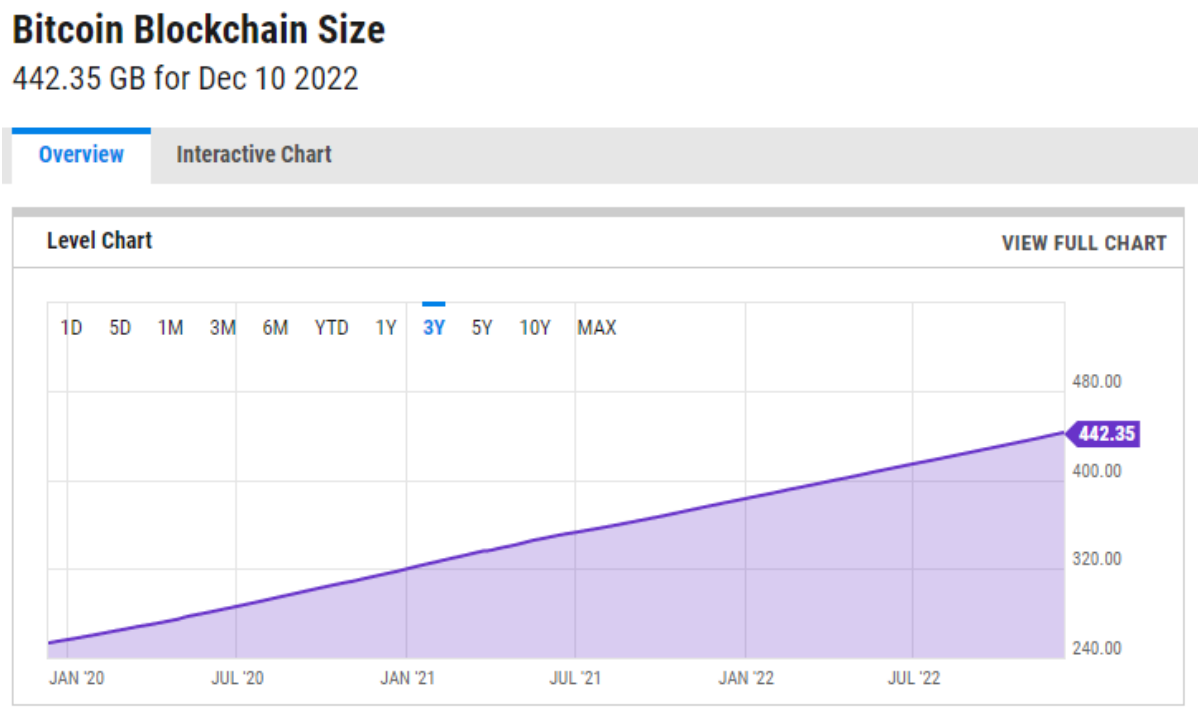


Figura 34: Tamaño de la BD de Bitcoin.

4.4.1. Testnet de Bitcoin

La red de Testnet de Bitcoin se creó en 2010 para habilitar la capacidad de que los desarrolladores probasen los cambios antes de enviarlos a la red principal. En esta red vive una criptomoneda idéntica al Bitcoin pero sin valor. Tiene su propio tipo de direcciones y su propio registro de contabilidad. El formato de las direcciones es ligeramente distinto para que no se pueda operar en la red principal por error. Además, se han tomado otro tipo de medidas para que no haya interoperabilidad entre las dos redes.

En esta red minar monedas es muy sencillo porque la dificultad es muy baja.

En cuanto al tamaño, como ya se ha indicado, es mucho menor, mientras que la de Bitcoin ocupa 442 GB, esta está en 28 GB.

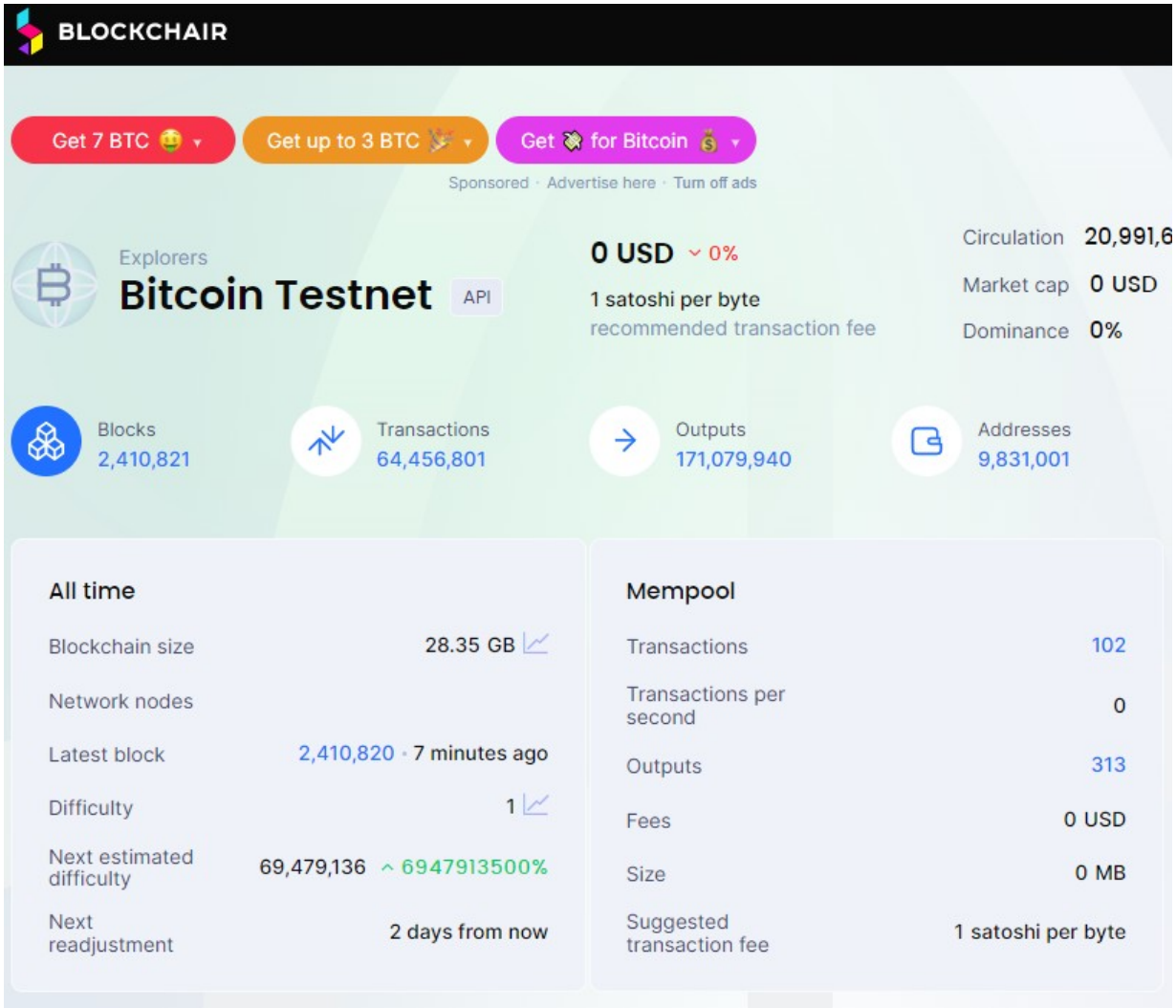


Figura 35: Datos de la red de Tesnet.

4.4.2. Crear nodo de Bitcoin

Para crear la red se utiliza el software oficial de Bitcoin, Bitcoin-core[70] que puede obtenerse de la página oficial.

Una vez descargado el código es necesario crear un fichero bitcoin.conf para indicar algunas de las propiedades que se necesita que tenga el nodo. Si no se crea este fichero, el nodo se conecta a la red Main de Bitcoin.

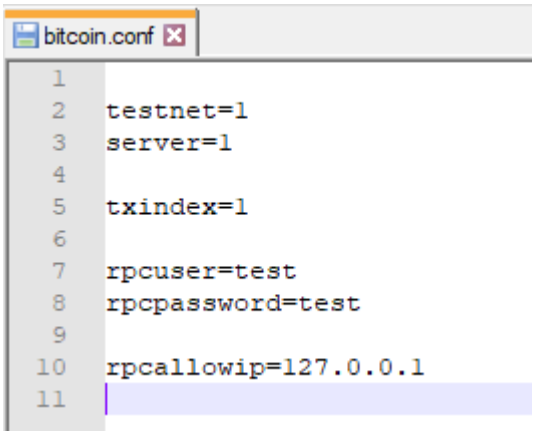


Figura 36: Bicoin.conf.

Los propiedades definidas son:

- testnet=1 → Indica que debe conectarse a la red de Testnet
- server=1 → Permite que acepte comandos RPC
- txindex=1 → Por defecto Bitcoin sólo indexa información de las transacciones relacionadas con el wallet del usuario. Si se quiere acceder a todas las transacciones es necesario poner este valor a 1.
- rpcuser=test → nombre de usuario para acceder al nodo
- password=test → contraseña de usuario para acceder al nodo
- rpcallowip=127.0.0.1 → Se indica la IP desde la que se va a permitir realizar conexiones RPC

Ahora que ya está el fichero creado, ya es posible lanzar la aplicación. El sistema de Bitcoin Core se compone de cuatro ejecutables:

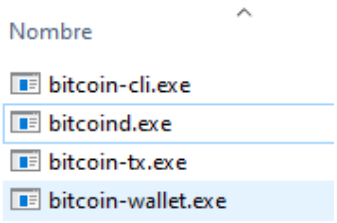


Figura 37: Ejecutables de BitcoinCore.

El ejecutable que lanza el nodo es bitcoind.exe, una vez ejecutado, se ve que empieza a sincronizar:

```
λ bitcoind.exe
2022-12-11T17:30:17Z Bitcoin Core version v22.0.0 (release build)
2022-12-11T17:30:17Z Assuming ancestors of block 0000000000004ae2f3896ca8ecd41c460a35bf6184e145d91558ce
ce1c688a76 have valid signatures.
2022-12-11T17:30:17Z Setting nMinimumChainWork=0000000000000000000000000000000000000000000005180c3bd829
0da33a1a
2022-12-11T17:30:17Z Using the 'sse4(1way),sse41(4way),avx2(8way)' SHA256 implementation
2022-12-11T17:30:17Z Using RdRand as an additional entropy source
2022-12-11T17:30:18Z Default data directory C:\Users\Vanesa\AppData\Roaming\Bitcoin
2022-12-11T17:30:18Z Using data directory C:\Users\Vanesa\AppData\Roaming\Bitcoin\testnet3
2022-12-11T17:30:18Z Config file: C:\Users\Vanesa\AppData\Roaming\Bitcoin\bitcoin.conf
2022-12-11T17:30:18Z Config file arg: rpcallowip="127.0.0.1"
2022-12-11T17:30:18Z Config file arg: rpcpassword=****
2022-12-11T17:30:18Z Config file arg: rpcuser=****
2022-12-11T17:30:18Z Config file arg: server="1"
2022-12-11T17:30:18Z Config file arg: testnet="1"
2022-12-11T17:30:18Z Config file arg: txindex="1"
2022-12-11T17:30:18Z Setting file arg: wallet = ["wallet"]
2022-12-11T17:30:18Z Using at most 125 automatic connections (2048 file descriptors available)
2022-12-11T17:30:18Z Using 16 MiB out of 32/2 requested for signature cache, able to store 524288 eleme
nts
2022-12-11T17:30:18Z Using 16 MiB out of 32/2 requested for script execution cache, able to store 52428
8 elements
2022-12-11T17:30:18Z Script verification uses 3 additional threads
2022-12-11T17:30:18Z scheduler thread start
2022-12-11T17:30:18Z WARNING: option -rpcallowip was specified without -rpcbind; this doesn't usually m
ake sense
2022-12-11T17:30:18Z HTTP: creating work queue of depth 16
2022-12-11T17:30:18Z Config options rpcuser and rpcpassword will soon be deprecated. Locally-run instan
ces may remove rpcuser to use cookie-based auth, or may be replaced with rpcauth. Please see share/rpca
uth for rpcauth auth generation.
2022-12-11T17:30:18Z HTTP: starting 4 worker threads
2022-12-11T17:30:18Z Using wallet directory C:\Users\Vanesa\AppData\Roaming\Bitcoin\testnet3\wallets
2022-12-11T17:30:18Z init message: Verifying wallet(s)…
2022-12-11T17:30:18Z Using BerkeleyDB version Berkeley DB 4.8.30: (April  9, 2010)
2022-12-11T17:30:18Z Using wallet C:\Users\Vanesa\AppData\Roaming\Bitcoin\testnet3\wallets\wallet\walle
t.dat
```

Figura 38: Ejecucion de BitcoinCore.

```
2022-12-11T17:31:00Z UpdateTip: new best=00000000000004433932fe1780cec425af6d3be0858775254f9e9fb134fd45
5f height=1564580 version=0x3fff0000 log2_work=72.149925 tx=50889425 date='2019-06-22T23:21:56Z' progre
ss=0.792868 cache=1.1MiB(8420txo)
2022-12-11T17:31:00Z UpdateTip: new best=00000000000002ec4e4d4ff09a7a6821f84f3629eaf9a4fbcd6cb3137578f1
13 height=1564581 version=0x3fff0000 log2_work=72.149928 tx=50889502 date='2019-06-22T23:34:12Z' progre
ss=0.792870 cache=1.1MiB(8489txo)
2022-12-11T17:31:00Z UpdateTip: new best=00000000000000d27b27268812d126cf7b6721803414929a4ca975381957a7
1b height=1564582 version=0x2000e000 log2_work=72.149930 tx=50889538 date='2019-06-22T23:36:28Z' progre
ss=0.792870 cache=1.1MiB(8522txo)
2022-12-11T17:31:00Z UpdateTip: new best=00000000000072be3b914ca469b0be031ad912e5674b4f083359aa6fbe6db3
5f height=1564583 version=0x2000e000 log2_work=72.149930 tx=50889722 date='2019-06-22T23:57:02Z' progre
ss=0.792873 cache=1.2MiB(8825txo)
```

Figura 39: Log de sincronización de BitcoinCore.

4.4.1.1 Interactuar con el nodo de Bitcoin

Una vez que se ha creado el nodo, ya es posible hacer consultas al nodo de Bitcoin

Se obtiene información de la red.

```
λ bitcoin-cli getblockchaininfo
{
  "chain": "test",
  "blocks": 405296,
  "headers": 2410799,
  "bestblockhash": "00000000005ce5346c1b0a7514ef9b7d063bda012576639bb68aafb6a6d1f274",
  "difficulty": 464.4398305788555,
  "mediantime": 1431865390,
  "verificationprogress": 0.05748269390781585,
  "initialblockdownload": true,
  "chainwork": "0000000000000000000000000000000000000000000040daed66e5ed3fc6e",
  "size_on_disk": 1912847617,
  "pruned": false,
  "softforks": {
    "bip34": {
      "type": "buried",
      "active": true,
      "height": 21111
    },
    "bip66": {
      "type": "buried",
      "active": true,
      "height": 330776
    },
```

Figura 40: Resultado de getblockchaininfo.

Se consulta el número de bloques que se han sincronizado. Aún no está la red completa.

```
λ bitcoin-cli.exe getblockcount
1570386
```

Figura 41: Resultado de getblockcount.

Se prueba a crear una dirección:

```
C:\Program Files\Bitcoin\daemon
λ bitcoin-cli.exe -named getnewaddress address_type=bech32
tb1qf36rswuknts8ky5jq8zlgfsp2e5xna2wv2h8vt
```

Figura 42: Resultado de crear dirección en Testnet.

Se ha creado la dirección con número *tb1qf36rswuknts8ky5jq8zlgfsp2e5xna2wv2h8vt*

Ver el balance del wallet:

```
C:\Program Files\Bitcoin\daemon
λ bitcoin-cli.exe getunconfirmedbalance
0.00000000
```

Figura 43: Resultado de getunconfirmedbalance.

Ahora se va a usar la web bitcoinfaucet[72] para conseguir saldo:

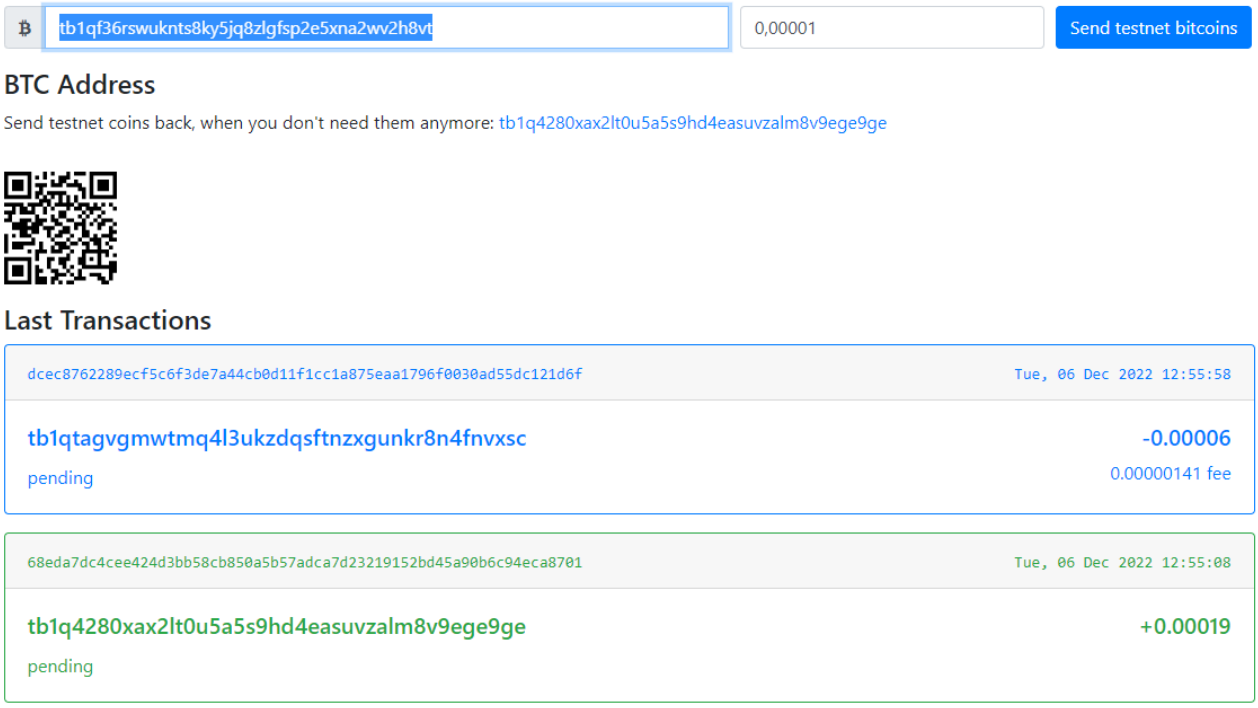


Figura 44: Obtener saldo en la address.

Se consulta el saldo de nuevo

```
C:\Program Files\Bitcoin\daemon
λ bitcoin-cli.exe getunconfirmedbalance
0.00001000
```

Figura 45: Resultado de consultar saldo tras transferencia.

Se consulta la lista de transacciones, se ve que ya figura el saldo enviado a la cuenta de prueba.

```
C:\Program Files\Bitcoin\daemon
λ bitcoin-cli.exe listtransactions
[
  {
    "address": "tb1qf36rswuknts8ky5jq8zlgfsp2e5xna2wv2h8vt",
    "category": "receive",
    "amount": 0.00001000,
    "label": "",
    "vout": 0,
    "confirmations": 0,
    "trusted": false,
    "txid": "63154cf7323065a2b770205165b5fd93e2012038bb5a53fb81bd6eb5e9bf9de6",
    "walletconflicts": [
    ],
    "time": 1670331209,
    "timereceived": 1670331209,
    "bip125-replaceable": "no"
  }
]
```

Figura 46: Resultado de obtener la lista de transacciones.

Se obtiene el balance de nuevo:

```
C:\Program Files\Bitcoin\daemon
λ bitcoin-cli.exe getbalance
0.00001000
```

Figura 47: Resultado de obtener el balance confirmado.

4.4.1.2 Instalar explorer de Bitcoin local

Para acceder a los datos de la red, se va a instalar una web en local que funciona de explorer de Bitcoin. Se utiliza el código de btc-rpc-explorer[73].

Se descarga el código y se configura el acceso al nodo local en el fichero .env. Sólo es necesario cambiar el usuario, contraseña, host y puerto.

```
# Bitcoin RPC Credentials (URI -OR- HOST/PORT/USER/PASS)
# Defaults:
#    - [host/port]: 127.0.0.1:8332
#    - [username/password]: none
#    - cookie: '~/.bitcoin/.cookie'
#    - timeout: 5000 (ms)
BTCEXP_BITCOIND_URI=bitcoin://test:test@host.docker.internal:18332?timeout=10000
#BTCEXP_BITCOIND_HOST=127.0.0.1
#BTCEXP_BITCOIND_PORT=8332
#BTCEXP_BITCOIND_USER=rpcusername
#BTCEXP_BITCOIND_PASS=rpcpassword
#BTCEXP_BITCOIND_COOKIE=/path/to/bitcoind/.cookie
#BTCEXP_BITCOIND_RPC_TIMEOUT=5000
```

Figura 48: Contenido del fichero .env de btc-rpc-explorer.

Como el nodo de Bitcoin está en la máquina local y btc-rpc-explorer va a estar en un contenedor docker, es necesario usar como host host.docker.internal. Además, al ser una red de testnet, hay que usar el puerto 18332.

Se crea la imagen docker:

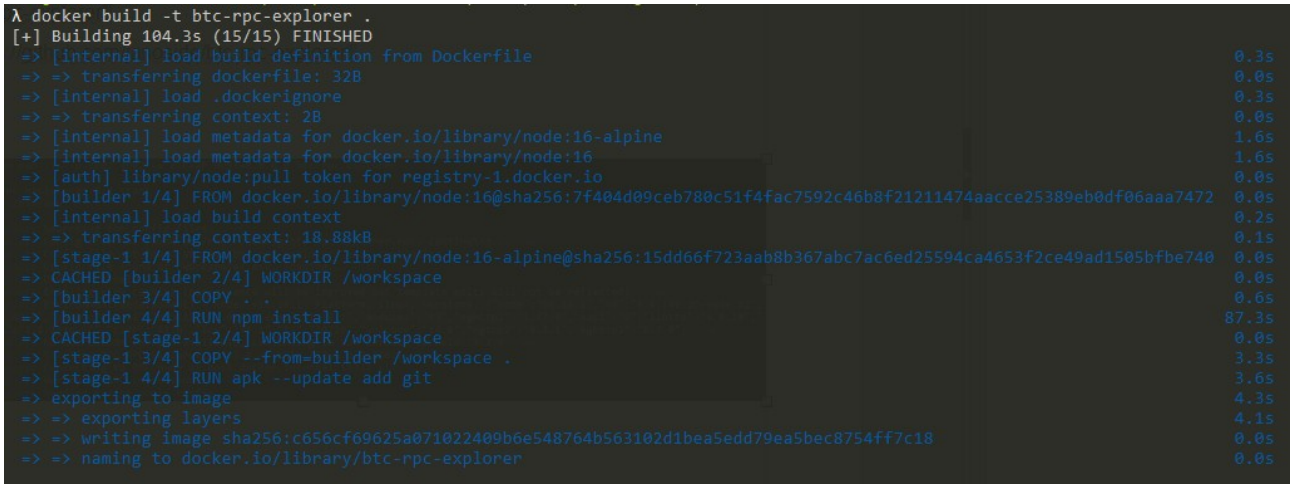


Figura 49: Creación de imagen docker de btc-rpc-explorer.

Se lanza el contenedor:

```
λ docker run -it -p 3002:3002 -e BTCEXP_HOST=0.0.0.0 btc-rpc-explorer

> btc-rpc-explorer@3.3.0 start
> node ./bin/www

  btcexp:app Searching for config files... +0ms
  btcexp:app Config file not found at /root/.config/btc-rpc-explorer.env, continuing... +3ms
  btcexp:app Config file not found at /etc/btc-rpc-explorer/.env, continuing... +1ms
  btcexp:app Config file found at /workspace/.env, loading... +1ms
  btcexp:app Default cacheId '3.3.0' +3s
  btcexp:app Enabling view caching (performance will be improved but template edits will not be reflected) +40ms
  btcexp:app Environment(development) - Node: v16.18.1, Platform: linux, Versions: {"node":"16.18.1","v8":"9.4.146.26-node.22",
"uv":"1.43.0","zlib":"1.2.11","brotli":"1.0.9","ares":"1.18.1","modules":"93","nghttp2":"1.47.0","napi":"8","llhttp":"6.0.10","
openssl":"1.1.1q+quic","cldr":"41.0","icu":"71.1","tz":"2022b","unicode":"14.0","ngtcp2":"0.8.1","nghttp3":"0.7.0"} +22ms
  btcexp:app No sourcecode version available, continuing to use default cacheId '3.3.0' +0ms
  btcexp:app Starting BTC RPC Explorer, v3.3.0 at http://0.0.0.0:3002/ +0ms
  btcexp:app Connecting to RPC node at host.docker.internal:18332 +1ms
  btcexp:app Verifying RPC connection... +21ms
  btcexp:app Loading mining pools config +3ms
```

Figura 50: Ejecución de contenedor docker de btc-rpc-explorer.

Se obtiene la siguiente web en la que es posible ver datos de la red, navegar por los bloques, las transacciones o ver datos del wallet.

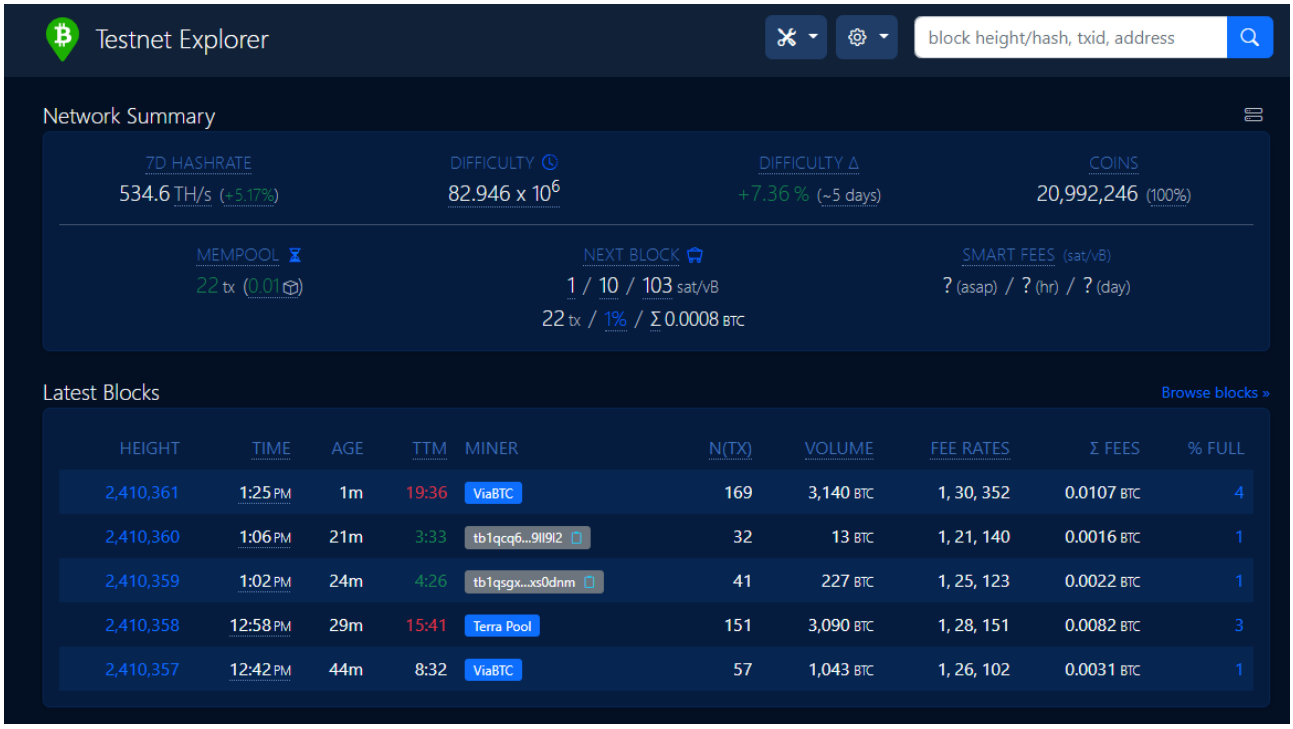


Figura 51: Web de btc-rpc-explorer.

Con este último apartado se da por finalizada la creación de un nodo local de Bitcoin.

5 Carga y tratamiento de datos

5.1 Carga de datos de Bitcoin en ELK

Se ha optado por usar Logstash para cargar datos en Elasticsearch. Para eso será necesario modificar el docker-compose que se utilizó en el capítulo 3. A continuación se hará una carga de datos desde fichero y por último, se cargarán los datos del nodo de Bitcoin

5.1.1. Añadir Logstash al entorno

En el fichero de docker-compose se ha eliminado el Beat de EtherBeat y se añade el servicio de Logstash

```
logstash:
  image: docker.elastic.co/logstash/logstash-oss:8.5.0
  container_name: logstash
  volumes:
    - "./logstash-data/pipeline:/usr/share/logstash/pipeline"
    - "./logstash-data/config:/usr/share/logstash/config"
  ports:
    - "5000:5000"
  environment:
    LS_JAVA_OPTS: "-Xmx512m -Xms512m"
  depends_on:
    - elasticsearch
  links:
    - elasticsearch
```

Figura 52: Servicio de Logstash en docker-compose.

Tras añadir este elemento, se recarga el docker-compose y ya está el nuevo servicio funcionando.

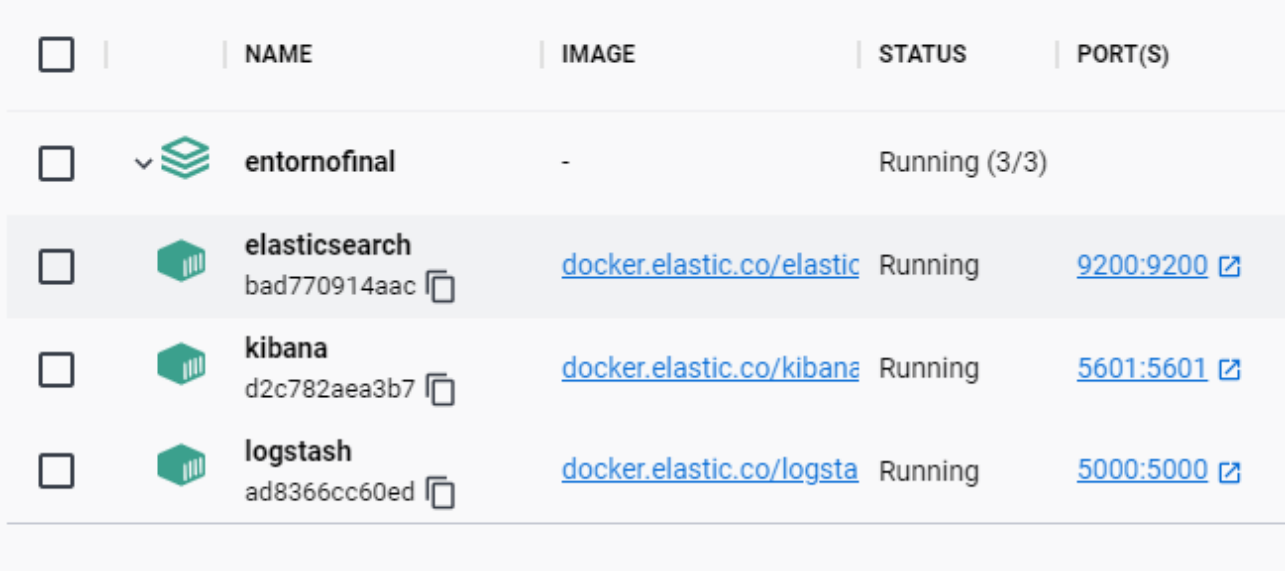


Figura 53: Servicios de ELK en Docker Desktop.

5.1.2. Creación de pipeline de prueba

Ahora se va a probar a cargar datos en Elasticsearch, para ello se crea un pipeline sencillo que lee datos de un fichero y los carga en Elasticsearch.

```
input {
   file {
      path => "/tmp/logstash-dataset"
   }
}

output {
   elasticsearch {
      hosts => ["elasticsearch:9200"]
      index => "logback-%{+YYYY-MM-dd}"
    }
}
```

Figura 54: Pipeline de logstash.

Este fichero se copia a la ruta /usr/share/logstash/pipeline y se reinicia el contenedor. Ahora cada vez que se cree en el directorio tmp un fichero con el nombre logstash-dataset, este se parsea y se carga en Elasticsearch. Se ve que en el log muestra el siguiente mensaje:

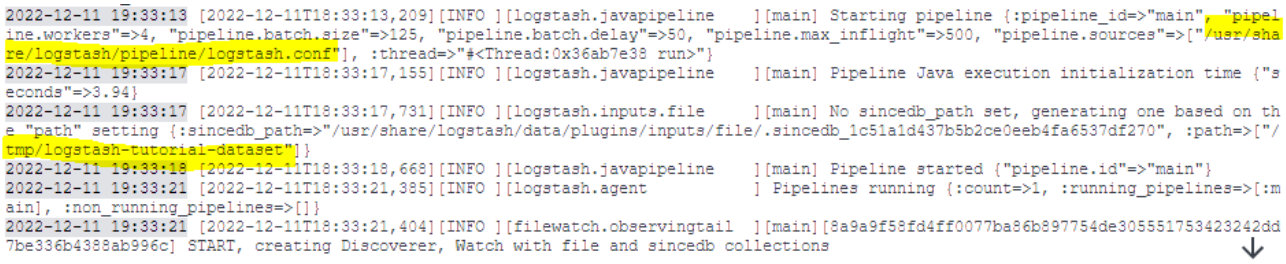


Figura 55: Logs de ejecución de pipeline de Logstash.

Se puede ver que ha cargado el pipeline */usr/share/logstash/pipeline/logstash.conf* y que está monitorizando la búsqueda del fichero */tmp/logstash-datatset.*

Al ir a Kibana, se ve que se ha creado el índice logback-2022-12-11.

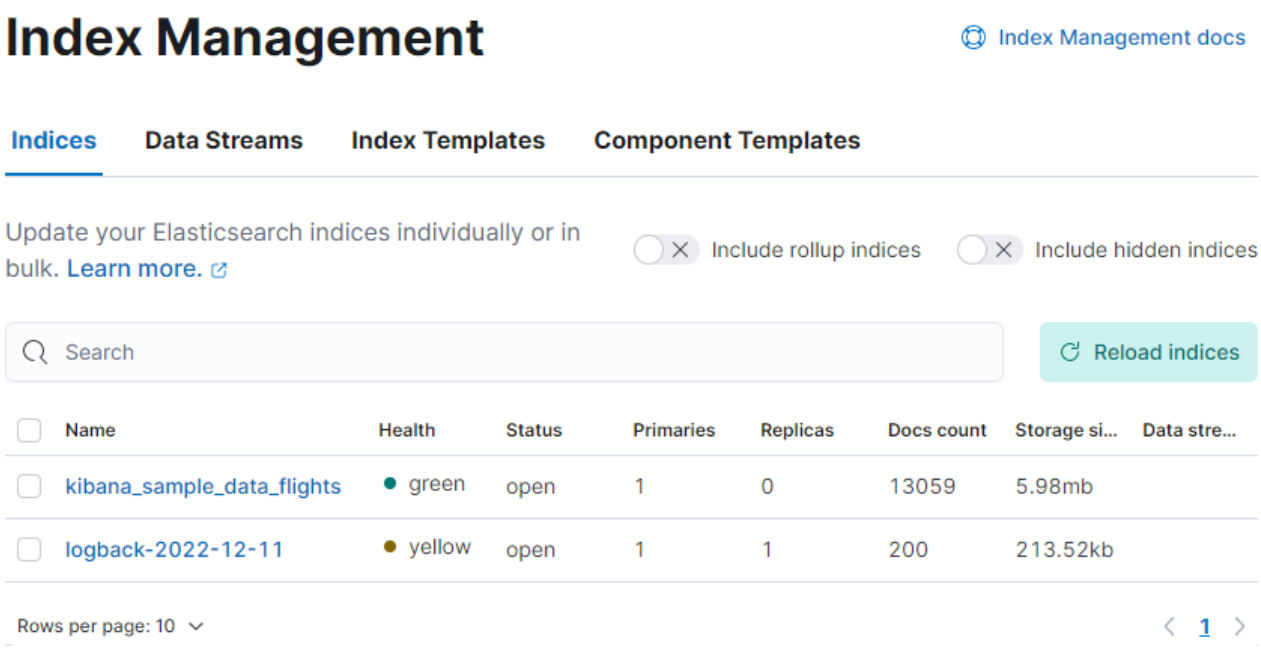


Figura 56: Kibana. Índices tras cargar fichero de logs.

5.1.3. Cargar datos de Bitcoin

Se va a proceder a cargar los datos del nodo de Bitcoin, para esto, como se comentaba en la introducción de este capítulo, es necesario encontrar la forma de acceder a los datos.

Se ha encontrado un plugin para la parte de input que permitirá acceder al nodo de Bitcoin. El plugin se llama logstash-input-blockchain[74], está desarrollado en Ruby y sigue el ejemplo base de logstash[75]

Para instalar el plugin es necesario:

- descargarse el código del repositorio
- compilarlo con Ruby, además ha sido necesario instalar la librerías de Ruby

```
apt-get install rubygems-integration

apt-get install ruby ruby-dev

gem build logstash-input-blockchain.gemspec
```

- instalarlo en logstash

```
/usr/share/logstash/bin/logstash-plugin                    install
/usr/share/logstash/config/logstash-input-blockchain-master/logstas
h-input-blockchain-0.1.0.gem
```

Ahora al listar los plugins instalados en Logstash, se puede ver que hay uno de tipo input para Blockchain.

```
$ bin/logstash-plugin list
Using bundled JDK: /usr/share/logstash/jdk
logstash-codec-avro
logstash-codec-cef
logstash-codec-collectd
(...)
logstash-codec-edn_lines
logstash-codec-es_bulk
logstash-codec-fluent
logstash-codec-graphite
logstash-input-beats
└── logstash-input-elastic_agent (alias)
logstash-input-blockchain
logstash-input-couchdb_changes
logstash-input-dead_letter_queue
logstash-input-elasticsearch
logstash-input-exec
logstash-input-file
logstash-input-ganglia
```

Figura 57: Logstash. Plugins instalados.

Se procede ahora a crear el pipeline que cargará los datos en Logstash con el nombre logstash_blockchain.conf

```
1  input {
2    blockchain {
3      protocol => "bitcoin"
4      host => "host.docker.internal"
5      port => 18332
6      user => "test"
7      password => "test"
8      granularity => transaction
9      interval => 1
10   }
11 }
12 output {
13     elasticsearch {
14         hosts => ["elasticsearch:9200"]
15         index => "bitcoin-init"
16     }
17 }
```

Figura 58: Fichero pipeline de logstash para cargar datos de Bitcoin.

Siguiendo las instrucciones de configuración del plugin

- línea 1: parámetros para el plugin de input
- línea 2: se indica el plugin a usar, en este caso blockchain.
- línea 3: se indica el protocolo. Con este plugin también es posible cargar datos de Ethereum.
- línea 4: nombre del host, como ya pasaba con btc-rcp-explorer, al ser una aplicación ejecutada en un contenedor docker es necesario usar ese host.
- línea 5: puerto de conexión, el puerto de testnet
- línea 6 y 7: usuario y contraseña
- línea 8: el plugin permite cargar bloques o transacciones. De cara a obtener más información, se ha elegido transacción.
- Línea 9: con el valor 1 se obtiene un bloque por segundo. El valor por defecto es 0, obtiene un bloque al terminar el siguiente. Se elige 1 para no saturar el sistema.
- línea 12: parámetros del plugin de output
- línea 13: el plugin de salida será elasticsearch
- línea 14: datos de acceso a elasticsearch
- línea 15: nombre del índice a crear. Se utiliza el nombre de bitcoin-init para cargar los datos de entrada.

Tras copiar este fichero a la carpeta de pipeline y reiniciar el contenedor, se revisa el log de Logstash y se ven las siguientes entradas.

Inicia el pipeline logstash_blockchain.conf

```
2022-12-11 15:37:39 [2022-12-11T14:37:39,221][INFO ][logstash.javapipeline    ][main] Starting pipeline {:p
ipeline_id=>"main", "pipeline.workers"=>4, "pipeline.batch.size"=>125, "pipeline.batch.delay"=>50, "pipelin
e.max_inflight"=>500, "pipeline.sources"=>["/usr/share/logstash/pipeline/logstash_blockchain.conf"], :threa
d=>"#<Thread:0x32a8a375 run>"}
2022-12-11 15:37:48 [2022-12-11T14:37:48,432][INFO ][logstash.javapipeline    ][main] Pipeline Java executi
on initialization time {"seconds"=>9.21}
2022-12-11 15:37:48 [2022-12-11T14:37:48,543][INFO ][logstash.inputs.blockchain][main] Setting up blockchai
n RPC client {:protocol=>"bitcoin", :host=>"host.docker.internal", :port=>18332}
2022-12-11 15:37:51 [2022-12-11T14:37:51,923][INFO ][logstash.javapipeline    ][main] Pipeline started {"pi
peline.id"=>"main"}
2022-12-11 15:37:52 [2022-12-11T14:37:52,274][INFO ][logstash.agent           ] Pipelines running {:count=>
1, :running_pipelines=>[:main], :non_running_pipelines=>[]}
2022-12-11 15:37:53 [2022-12-11T14:37:53,031][INFO ][logstash.inputs.blockchain][main][985f013110bc1d1557a6
8f3b43ce5ce38b49d2efab32936b0e66bd9a44f367b7] Retrieving latest block height {:height=>811872}
2022-12-11 15:37:53 [2022-12-11T14:37:53,034][INFO ][logstash.inputs.blockchain][main][985f013110bc1d1557a6
8f3b43ce5ce38b49d2efab32936b0e66bd9a44f367b7] Starting at block height {:height=>0}
2022-12-11 15:37:53 [2022-12-11T14:37:53,230][WARN ][logstash.inputs.blockchain][main][985f013110bc1d1557a6
8f3b43ce5ce38b49d2efab32936b0e66bd9a44f367b7] Could not find any information about transaction {:tx=>"4a5e1
e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"}
```

Figura 59: Log de Blockchain. Inicia pipeline de blockchain.

Se carga el output de elasticsearch y parece que conecta correctamente.

```
2022-12-11 15:09:01 [2022-12-11T14:09:01,272][INFO ][logstash.outputs.elasticsearch][main] New Elasticsearc
h output {:class=>"LogStash::Outputs::ElasticSearch", :hosts=>["//elasticsearch:9200"]}
2022-12-11 15:09:02 [2022-12-11T14:09:02,171][INFO ][logstash.outputs.elasticsearch][main] Elasticsearch po
ol URLs updated {:changes=>{:removed=>[], :added=>[http://elasticsearch:9200/]}}
2022-12-11 15:09:02 [2022-12-11T14:09:02,644][WARN ][logstash.outputs.elasticsearch][main] Restored connect
ion to ES instance {:url=>"http://elasticsearch:9200/"}
2022-12-11 15:09:02 [2022-12-11T14:09:02,679][INFO ][logstash.outputs.elasticsearch][main] Elasticsearch ve
rsion determined (8.5.0) {:es_version=>8}
```

Figura 60: Log de Logstash. Se conecta a elasticsearch.

Al ir de nuevo a Kibana, se ve que se ha creado el índice bitcon-init y que se han cargado ya más de 7000 documentos.

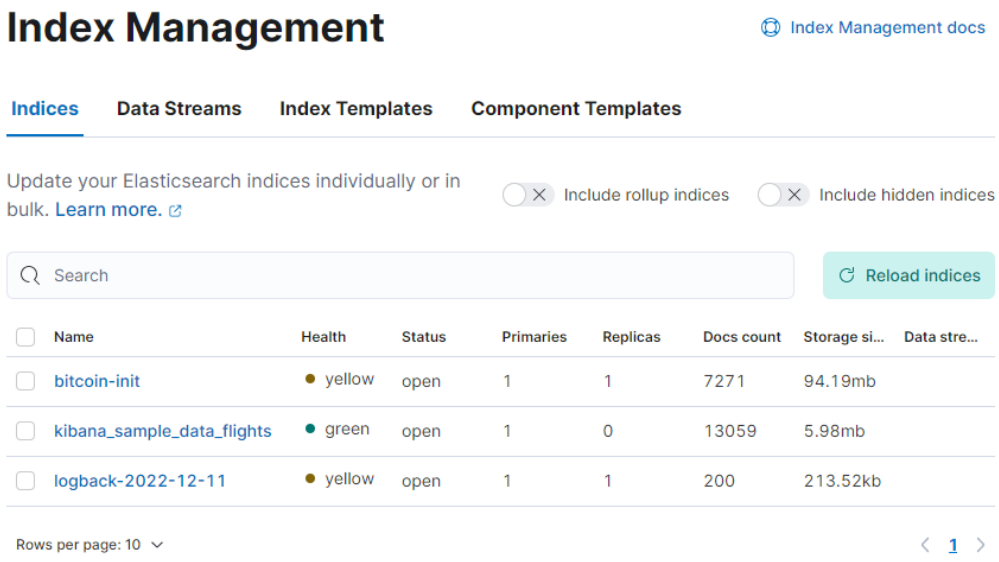


Figura 61: Kibana. Listado de índices.

Se procede a crear la Data View. Como Index pattern se elige bitcoin-*. Esto se hace porque, pensando en tener la información actualizada, se terminará de cargar todos los datos en un índice llamado bitcoin-init. Pero una vez terminado, se podría modificar el pipeline para que el nombre del índice se crease incluyendo la fecha en el índice, de esta forma, habría un índice por día, pero todos esos índices se mostrarían en el mismo Data View.

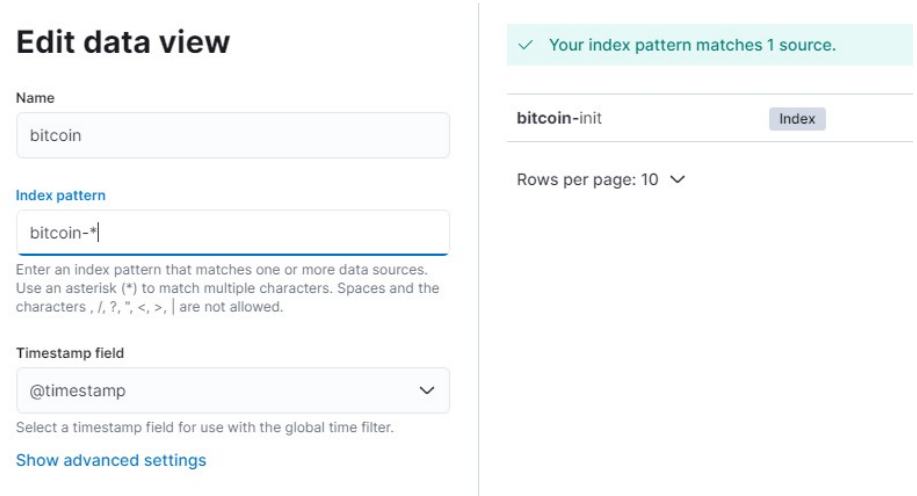


Figura 62: Kibana. Data view.

Se observa que al elegir el Index-pattern como bitcoin-*, existe un índice que cumple las condiciones.

Por último, en el apartado Discover de Kibana es posible ver los datos cargados.

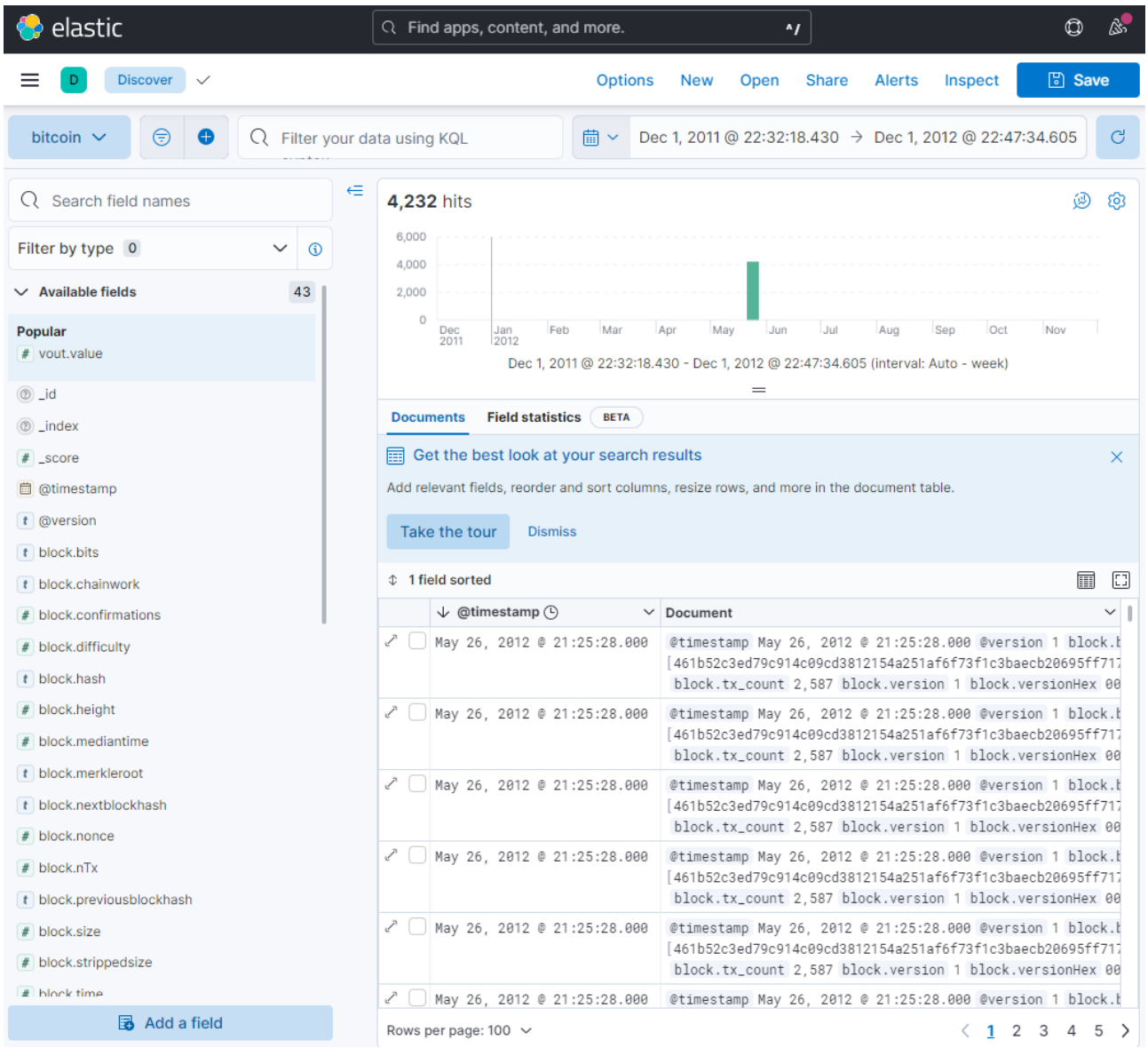


Figura 63: Kibana: apartado Discover.

5.2 Tratamiento de datos

5.2.1. Filtrado de datos

Una de las primeras tareas que se proponen es el filtrado de datos. Las razones para filtrar y tratar datos pueden ser muchas, puede ser para un ahorro de espacio, en este caso cabe recordar que se manejarán muchos datos. Podría ser porque hay datos redundantes o sobrantes. También es posible utilizar este sistema para obtener campos con datos calculados, unificar conceptos..

Llegados a este punto, es posible realizar esta tarea de forma muy sencilla gracias a la potencia de Logstash. Tal y como se indicaba en el apartado “1.4 Herramientas y tecnologías”, “Usando Logstash, se puede limpiar el dato usando filtros antes de enviarlo a Elastic.·

Logstash dispone de una gran cantidad de plugins de tratamiento de dato, en la web oficial es posible consultar la lista de plugins y sus descripciones[76].

A continuación se indica un ejemplo del uso del plugin Prune[77] para eliminar campos en los documentos a cargar. Para esto es necesario modificar el fichero de configuración del pipeline de carga de datos usando Logstash. En este caso se eliminará un dato que no aporta información, que es la versión de Bitcoin, puesto que todos los documentos hasta el momento tienen la el mismo valor.

En el apartado 5.1.3, se incluye el fichero usado para cargar los datos. Ahora se hace una modificación del mismo incluyendo el apartado filter:

En el fichero se han incluido las líneas 13-17 donde:

```
1  input {
2    blockchain {
3      protocol => "bitcoin"
4      host => "host.docker.internal"
5      port => 18332
6      user => "test"
7      password => "test"
8      granularity => transaction
9      interval => 1
10     start_height => 100000
11   }
12 }
13 filter {
14       prune {
15         remove_field => [ "version", "vsize", "locktime" ]
16       }
17     }
18 output {
19     elasticsearch {
20         hosts => ["elasticsearch:9200"]
21         index => "bitcoin_filter_v1"
22     }
23 }
```

Figura 64: Fichero de configuración de pipeline incluyendo filtro.

- línea 13 → apartado de filtros
- línea 14 → filtro a usar, en este caso prune.
- Línea 15 → configuración del filtro. En el ejemplo se eliminan los campos cuyo nombre contiene "version", "vsize", "locktime"

En la siguiente imagen se observa la diferencia entre el documento filtrado y el documento sin filtrar. Al revisar los documentos en detalle, se observa que ahora no está el campo versión.

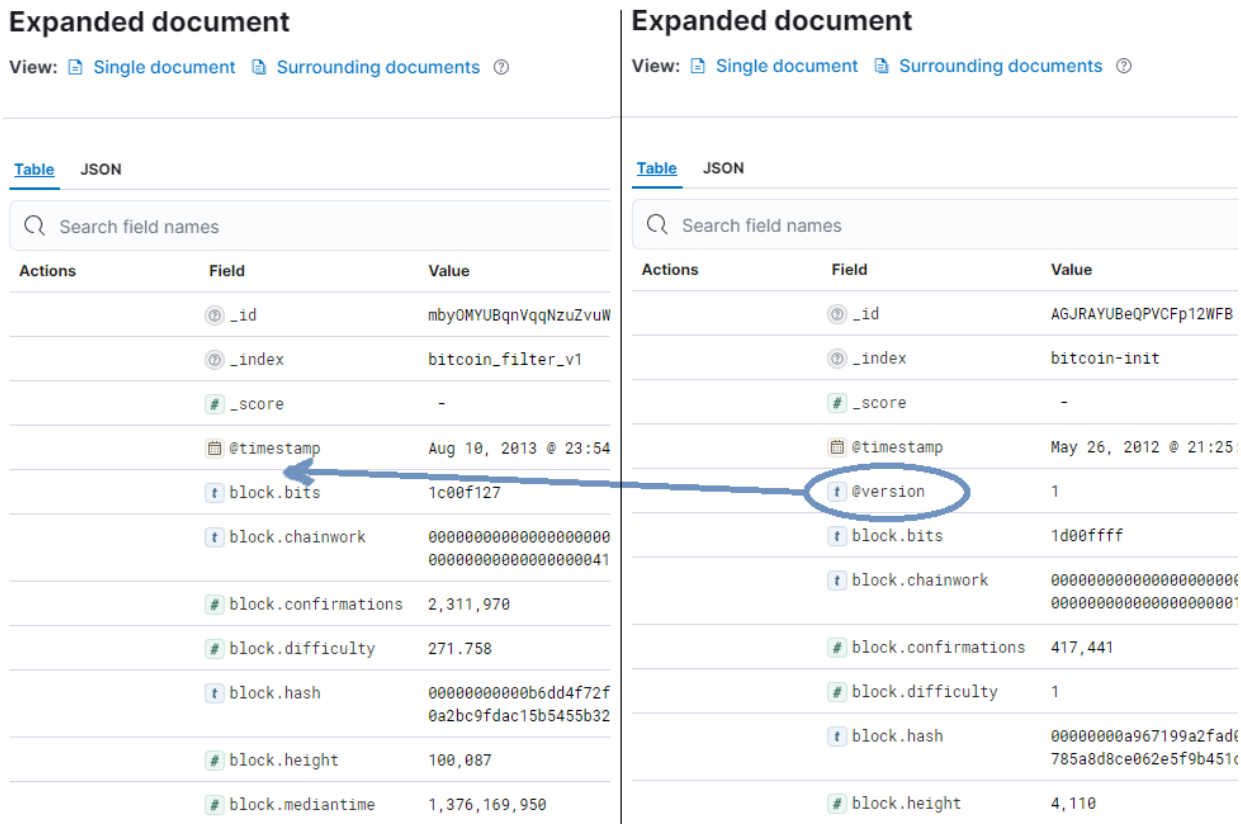


Figura 65: Comparativa de documentos tras insertar filtro.

5.2.2. Consultar información en varios índices

5.2.1.1 Introducción

Una de las características más interesantes que permite implementar Kibana es la posibilidad de crear visualizaciones consultando datos de varios índices. Se va a suponer que se quiere intentar asociar las direcciones de Bitcoin con un usuario. Si se obtiene un índice en el que se asocie esa dirección al usuario, sería posible consultar esa información. Esto puede realizarse asociando más de dos índices.

5.2.1.2 Cargar otro índice

De donde o como obtener los datos a cargar de ese otro índice se escapan del alcance de este proyecto, pero podría, por ejemplo, realizarse un escaneo de Twitter.

Se supone que se generará un fichero con un formato similar al siguiente


```
{"id":1,
    "vout.scriptPubKey.address":"mpTNatep9FEi2L52nLoYBLmcloVuveLDLW",
    "user":"Lucia",
    "timestamp": "2022-08-11T03:40:24.000Z"},
{"id":2,
    "vout.scriptPubKey.address":"mgCXSzqch9Hq6txaPHZm8BXWtmeLBQeoSz",
    "user":"Laura",
    "timestamp": "2022-07-12T03:40:24.000Z"},
{"id":3,
    "vout.scriptPubKey.address":"2MvzFjyi45C8TYnhfuX9mzZH6WKx44EfGY3",
    "user":"Carmen",
    "timestamp": "2022-03-17T03:40:24.000Z"},
{"id":4,
    "vout.scriptPubKey.address":"mtkbaiLiUH3fvGJeSzuN3kUgmJzqinLejJ",
    "user":"Roi",
    "timestamp": "2022-02-28T03:40:24.000Z"}
```


Figura 66: Datos de índices con usernames.

Usando Logstash se carga el índice en Kibana, se crea el Data View [bitcoin_address] y en Discover es posible consultar los datos cargados.

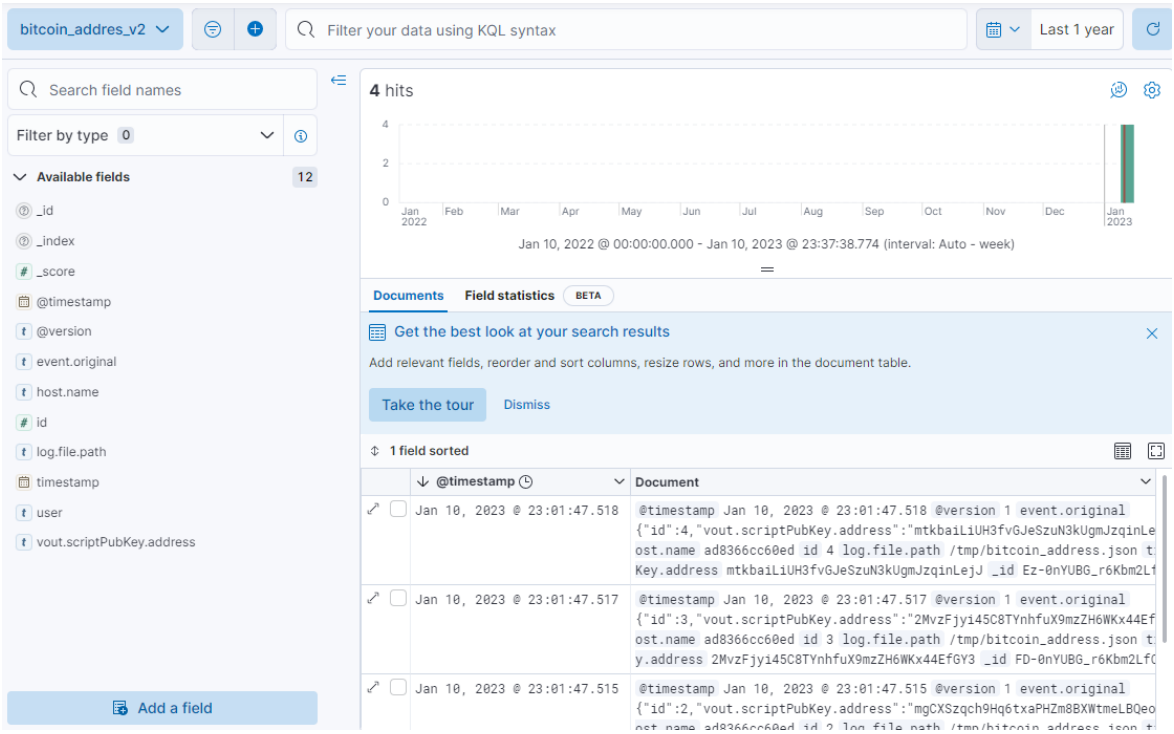


Figura 67: Discover de Bitcoin_address_v2.

5.2.1.3 Crear Dashboard

Se va a crear ahora un Dashboard que incluya visualizaciones de ambos índices.

Se crearán varias visualizaciones de Lens, dos para el Data View de *Bitcoin* y una para la de *bitcoin_address*

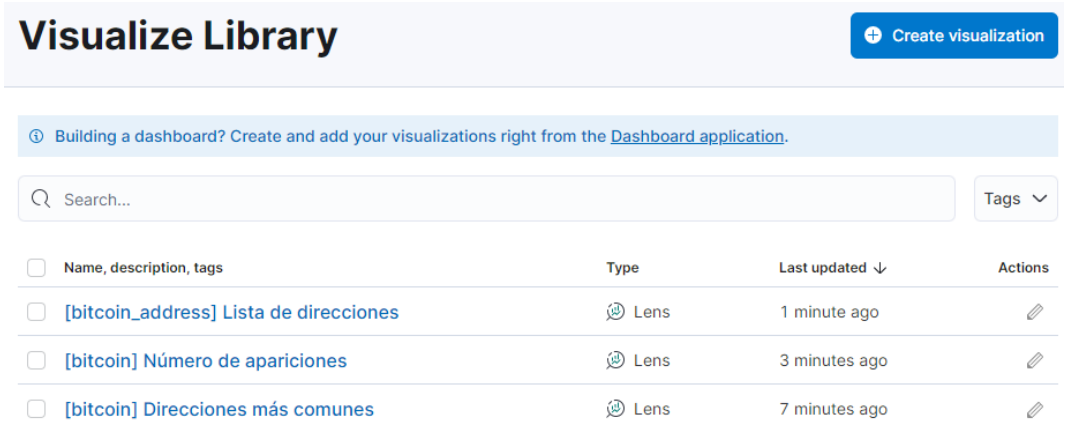


Figura 68: Kibana: Visualize Library.

- *[bitcoin] Direcciones más comunes*: las 10 cuentas más frecuentes en los datos de Bitcoin cargados

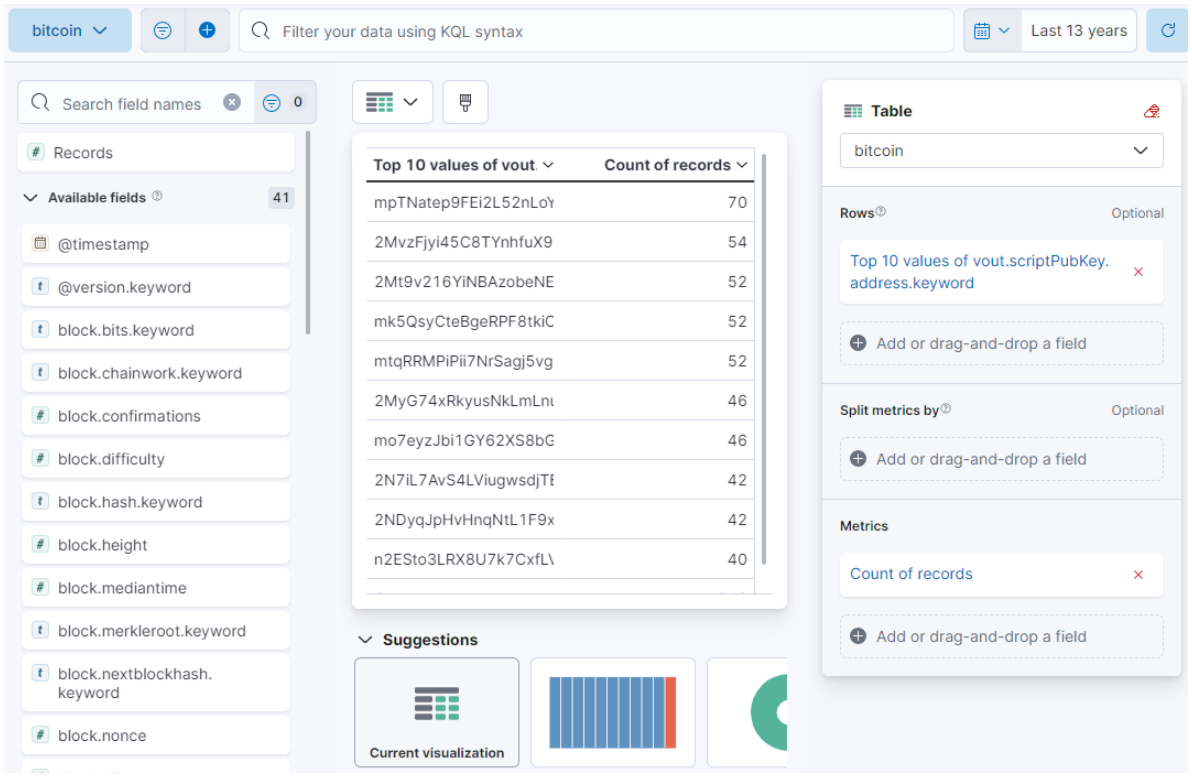


Figura 69: Kibana: Visualización de las 10 cuentas más comunes.

- *[bitcoin] Número de apariciones:* Número de veces que aparece esa cuenta.

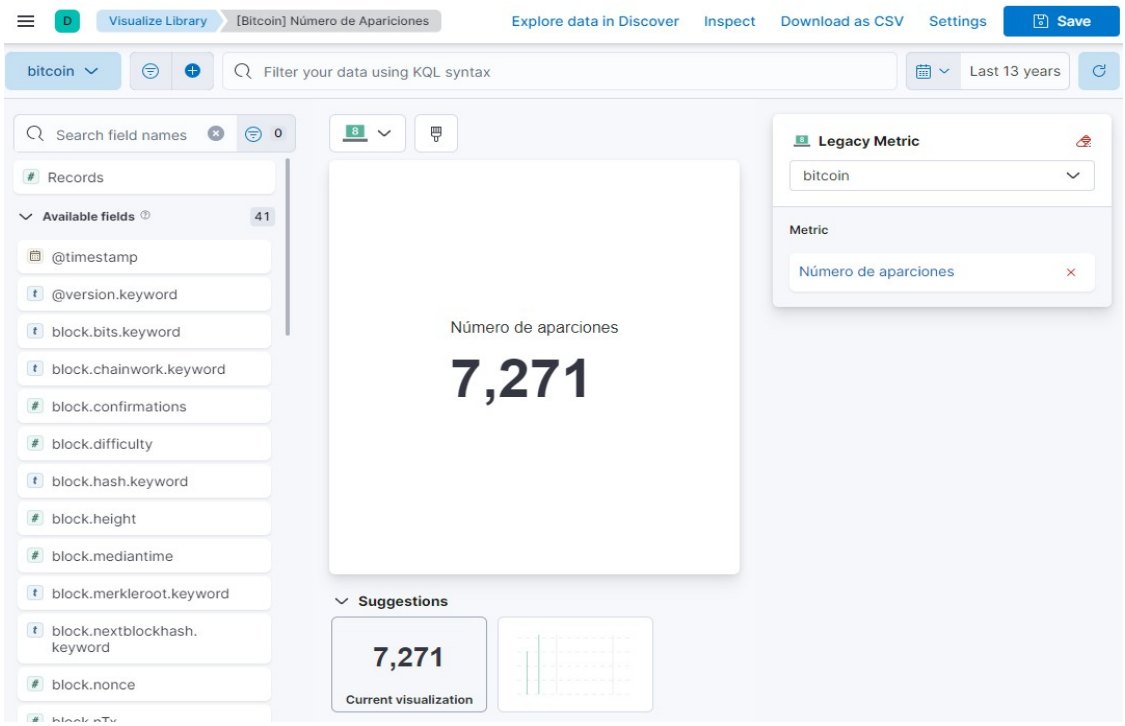


Figura 70: Kibana: Visualización Lens Número de apariciones.

- *[bitcoin_address] Lista de direcciones*. Muestra todas las cuentas en índice bitcoin_address asociadas al nombre de usuario.

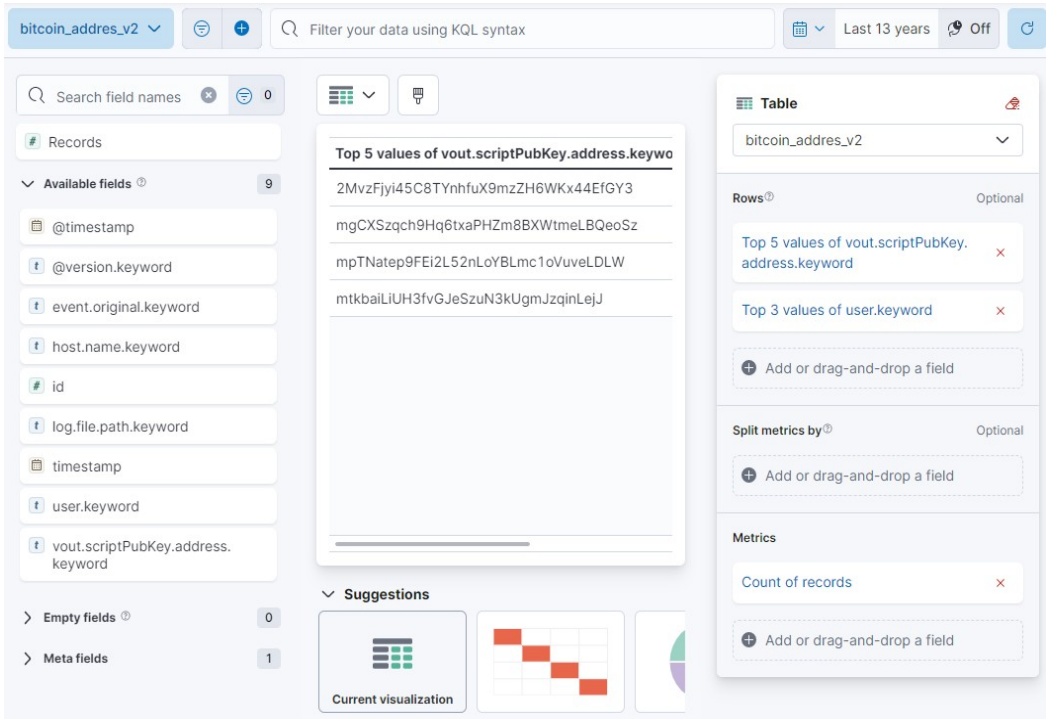


Figura 71: Kibana: Lista de direcciones de bitcoin_address.

Se crear ahora un Dashboard con estas tres visualizaciones y se añade un elemento de control, el cual permitirá seleccionar un número de cuenta. Cuando se crea este tipo de elemento debe seleccionarse:

- el Data View, en este caso se usará el de *bitcoin*
- el Field, se usará vout.scriptPubKey.address.keyword
- el tipo: será de tipo *Option list.*

Y es aquí es donde se debe ser especialmente cuidadoso. Cuando se crea un control, en este se permitirá igualar un campo a un valor, para que los Dashboards con varios índices funcionen de forma adecuada, deben tener campos comunes. En este caso, el campo que incluye las direcciones se llama en ambos índices: vout.scriptPubKey.address.keyword.

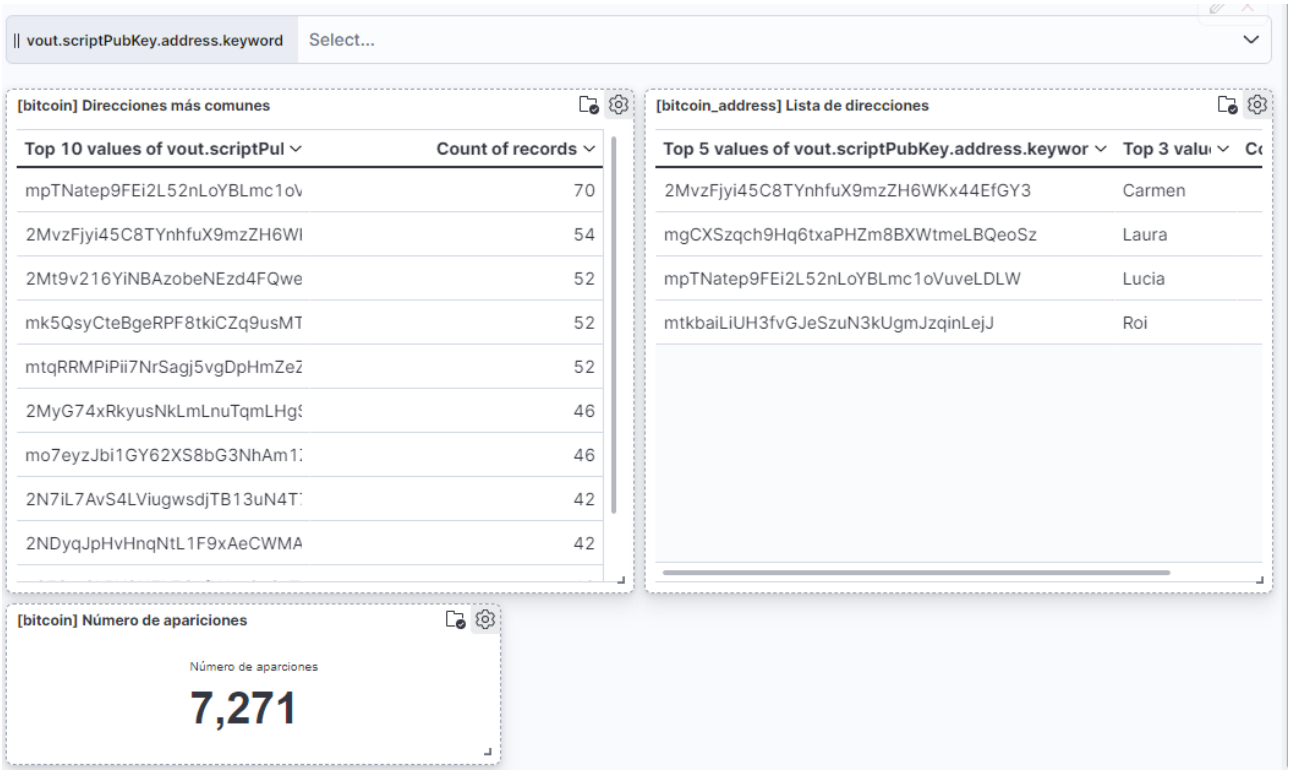


Figura 72: Kibana: Dashboard con visualizaciones de varios índices.

Si en el control se elige la address más común, se actualizará la información de todas las visualizaciones

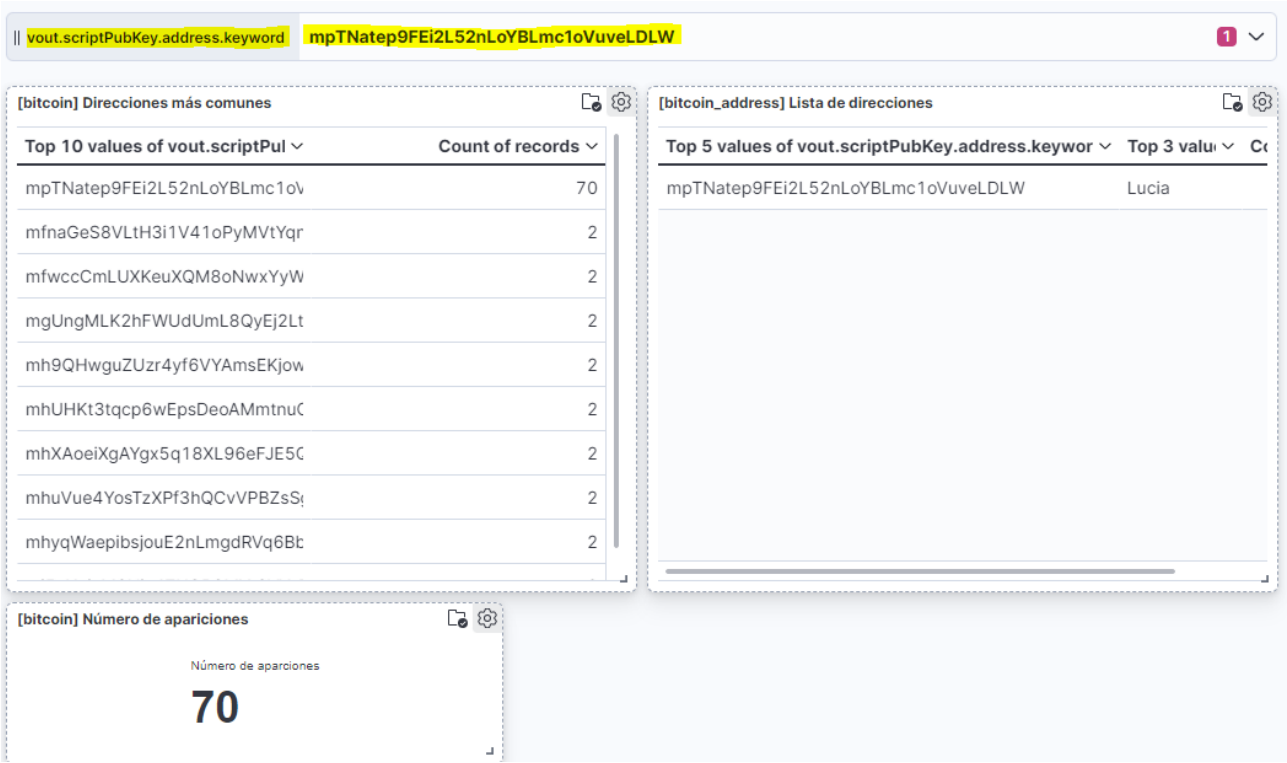


Figura 73: Dashboard con datos filtrados.

Si se comparan las imágenes, ahora los datos que se muestran se corresponden sólo con aquellos documentos cuya address sea la seleccionada en el control.

Cabe comentar, que en la lista de cuentas más comunes aparecen otras cuentas, porque en varios documentos, se incluyen transacciones a más de una cuenta y esas cuentas son las que acompañan en esos documentos. En un caso real, sería muy interesante investigar sobre esas direcciones.

5.2.1.4 Crear acceso a otros paneles

Por último, es posible añadir enlaces en las visualizaciones, que en Kibana llaman Drilldown, que permiten realizar ciertas acciones, en este caso se a configurar para llamar a un panel con más información sobre esa cuenta.

Se crea el Drilldown apuntando a otro panel.

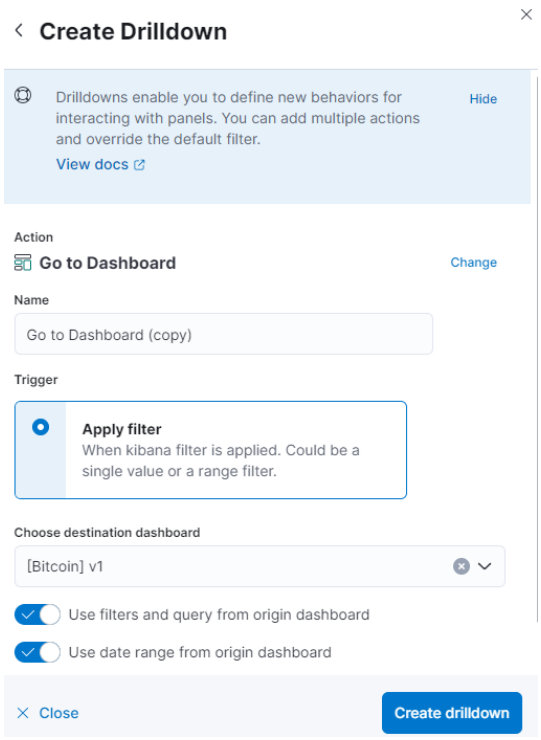


Figura 74: Dashboard: Crear drilldown.

Se puede observar que la visualización tiene un drilldown activado

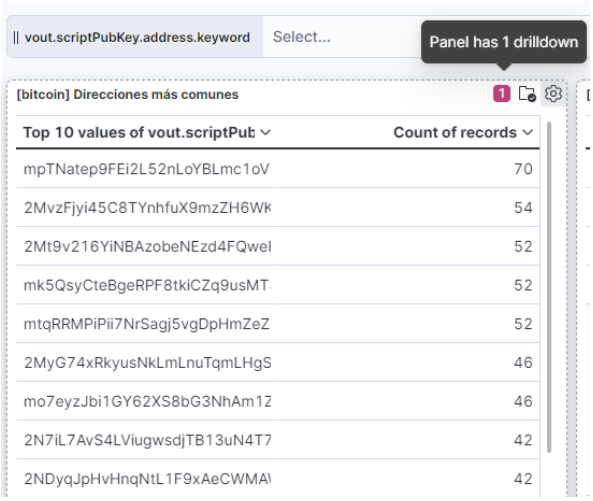


Figura 75: Dashboard: vista de drilldown activado.

Al pulsar sobre el elemento, se muestra el enlace “Go to Dashboard”

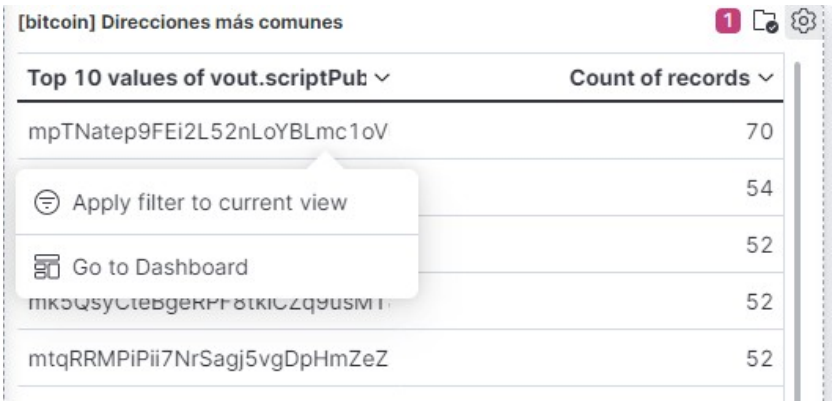


Figura 76: Dashboard: Usar el drilldown.

Al pulsar en el enlace, se muestra el Dashboard enlazado pasando como filtro la address.

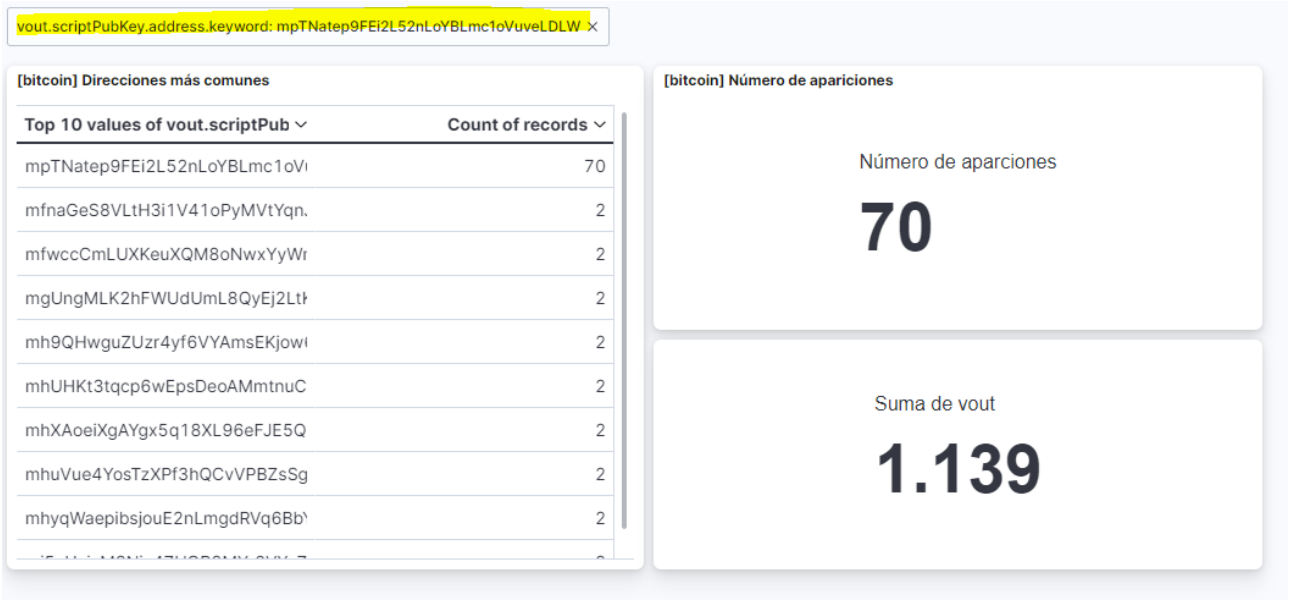


Figura 77: Dashboard enlazado con drilldown.

6 Conclusiones y trabajos futuros

6.1 Conclusiones

6.1.1. Introducción

Durante la realización del proyecto se han llegado a distintas conclusiones acerca de varios aspectos.

El proceso ha sido más complicado de lo que parecía en un principio, la información que se encuentra en Internet para realizar este tipo de tareas es bastante antigua, la mayoría tiene entre 3 y 5 años y en tecnología es mucho tiempo, eso hace que muchas de las soluciones que se aportan ya no funcionan o que las tecnologías que emplean no sean sencillas de encontrar. No sé la razón por la que todas estas soluciones tienen ese tiempo, es posible que el interés de las criptomonedas se haya estancado y por lo tanto, no se invierta tanto tiempo en hacer aplicaciones para las mismas.

Además, como se indicaba en la introducción, el hecho de trabajar con criptomonedas y la cantidad de datos que generan hacen que el tratamiento de los mismos necesite de un hardware con ciertas prestaciones, lo que complica el proceso.

Entrando en una análisis más detallado, se hablará primero de las herramientas elegidas para la ejecución del proyecto.

- La ejecución de todas las herramientas en contenedores docker ha facilitado la investigación, porque permite probar de forma rápida y sencilla distintas versiones de productos y elegir el que más se adecuaba a los objetivos planteados. Esta elección ha sido un acierto.
- La primera aproximación que se hizo para la carga de datos, resultó que no era la más acertada, pero creo que el camino elegido ha sido correcto. Logstash es uno de los elementos de cabecera de ELK y su uso ha aportado sencillez a las tareas de tratamiento de datos.
- La elección de ELK como BD para el tratamiento de datos también parece adecuada. Además de poder realizar tratamientos sobre los datos, permite combinar datos de distintas fuentes lo que aporta fortaleza a la solución.

6.1.2. Resultados

Se comprobará si se han cumplido los objetivos del proyecto.

En cuanto a los objetivos del trabajo:

- OBJ-01: Crear un entorno de ELK para la carga de datos de una blockchain.
  - Se ha conseguido el objetivo de forma exitosa usando tecnología docker que aporta sencillez y facilidad de mantenimiento del entorno.
- OBJ-02: Investigar la forma de cargar datos de Bitcoin.
  - La investigación ha dado los resultado deseados, a pesar de que ha sido necesario replantearse el sistema que se había seleccionado en un principio.
- OBJ-03: Cargar y tratar datos de Bitcoin.
  - Se han cargado los datos de forma correcta utilizando Logstash y un plugin para conectarse a la red de Bitcoin. Se aporta un ejemplo de tratamiento de datos que podrá ampliarse en el futuro.

6.2 Trabajos futuros

En este apartado se irá haciendo una lista de tareas a realizar. Tratarán de abordarse esas tareas durante el desarrollo, pero si no fuese así, quedarán recogidas para desarrollos futuros.

- Modificaciones en entorno docker
  - usar certificados digitales.
  - Crear varios nodos para elasticsearch
  - Hacer uso de *volumes* no *linkados* para la persistencia de datos.
  - Crear una imagen de docker que incluya la instalación del plugin o un script de instalación que ejecute todos los pasos.
- Modificar el fichero de configuración del pipeline de Logstash para cargar sólo los datos que aporten información.
- Crear un nodo que se conecte a la red de Bitcoin main.
  - Sincronizar el nodo de forma completa.
  - Cargar todos los datos en ELK.

- Buscar otras fuentes de información para enriquecer los índices en ELK y dar respuesta a preguntas más complejas. Se podría utilizar para hacer seguimiento del dinero, para la investigación del blanqueo de capitales...

7 Glosario

Blockchain: es la tecnología que se creó originalmente para albergar a las criptomonedas como Bitcoin o Ethereum.

Dapp: Aplicación donde alguno de sus componentes más relevantes está descentralizado.

SmartContract: o Contrato inteligente, es un contrato que es capaz de ejecutarse y hacerse cumplir por sí mismo, de forma automática, sin intermediarios. Son los programas que se ejecutan en una Blockchain.

Wallet: permite guardar las cuentas, software o hardware diseñado para almacenar cuentas.

Addess: cuando se crea una cuenta, se crea la clave privada y la pública, a partir de esta última se crea la dirección o address, es un código en hexadecimal que identifica la cuenta:

0xF5d7968eDfAF9A746E71928969569ba41a7dbA53

Transacción: mensajes que contienen información y que van firmados por el que los emite

Bloque: conjunto de transacciones.

Inmutable: crea una especie libro de cuentas distribuido.

Transparente: cualquiera puede verificar que la información registrada es correcta.

Descentralizado: la información registrada recae en múltiples participantes.

IaC: Infraestructura as Code. es el proceso de gestión y aprovisionamiento de sistemas a través de archivos de definición legibles por máquina, en lugar de configuración de hardware físico o herramientas de configuración interactiva.

Índices invertidos: es una forma de estructurar la información que va a ser recuperada por un motor de búsqueda. A diferencia de lo que sucede en una base de datos tipo SQL, donde el índice ha sido definido a priori, en el índice invertido el índice se crea a posteriori, cuando el motor ha analizado los documentos sobre los que se basará la búsqueda.

Visualización en Kibana:formas de ver datos. Pueden ser tablas, gráficos, mapas...

Dashboard de Kibana: recopilación de gráficos, grafos, métricas, búsquedas y mapas. Permiten obtener información de un vistazo sobre datos de varias perspectivas y permiten a los usuarios explorar los detalles.

8 Bibliografía

[1] RAE (edición 2022) https://dle.rae.es/

[2] Equipo de ELK (Versión 8.5) https://www.elastic.co/

[3] Blockchain: Apuntes de la asignatura Sistemas de Blockchain del Máster en Ciberseguridad y Privacidad de la UOC.

[4] Centro Nacional de Inteligencia(2022) https://www.cni.es/la-inteligencia

[5] Jaime Fábregas (16 de dicimebre 2021) We tracked 800 million transactions in the Ethereum Blockchain. Here is how we did it. https://www.tarlogic.com/blog/download-ethereum-blockchain/

[6] Evgeny Medvedev (27 de marzo de 2018) How to Export the Entire Ethereum Blockchain to CSV in 2 hours for $10.

https://medium.com/coinmonks/how-to-export-the-entire-ethereum-blockchain-to-csv-in-2-hours-for-10-69fef511e9a2

[7] Equipo de ELK (Versión 8.5) Install Elasticsearch with Docker.

https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html

[8] ChainAnalisys (2022) https://www.chainalysis.com/

[9] I-Intelligence (2022) https://i-intelligence.eu/resources/osint-toolkit

[10] Elliptic(2022) https://www.elliptic.co/

[11] Blockchain Intelligence Group.(2022)

https://blockchaingroup.io/solutions/?utm_medium=Paid&utm_source=Google&utm_campaign=Chainalysis

[12] Chainanalisys (2022) Reactor Certificacion https://www.chainalysis.com/crc/

[13] Renata Luebertaité (19 de abirl de 2022) 9 Best On-Chain Analysis Tools In 2022.

https://dexterlab.com/best-on-chain-analysis-tools/

[14] Packt (2022) What is ELK Stack?

https://subscription.packtpub.com/book/big-data-and-business-intelligence/9781788831031/1/ch01lvl1sec10/what-is-elk-stack

[15] Equipo de ELK (Versión 8.5) Instalar Elastic con Docker desktop:

https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html#docker

[16] Changzheng Liu (18 de octubre de 2019) Index Ethereum Blocks into Elasticsearch via Etherbeat

https://medium.com/@hatricker/index-ethereum-blocks-into-elasticsearch-via-etherbeat-33fcbc14dbed

[17] Anthony Lapenna (septiembre 2022) Repositorio de GitHub

https://github.com/deviantony/docker-elk

[18] Ritesh Agrawal (1 de mayo 2021) How to connect Ganache with Metamask and deploy Smart contracts on remix without

https://dapp-world.com/blogs/01/how-to-connect-ganache-with-metamask-and-deploy-smart-contracts-on-remix-without-1619847868947

[19] Engine Yard Team (18 de marzo de 2022) Docker Vs Virtual Machines Explained https://www.engineyard.com/blog/docker-vs-virtual-machines-explained/

[20] HashiCorp Vagrant (2022) https://www.vagrantup.com/

[21] HashiCorp Terraform (2022) https://www.terraform.io/

[22] Wikipedia (13 de diciembre 2022) Infraestructura como código (IaC)

https://es.wikipedia.org/wiki/Infraestructura_como_c%C3%B3digo

[23] Kirill Shirinkin (16 de noviembre 2020) Terraform vs Docker

https://stackshare.io/stackups/docker-vs-terraform

[24] Eduardo Barrios (15 de junio 2020) ¿Qué demonios es Docker y Docker-Compose? y cómo Dockerizar Dotnet Core WebApi y SQL Server en un ambiente de desarrollo ideal https://dev.to/ebarrioscode/que-demonios-es-docker-docker-compose-y-como-dockerizar-dotnet-core-webapi-y-sql-server-en-un-ambiente-de-desarrollo-ideal-95a

[25] Kubernetes (2022) https://kubernetes.io/

[26] Wikipedia (27 de agosto 2022) Índices invertidos

https://es.wikipedia.org/wiki/%C3%8Dndice_invertido

[27] Docker desktop (version 4.15.0) https://www.docker.com/products/docker-desktop/

[28] Equipo de ELK (Versión 8.5) Instalar Elasticsearch en un nodo multi-clúster https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html#docker-compose-file

[29] Sysctl-explorer(09 de diciembre de 2018) max_map_count https://sysctl-explorer.net/vm/max_map_count/

[30] Equipo de ELK (Versión 8.5) Elastic subscription https://www.elastic.co/guide/en/enterprise-search/current/license-management.html

[31] Equipo de ELK (Versión 8.5) Elastic Observability: https://www.elastic.co/guide/en/observability/current/observability-introduction.html

[32] Equipo de ELK (Versión 8.5) Community Beats https://www.elastic.co/guide/en/beats/libbeat/current/community-beats.html

[33] Linux Post Install (16 de junio 2020) Truffle Suite: Herramientas de código abierto para Blockchain https://blog.desdelinux.net/truffle-framework-herramientas-codigo-abierto-blockchain/

[34 Ganache (v2.5.4) https://trufflesuite.com/ganache/]

[35] Index Ethereum Blocks into Elasticsearch via Etherbeat https://medium.com/@hatricker/index-ethereum-blocks-into-elasticsearch-via-etherbeat-33fcbc14dbed

[36] Changzheng Liu (29 de septiembre 2019) repositorio de Gitlab https://gitlab.com/hatricker/etherbeat

[37] Visual Studio(21 de junio de 2022) Go in Visual Studio Code. https://code.visualstudio.com/docs/languages/go

[38] Google Open source (28 de enero de 2021) Getting started with VS Code Go [Video] YouTube https://www.youtube.com/watch?v=1MXIGYrMk80

[39] GitHub Action (3 de agosto de 2022), repositorio de GitHUb, https://github.com/golang/vscode-go/wiki/debugging

[49] Mónica Sarbu(14 de julio de 2018) Build your own Beat https://www.elastic.co/es/blog/build-your-own-beat

[50] Audrey Roy and Cookiecutter community (Revision e9b3b841) Instalación de Coolkiecutter https://cookiecutter.readthedocs.io/en/latest/installation.html

[51] Ge Comunity() Información de Go. https://go.dev/

[52] Go community() Download and Install https://go.dev/doc/install

[53] Microsoft () Visual Studio Code https://code.visualstudio.com/

[54] George Bridgeman (01 de diciembre de 2021) Custom Beats Deprecated in 7.16 https://georgebridgeman.com/posts/custom-beats-deprecated/

[55] George Bridgeman (21 de mayo de 202) Creation a Custom Elasticsearch Beat https://georgebridgeman.com/posts/creating-a-custom-beat/

[56] Go Community (02 de agosto de 2022) Release History de versiones de Go. https://go.dev/doc/devel/release

[57] Denis Leonov (23 de febrero de 2022) Blockchain parser. Repositorio de GitHub https://github.com/ragestack/blockchain-parser

[58] Elastic (versión 8.5) Logstash Reference https://www.elastic.co/guide/en/logstash/current/input-plugins.html

[59] Elastic (versión 8.5) Logstash Reference. Output plugins

https://www.elastic.co/guide/en/logstash/current/plugins-outputs-elasticsearch.html

[60] Jeff Garzik (2011) bitcoinlib.services.authproxy module https://bitcoinlib.readthedocs.io/en/latest/source/bitcoinlib.services.authproxy.html

[61] Programcreek (-) Python bitcoinrpc.authproxy.AuthServiceProxy() Examples https://www.programcreek.com/python/example/94600/bitcoinrpc.authproxy.AuthServiceProxy

[62] Python Elasticsearch client (2022 , v8.5.3) https://elasticsearch-py.readthedocs.io/en/v8.5.3/api.html

[63] Josh Bressers (29 de diciembre 2019) Repositorio de GitHub blockchain-elasticsearch https://github.com/joshbressers/blockchain-elasticsearch

[64] Blockchain (-) Blockchain.com APIx https://www.blockchain.com/explorer/api

[65] Blockchain (-) Blockchain.com Query API https://www.blockchain.com/explorer/api/q

[66] Ycharts (10 de diciembre de 2022) Bitcoin Blockchain size https://ycharts.com/indicators/bitcoin_blockchain_size

[67] Bit2me (-) Testnet, la red de pruebas para Bitcoin https://academy.bit2me.com/testnet-red-pruebas-bitcoin/

[68] BlockChair (2022) BlockChair https://blockchair.com/bitcoin/testnet

[69] Aitor Ibañez (18 de mayo de 2022) Montar nodo de Bitcoin e interactuar con él https://www.youtube.com/watch?v=CIrk3ReeeCk&list=PLocA6zRHOS_pB63Oq5uikm2AAHGkgIpDp&index=1

[70] BitcoinCore (-) Download Bitcoin Core https://bitcoin.org/en/download

[71] Grokking Bitcoin (2018) Using bitcoin-cli https://livebook.manning.com/book/grokking-bitcoin/a-using-bitcoin-cli/v-6/

[72] Bitcoin Testnet Faucet (-) Get Testnet Coins https://bitcoinfaucet.uo1.net/

[73] Dan Janosik (9 de diciembre de 2022) Repositorio de github https://github.com/janoside/btc-rpc-explorer

[74] Valentin Crettaz (11 de junio de 2017) Repositorio de GitHub https://github.com/consulthys/logstash-input-blockchain

[75] Grag Back, Julien Mailleret, Pius (-) Repositorio GitHUb https://github.com/logstash-plugins?query=example

[76] Equipo de ELK (Versión 8.5) Documentación de filtros de Logstah https://www.elastic.co/guide/en/logstash/current/filter-plugins.html

[77]Equipo de ELK (Versión 8.5) Documentación de filtro Prune de Logstah https://www.elastic.co/guide/en/logstash/current/plugins-filters-prune.html

[78] BOE (22 de febrero de 2018) Resolución de 22 de febrero de 2018, de la Dirección General de Empleo, por la que se registra y publica el XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública.

https://www.boe.es/diario_boe/txt.php?id=BOE-A-2018-3156

[79] Talent.com(enero 2023) Salario medio para Jefe De Proyecto en España, 2023 https://es.talent.com/salary?job=jefe+de+proyecto

9 Anexos

9.1 Anexo I

Información obtenida de sysctl-explorer.net[30]

Sysctl Explorer

vm / max_map_count

max_map_count

file: /proc/sys/vm/max_map_count
variable: vm.max_map_count

Official reference

This file contains the maximum number of memory map areas a process may have. Memory map areas are used as a side-effect of calling malloc, directly by mmap, mprotect, and madvise, and also when loading shared libraries.

While most applications need less than a thousand maps, certain programs, particularly malloc debuggers, may consume lots of them, e.g., up to one or two maps per allocation.

The default value is 65536.

source

Last update: 2018-12-09 18:93:01 UTC

Sysctl Explorer is available under the GPL-2.0.
Kernel documentation © the Linux kernel authors.
Linux® is a registered trademark of Linus Torvalds.

9.2 Anexo II: Docker-compose utilizado en fase de creación del entorno de trabajo.

```
version: '3'

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.5.0
    container_name: elasticsearch
    environment:
      - node.name=es01
      - discovery.type=single-node
      - xpack.security.enabled=false
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - elasticsearch-data:/usr/share/elasticsearch/data
    ports:
      - 9200:9200

  kibana:
    image: docker.elastic.co/kibana/kibana:8.5.0
    container_name: kibana
    ports:
      - 5601:5601
    environment:
      ELASTICSEARCH_HOSTS: http://elasticsearch:9200

  ethereum:
    image: etherbeat:1.1
    environment:
      ELASTIC_HOST: "elasticsearch:9200"
      ETH_RPC_ADDR: $ETH_RPC_ADDR
    depends_on:
      - kibana
      - elasticsearch

volumes:
  elasticsearch-data:
```

9.3 Anexo III: Docker-compose utilizado en fase de análisis.

```
version: '3'

x-logging:
  &default-logging
    options:
      max-size: '10m'
      max-file: '5'
    driver: json-file

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.5.0
    container_name: elasticsearch
    logging: *default-logging
    environment:
      - node.name=es01
      - discovery.type=single-node
      - xpack.security.enabled=false
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - elasticsearch-data:/usr/share/elasticsearch/data
    ports:
      - 9200:9200

  kibana:
    image: docker.elastic.co/kibana/kibana:8.5.0
    container_name: kibana
    logging: *default-logging
    ports:
      - 5601:5601
    environment:
      ELASTICSEARCH_HOSTS: http://elasticsearch:9200

  logstash:
    image: docker.elastic.co/logstash/logstash-oss:8.5.0
    container_name: logstash
    logging: *default-logging
    volumes:
      - "C:/TFM/entorno/logstash-data/pipeline:/usr/share/logstash/pipeline"
      - "C:/TFM/entorno/logstash-data/config:/usr/share/logstash/config"
    ports:
      - "5000:5000"
    environment:
      LS_JAVA_OPTS: "-Xmx512m -Xms512m"
    depends_on:
      - elasticsearch
    links:
      - elasticsearch

volumes:
  elasticsearch-data:
```

9.4 Anexo IV: Pasos a seguir para ejecutar el entorno final

En el trabajo se han ido indicado los pasos necesarios para crear el entorno completo, en este apartado se hará un resumen de los mismos.

9.4.1 Hardware empleado

El entorno de trabajo se ha creado en una máquina personal con las siguientes características:

- Intel(R) core™ i5-4300U CPU @ 1.90GHz 2.50 GHz
- RAM instalada: 8.00 GB
- Sistema operativo: Windows 10 Pro

9.4.2. Crear nodo de local de Bitcoin

Para crear el nodo local se utiliza el software oficial de Bitcoin, Bitcoin-core[70] que puede obtenerse de la página oficial.

- Se descarga el código y se instala.
- Se crea el fichero bitcoin.conf para indicar algunas de las propiedades que se necesita que tenga el nodo. Si no se crea este fichero, el nodo se conecta a la red Main de Bitcoin. El fichero debe crearse en Windows 10 en la ruta *C:\Users\<username>\AppData\Roaming\Bitcoin*

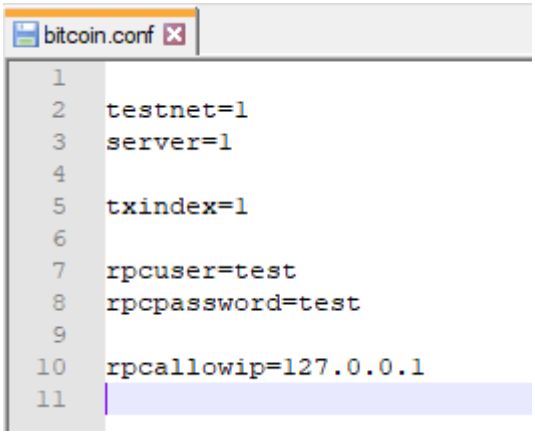


Figura 78: Bicoin.conf.

- Desde *C:\Program Files\Bitcoin\daemon*, se lanza la aplicación ejecutando bitcoind.exe

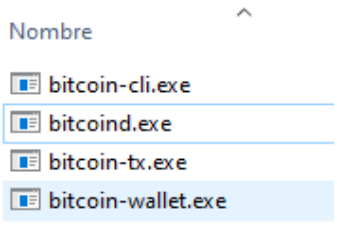


Figura 79: Ejecutables de BitcoinCore.

El nodo empieza a sincronizar, en este caso tardó sobre 12 horas en sincronizar toda la red de testnet.

```
λ bitcoind.exe
2022-12-11T17:30:17Z Bitcoin Core version v22.0.0 (release build)
2022-12-11T17:30:17Z Assuming ancestors of block 0000000000004ae2f3896ca8ecd41c460a35bf6184e145d91558ce
ce1c688a76 have valid signatures.
2022-12-11T17:30:17Z Setting nMinimumChainWork=00000000000000000000000000000000000000000000005180c3bd829
0da33a1a
2022-12-11T17:30:17Z Using the 'sse4(1way),sse41(4way),avx2(8way)' SHA256 implementation
2022-12-11T17:30:17Z Using RdRand as an additional entropy source
2022-12-11T17:30:18Z Default data directory C:\Users\Vanesa\AppData\Roaming\Bitcoin
2022-12-11T17:30:18Z Using data directory C:\Users\Vanesa\AppData\Roaming\Bitcoin\testnet3
2022-12-11T17:30:18Z Config file: C:\Users\Vanesa\AppData\Roaming\Bitcoin\bitcoin.conf
2022-12-11T17:30:18Z Config file arg: rpcallowip="127.0.0.1"
2022-12-11T17:30:18Z Config file arg: rpcpassword=****
2022-12-11T17:30:18Z Config file arg: rpcuser=****
2022-12-11T17:30:18Z Config file arg: server="1"
2022-12-11T17:30:18Z Config file arg: testnet="1"
2022-12-11T17:30:18Z Config file arg: txindex="1"
2022-12-11T17:30:18Z Setting file arg: wallet = ["wallet"]
2022-12-11T17:30:18Z Using at most 125 automatic connections (2048 file descriptors available)
2022-12-11T17:30:18Z Using 16 MiB out of 32/2 requested for signature cache, able to store 524288 eleme
nts
2022-12-11T17:30:18Z Using 16 MiB out of 32/2 requested for script execution cache, able to store 52428
8 elements
2022-12-11T17:30:18Z Script verification uses 3 additional threads
2022-12-11T17:30:18Z scheduler thread start
2022-12-11T17:30:18Z WARNING: option -rpcallowip was specified without -rpcbind; this doesn't usually m
ake sense
2022-12-11T17:30:18Z HTTP: creating work queue of depth 16
2022-12-11T17:30:18Z Config options rpcuser and rpcpassword will soon be deprecated. Locally-run instan
ces may remove rpcuser to use cookie-based auth, or may be replaced with rpcauth. Please see share/rpca
uth for rpcauth auth generation.
2022-12-11T17:30:18Z HTTP: starting 4 worker threads
2022-12-11T17:30:18Z Using wallet directory C:\Users\Vanesa\AppData\Roaming\Bitcoin\testnet3\wallets
2022-12-11T17:30:18Z init message: Verifying wallet(s)…
2022-12-11T17:30:18Z Using BerkeleyDB version Berkeley DB 4.8.30: (April  9, 2010)
2022-12-11T17:30:18Z Using wallet C:\Users\Vanesa\AppData\Roaming\Bitcoin\testnet3\wallets\wallet\walle
t.dat
```

Figura 80: Ejecucion de BitcoinCore.

9.4.3. Instalar Docker Desktop

Para ejecutar Docker en la máquina se ha instalado Docker Desktop[27]. Esta aplicación, además de instalar el servicio que permite ejecutar imágenes Docker, dispone de un entorno gráfico sencillo que permite ver que contenedores se están ejecutando, ver los

logs, acceder a los contenedores. Es una herramienta con licencia, pero que se puede usar en entornos personales.

La instalación de Docker incluye el Docker Compose.

Versiones de:

- Docker: 20.10.14, build a224086
- Docker-compose: 1.29.2, build 5becea4c

Para la instalación se descarga el software oficial y se siguen los pasos de instalación.

9.4.4. Ejecutar contenedores ELK

El entorno final se compone de tres contenedores docker: elasticsearch, kibana y logstash. Para la prueba de concepto, los dos primeros tienen la configuración que necesitan incluida en el docker-compose.yml, ver AnexoIII.

Para Logstash, se han creado dos volúmenes locales que se utilizan para cambiar la configuración que se usa para cargar datos. Por lo tanto, es necesario crear los directorios que se usan en el docker:

- C:/TFM/entorno/logstash-data/pipeline
- C:/TFM/entorno/logstash-data/config.

Por lo tanto, tendremos una carpeta ubicada en C:/TFM/entorno/ que contendrá el docker-compose.yml y las dos carpetas anteriores.

Ahora, desde una consola de comandos ubicada en ese directorio se debe ejecutar

```
>docker-compose up -d
```

Esto levantará los 3 contenedores.

9.4.5. Instalar el plugin de logstash para blockchain

Para realizar la instalación es necesario:

- Descarga el código fuente del plugin logstash-input-blockchain[74] de su página oficial.
- Copia el código del plugin al contenedor, puede hacerse con el comando cp de docker.

```
> docker cp logstash-input-blockchain-master logstash:/tmp
```

- Acceder al contenedore

```
> docker exec -ti logstash bash
```

  - instalar las librerías de Ruby

```
apt-get install rubygems-integration

apt-get install ruby ruby-dev
```

  - ir al directorio /tmp y compilar el código con Ruby

```
gem build logstash-input-blockchain.gemspec
```

  - Instalar el plugin

```
/usr/share/logstash/bin/logstash-plugin install /tmp/logstash-input-blockchain-master/logstash-input-blockchain-0.1.0.gem
```

- Listar los plugins instalados en Logstash para comprobar que hay uno de tipo input para Blockchain.

```
logstash-input-beats
└── logstash-input-elastic_agent (alias)
logstash-input-blockchain
logstash-input-couchdb_changes
logstash-input-dead_letter_queue
```

Figura 81: Plugins de Logstash

9.4.6. Configurar el plugin de blockchain

Crear el fichero para cargar datos de bitcoin y copiarlo en el directorio C:/TFM/entorno/logstash-data/pipeline

```
input {
  blockchain {
    protocol => "bitcoin"
    host => "host.docker.internal"
    port => 18332
    user => "test"
    password => "test"
    granularity => transaction
    interval => 1
  }
}
output {
    elasticsearch {
        hosts => ["elasticsearch:9200"]
        index => "bitcoin-init"
    }
}
```

Figura 82: Fichero de logstash para cargar datos de Bitcoin.

Reinicar el contenedor de logstash

```
>docker-compose restart logstash
```