Plataforma para analizar la red Bitcoin

Jaime Pérez Díaz
Grado de Ingeniería Informática

Cristina Pérez Solà

7 de junio de 2017

A mis sobrinas Noelia y Carolina.

FICHA DEL TRABAJO FINAL

Título del trabajo:	Plataforma para analizar la red Bitcoin
Nombre del autor:	Jaime Pérez Díaz
Nombre del consultor:	Cristina Pérez Solà
Fecha de entrega (mm/aaaa):	06/2017
Área del Trabajo Final:	Seguridad Informática
Titulación:	*Grado de Ingeniería Informática*

Resumen del Trabajo (máximo 250 palabras):

Hoy en día, dentro del mercado financiero tenemos como una opción de moneda electrónica, la moneda bitcoin. Con esta moneda, podemos realizar una gestión de las transacciones de dinero entre personas.

Debido a la arquitectura de los nodos gestores de la moneda bitcoin, basada en una red P2P, se hace complicado el estudio de los comportamientos que pueden tener, diferentes topologías de red de nodos Bitcoin, así como la evaluación de estas redes a lo largo de un periodo de tiempo.

Con este trabajo, conseguimos ofrecer una plataforma con la que poder analizar de forma gráfica, sencilla e intuitiva, diferentes topologías de redes Bitcoin. Estas redes podrán desplegarse en cada uno de los diferentes entornos de ejecución que provee Bitcoin, como puede ser regtest, mainnet y testnet, pudiendo gestionar en ejecución las tres redes a la vez.

En la primera parte del documento se presenta una descripción de la arquitectura del proyecto junto con una descripción de cada una de las tecnologías que se utilizan dentro de la solución. Estos componentes son Docker, Docker Swarm, Bitcoin, Statds, Graphite, Grafana y un portal para la gestión de las redes.

En la segunda parte se presenta la solución propuesta mostrando la arquitectura utilizada, desarrollando las necesidades de la solución, la personalización requerida por parte de los componentes para poder dar soporte a la plataforma. Además se describe el despliegue que la solución requiere. También, se muestra una descripción del interfaz de la solución.

Como tercera parte se realiza una valoración sobre la plataforma presentando que puntos puede aportar la solución para facilitar la comprensión de las topologías de redes Bitcoin.

Abstract (in English, 250 words or less):

Nowadays, within the financial market we have as an electronic currency option, the bitcoin currency. With this currency, we can manage money transactions between people.

Due to the architecture of bitcoin coin-based nodes, based on a P2P network, it becomes complicated to study the behaviors that may have different network topologies of Bitcoin nodes as well as the evaluation of these networks along A period of time.

With this work, we managed to offer a platform with which to analyze graphically, simply and intuitively, different topologies of Bitcoin networks. These networks can be deployed in each of the different execution environments provided by Bitcoin, such as regtest, mainnet and testnet, being able to manage the three networks in execution at the same time.

The first part of the document presents a description of the project architecture together with a description of each of the technologies that are used within the solution. These components are Docker, Docker Swarm, Bitcoin, Statds, Graphite, Grafana and a portal for network management.

The second part presents the proposed solution showing the architecture used, developing the needs of the solution, the customization required by the components to be able to support the platform. It also describes the deployment required by the solution. Also, a description of the solution interface is shown.

As a third part, a valuation is made on the platform presenting which points can provide the solution to facilitate the understanding of the topologies of Bitcoin networks.

Palabras clave (entre 4 y 8):

Bitcoin, blockchain, docker, graphite, grafana, statoshi.

Índice

Lista de figuras

1. Introducción

Este documento presenta la memoria del Trabajo Fin de Grado en el área de Seguridad Informática del Grado de Ingeniería Informática en el cual se realiza un estudio de Plataforma para analizar la red de Bitcoin.

Podemos establecer una diferenciación entre las monedas convencionales y Bitcoin, esto es, que bajo la moneda de Bitcoin no existe ninguna autorizad de expedición centralizada, utilizando la tecnología peer-to-peer para operar. La utilización de una red P2P permite establecer a Bitcoin la descentralización de la moneda, aunque dificulta la monitorización del estado de la red. Este modelo dificulta la evaluación del comportamiento de los modos de la red dependiendo de las circunstancias del momento.

El objetivo de este proyecto es crear una plataforma, que permita desplegar, configurar y monitorizar un conjunto de nodos Bitcoin.

Bitcoin surgió en 2009, sin embargo, hasta hace poco tiempo el sector financiero tradicional no se involucró en la utilización de esta moneda. Detrás de Bitcoin no existe una corporación que la avale, sino una forma de código abierto que ha permitido tener una nueva forma de gestionar, usar, enviar y almacenar el dinero por los consumidores.

Bitcoin se asemeja a los tiempos del principio de la contabilidad y el control del dinero y los bienes a través de los libros de contabilidad. Susan Athey, Profesora de Economía y Tecnología de Stanford GSB, establece una relación entre Bitcoin con una hoja de cálculo [1], siendo Bitcoin una manera más moderna de libro mayor.

Un punto a tener en cuenta es que el libro mayor no se almacena en ninguna computadora, sin embargo, existen copias de él en todo el mundo. Lo que ofrece Bitcoin es permitir que no existe ninguna forma por la cual pueda ser hackeada para cambiar el contenido que existe dentro de la hoja de cálculo.

Bitcoin facilita a los usuarios anexar entradas en la hoja de cálculo para asignar una ganancia o pérdida en los fondos. Esto se consigue a través de la gestión de una clave o contraseña que permita el acceso. Es necesario tener en custodia la clave para autorizar el movimiento de sus Bitcoins a otra persona. Sin embargo, hay que tener en cuenta que el valor de los Bitcoins fluctúa.

En las casas de intercambio de Bitcoin, puedes ver cuantos bitcoins puedes obtener por un Euro. En un corto periodo de tiempo, puedes utilizar un servicio para completar la transacción. En el momento que se

envían los bitcoins, el usuario final consigue intercambiar los bitcoins recibidos rápidamente para recibir la misma cantidad con independencia de cuál sea su moneda.

Nadie conoce cuál será el futuro de Bitcoin, pero se habla de la investigación de los gobiernos para la creación de una moneda digital patrocinadas por algunos gobiernos.

La moneda bitcoin puede haber caído recientemente, pero su utilización se duplicó en 2016 [2]. Respecto 2017 probablemente bitcoin siga liderando el incremento de su utilización.

Como hemos comentado Bitcoin se apoya en una red P2P [3], estas redes se basan en la libre compartición de datos, siendo los usuarios que más datos comparten los que alcanzan mayor status y obtienen mayores privilegios al acceder a otros nodos de la red.

La velocidad de transferencia depende directamente de los nodos a los que se está accediendo, siendo una red totalmente descentralizada, permitiendo que la desconexión de un nodo afecte mínimamente a los nodos.

Este tipo de redes presenta problemas de seguridad, ya que al no tener un control directo sobre el contenido, la información puede tener riesgos de seguridad.

1.1 Descripción del problema y motivación.

La red de Bitcoin se basa en el despliegue de una red Peer-to-Peer, los cuales, tienen diferentes roles (minería, retransmisión, cartera, etc). Esta red está basada en una tecnología opensource. Al apoyarse en una red P2P y ser una red distribuida, los nodos no tienen completa visibilidad sobre el conjunto de la red que componen. Aunque los nodos carecen de esta visibilidad sobre toda la red, si tienen la capacidad de descubrir otros nodos a los cuales poderse conectar.

Como hemos comentado, Bitcoin se basa en una red P2P y en un mecanismo broadcasts para el envío y recepción de las transacciones y bloques a través del protocolo TCP. Para la ejecución de los mensajes broadcast en un entorno controlado, los nodos utilizan una lista interna para enviar los paquetes a nodos seguros, la cual se modifica dinámicamente para obtener direcciones de nuevos nodos anexados a la red P2P.

El problema que se plantea en este tipo de redes, es que no existe un control sobre los nodos, ni se conoce la situación en que, se encuentran en cada momento. La monitorización de una red de este tipo se complica al no conocerse el estado de los nodos, ni cuando están conectados o cuando están dando servicio.

Una de las claves del problema de las redes peer-to-peer es la localización de los datos a través de diferentes nodos y el subsecuente aprovisionamiento para acceder a ellos y la manera de gestionar la carga de trabajo y asegurar la disposición sin añadir sobrecargas [4].

Los puntos importantes a abordar dentro de este trabajo a través de la monitorización son:

- El seguimiento tanto de la actividad de los nodos respecto a la gestión de las transacciones como a factores técnicos, como son:
  - El ancho de banda provisto.
  - Latencia peer-to-peer.
  - Distribución del poder de minado.
  - Utilización de memoria.
  - Envío y recepción de mensajes.
  - Métricas del sistema.
  - Transacciones.
- También, se permitirá generar redes conectadas a la red de Bitcoin o a redes standalone, a través de un interfaz basado en grafos.

1.2 Objetivos y resultados esperados

Se han identificado en la descripción del trabajo tres objetivos principales dentro del alcance del proyecto: Introducción de la red Bitcoin, plataforma de despliegue de nodos en un entorno local y plataforma de despliegue en máquinas remotas.

Código	Descripción
OP1	Introducción de la red Bitcoin
OP2	plataforma de despliegue de nodos en un entorno local
OP3	plataforma de despliegue en máquinas remotas

Para la consecución de estos tres objetivos principales, nos hemos marcado los siguientes objetivos:

Código		Descripción
OP1	Introducción de la red Bitcoin	
	O1-1	Introducción a la red Bitcoin
OP2	Plataforma de despliegue de nodos en un entorno local	
	O2-1	Selección e integración de la plataforma de despliegue en la arquitectura de la solución descrita.
	O2-2	Selección e integración de la solución de métricas de nodo en la arquitectura de la solución descrita.
	O2-2	Selección e integración de la solución de métricas globales en la arquitectura de la solución descrita.
	O2-3	Selección e integración de la solución para la gestión de grafos en la arquitectura de la solución descrita.
	O2-4	Despliegue de una red standalone de Bitcoin
	O2-5	Despliegue de una red integrada con la red Bitcoin
OP3	Plataforma de despliegue en máquinas remotas	
	O3-1	Selección e integración de la solución para la clusterización y programación de despliegues en la arquitectura de la solución descrita.

Después de fijar de los objetivos del trabajo, podemos establecer los resultados que se obtienen:

Código	Descripción	Entrega
R1	Documentación de la introducción de la red Bitcoin	PAC4
R2	Descripción de la solución global del trabajo	PAC4
R3	Despliegue de una red standalone de Bitcoin	PAC2
R4	Despliegue de un red de nodos sobre la red Bitcoin	PAC3
R5	Métricas de una red standalone de Bitcoin	PAC2
R6	Métricas de una red de nodos sobre la red Bitcoin	PAC3
R7	Memoria final del proyecto	PAC4

1.3 Requisitos Iniciales.

La plataforma que se definirá para poder realizar el seguimiento de la red Bitcoin se realizará bajo tres puntos principales, despliegue automatizado de la red Bitcoin, clusterización y programación sobre un sistema virtual de la red, y monitorización tanto de los nodos, como de la red en sí.

En concreto, la plataforma permitirá especificar la configuración de la red a través de la selección del número de nodos a desplegar, la configuración de los nodos y las conexiones de red entre los diferentes nodos a través de un grafo.

Una vez desplegados y configurados los nodos, la plataforma ofrece un cuadro de mano, a través del cual se tendrá acceso a la visualización de datos con métricas en tiempo real sobre los nodos desplegados y métricas de la red completa.

Esta plataforma permitirá la gestión de una red Bitcoin standalone o conectada a la red Bitcoin a través del despliegue de nodos Bitcoin desplegados según configuración

La capa de la plataforma de automatización nos permite, una automatización del despliegue de componentes Bitcoin, a través una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo.

También, nos permitirá realizar despliegues de la plataforma bajo una arquitectura clusterizada y programación de los despliegues, permitiendo establecer y gestionar cluster de los nodos de la plataforma como un sistema virtual.

1.4 Selección de tecnologías.

Las tecnologías que se utilizan en este proyecto, se basan en soluciones opensource separando en seis los componentes de la arquitectura de la plataforma:

- El primer componente, será el contendor de aplicaciones que nos permitirá desplegar los nodos Bitcoin. Para ello, nos basamos en tecnología docker [5], la cual nos proporciona las herramientas y componentes necesarios.

- Dockers swarm [6], que nos proporciona la clusterización y programación de los despliegues para del proyecto.

- Como plataforma de métricas de los nodos se utiliza la tecnología Statoshi - Realtime Bitcoin Node Stats [7] para las métricas de cada nodo.

- Como plataforma para las métricas globales se utiliza grafana [8].

- Como plataforma para la gestión del despliegue a través de grafos, utilizamos las librerías de javascript sigma.js [9]

- La arquitectura tendrá un portal desarrollado, que nos permita gestionar todos los componentes de la solución.

1.5 Metodología

Para la realización de este trabajo, se han desarrollo dos tipos de metodología, dependiendo de la naturaleza de las tareas a realizar. Por un lado, se ha establecido una metodología sobre las tareas relacionadas con el desarrollo del trabajo y, por otro lado, una metodología sobre la investigación.

1.5.1 Metodología relacionada con el desarrollo.

Para realizar la plataforma de análisis de Bitcoin, se ha establecido la siguiente metodología no formal:

- Seguimiento semanal de las tareas establecidas en la planificación.
- Se establecerá un versionado de los documentos generados, para poder realizar un seguimiento.
- Se realizará un versionado de las plataformas, que se vayan generando para un adecuado control de los avances.
- Utilización de un servidor NAS para guardar tanto la documentación, como la plataforma y los desarrollos de la solución.

1.5.2. Metodología sobre la investigación

Para realizar la investigación sobre la red Bitcoin y de cada uno de los componentes de la arquitectura, se seguirá la siguiente metodología:

- Búsqueda en internet de documentación sobre cada uno de los componentes definidos en la arquitectura del proyecto.
- Estudio de la documentación aportada.

1.6. Planificación.

Nombre	Fecha de inicio	Fecha de fin
Tareas Generales	22/02/17	4/05/17
TP1.1 Estudio red Bitcoin	22/02/17	4/05/17
TP1.2 Desarrollo documentación Bitcoin	22/02/17	4/05/17
HP1 Punto descripción red Bitcoin en memoria	13/03/17	21/03/17
PEC1	22/02/17	12/03/17
TP2.1 Estudio mercado solución de despliegues basados en contenedores	22/02/17	12/03/17
T2.1.1 Revisión Mercado	22/02/17	26/02/17
T2.1.2 Selección de la solución	27/02/17	12/03/17
HP2.1 Selección de la solución	12/03/17	12/03/17
TP2.2 Estudio de solución para métricas en nodos Bitcoins	22/02/17	12/03/17
T2.2.1 Revisión Mercado	22/02/17	26/02/17
T2.2.2 Selección de la solución	27/02/17	12/03/17
HP2.2 Selección de la solución para métricas en nodos Bitcoins	12/03/17	12/03/17
TP2.3 Estudio de solución para métricas globales	22/02/17	12/03/17
T2.3.1 Revisión Mercado	22/02/17	26/02/17
T2.3.2 Selección de la solución	27/02/17	12/03/17
HP2.3 Selección de la solución para métricas globales	12/03/17	12/03/17
TP2.4 Estudio de la solución para la gestión de nodos mediante grafos	22/02/17	12/03/17
T2.4.1 Revisión de Mercado	22/02/17	26/02/17
T2.4.2 Selección de la solución	27/02/17	12/03/17
HP2.4 Selección de la solución para la gestión de nodos mediante grafos	12/03/17	12/03/17
TP2.5 Estudio de la solución para clusterización y programación de desplieg...	22/02/17	12/03/17
T2.5.1 Revisión del Mercado	22/02/17	26/02/17
T2.5.2 Selección de la solución	27/02/17	12/03/17
HP2.5 Selección de la solución para la clusterización y programación de ...	12/03/17	12/03/17
TP2.6 Estudio de la solución de los nodos Bitcoins	22/02/17	12/03/17
TP2.6.1 Estudio de los nodos Bitcoins	22/02/17	12/03/17
HP2.6 Estudio de los nodos Bitcoins	12/03/17	12/03/17
TP2.7 Documentación PEC1	22/02/17	12/03/17
T2.7.1 Establecimiento del problema a resolver	22/02/17	12/03/17
T2.7.2 Establecimiento de objetivos	22/02/17	12/03/17
T2.7.3 Establecimiento de la metodología	22/02/17	12/03/17
T2.7.4 Establecimiento de las tareas a desarrollar y planificación temporal	22/02/17	12/03/17
T2.7.5 Estado del arte	22/02/17	12/03/17
HP2.7 Entrega de la PEC1	12/03/17	12/03/17
PEC2	13/03/17	9/04/17
TP3.1 Despliegue de la solución basada en contenedores (Docker)	13/03/17	23/03/17
T3.1.1 Preparación entorno en maquina virtual	13/03/17	14/03/17
T3.1.2 Desplieigue de la solución basada en contenedores (Docker)	15/03/17	20/03/17
T3.1.3 Configuración de la solución de contenedores (Docker)	20/03/17	23/03/17
TP3.2 Despliegue de un nodo Bitcoin con gestor de métricas	13/03/17	9/04/17
T3.2.1 Preparación del Nodo Bitcoins con gestor de métricas	13/03/17	20/03/17
T3.2.2 Despliegue del nodo Bitcoins con gestor de métricas	20/03/17	23/03/17
T3.2.3 Prueba configuraciones del Nodo Bitcoins	23/03/17	31/03/17
T3.2.4 Despliegue del gestor de métricas	1/04/17	9/04/17
TP3.3 Despliegue de la solución de métricas globales	13/03/17	9/04/17
T.3.3.1 Preparación del Dashboard para las métricas globales	13/03/17	19/03/17
T.3.3.2 Despliegue del Dashboard para las métricas globales	13/03/17	19/03/17
T.3.3.3 Prueba configuraciones de Dashboard para métricas globales	20/03/17	31/03/17
T.3.3.4 Despliegue de la solución global	1/04/17	9/04/17
PEC3	10/04/17	7/06/17
TP4.1 Despliegue de la solución para la gestion de nodos mediante grafos.	10/04/17	7/05/17
T4.1.1 Desarrollo javascript para la gestión de los nodos	10/04/17	20/04/17
T4.1.2 Desarrollo aplicación web para la gestión de los nodos	21/04/17	30/04/17
T4.1.3 Despligue de la solución para la gestión de nodos mediante grafos	1/05/17	7/05/17
TP4.2 Despliegue de la solución para la clusterización y programación de d...	10/04/17	7/06/17
T.4.2.1 Preparación de dockers swarm	10/04/17	15/04/17
T.4.2.2 Despliegue de dockers swarm	16/04/17	30/04/17
T.4.2.3 Prueba configuraciones docker swarm	8/05/17	7/06/17
PEC4	8/05/17	7/06/17
TP5.1 Preparación de la memoria	8/05/17	7/06/17

2. Introducción a las tecnologías.

2.1. Introducción a Docker

Docker es una tecnología de contenedores relativamente nueva [10] y, que ha crecido de forma rápida [11] al permitir crear contenedores, cuya funcionalidad es similar, pero no igual, a la de máquinas virtuales muy ligeras, basándonos en el tiempo de respuesta y uso de recursos, más cercanos a entornos virtuales enjaulados.

Como principio básico podemos establecer, que los contenedores, que nos proporciona Docker, son el camino para tener la misma utilización, que podemos conseguir a través de máquinas virtuales creadas de forma tradicional. A través de la utilización de estos contenedores, Docker permite aislar uno o más procesos, pudiendo especificarles, cual es la memoria, CPU, acceso a red y espacio en disco.

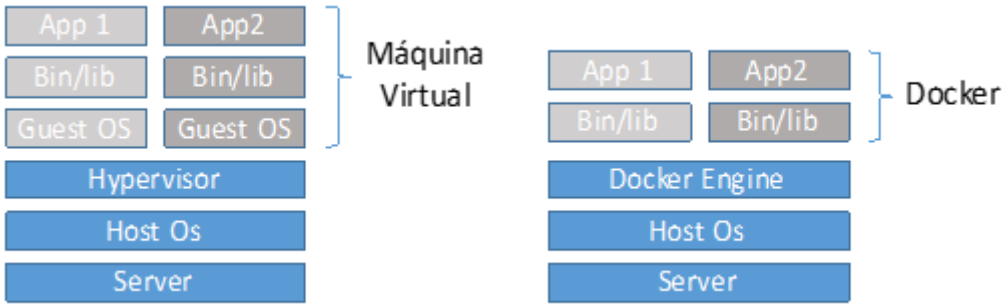

Ilustración 1. Arquitectura Docker

Para definir los componentes que gestiona Docker, podemos decir, que esta solución está compuesta por imágenes y contenedores.

Las imágenes de Docker podemos definirlas, como un sistema operativo, en el que podemos tener el sistema operativo, junto con el software y servicios que necesitamos desplegar. Una característica, que nos proporciona Docker, es que podemos clasificar de imágenes a partir de unos servicios ya instalados y hasta configurados.

Por otro lado, tenemos los contenedores, que se crean a partir de una imagen y son los que nos permiten ejecutar e instalar los servicios que queremos desplegar, como similitud, podemos establecer que un contenedor es una imagen (en tiempo de ejecución) pero bastante más ligero.

Una vez que se despliega el contenedor, a semejanza de las máquinas virtuales tradicionales, está aislado del host. Cuando se han creado los contenedores se establece un id, que lo identifica unívocamente y una etiqueta con un nombre amigable, para facilitar su gestión. Por otra

parte, puede que el contenedor necesite exponer sus servicios hacia el exterior. Para ello, Docker ofrece la posibilidad de exponer puertos del contenedor para, que el Host identifique, cuando queremos entrar en cada puerto, tecnología conocida como port forwarding[12].

Los contenedores están diseñados para la ejecución de aplicaciones, no están pensados para la ejecución de todo lo que, está incluido en una máquina. Sin embargo, un contendor se puede utilizar como una máquina virtual, aunque de esta forma se pierde flexibilidad, ya que, una de las principales características es poder establecer una separación entre la ejecución con el almacenamiento de los datos.

Las principales características de una imagen son: la portabilidad, ya que, tienen la posibilidad de ser versionadas en el repositorio Docker Hub o guardarse como un archivo tar, y contenido estático, ya que, no puede cambiar, si no se genera una nueva imagen.

Las principales características de un contenedor son: el tiempo de ejecución, ya que, cada contenedor se ejecuta en un único proceso, los permisos de escritura, al sólo poder acceder a sus propios archivos o a los volúmenes asociados y, por último, el modelo en capas, siendo una imagen basada en un sistema operativo.

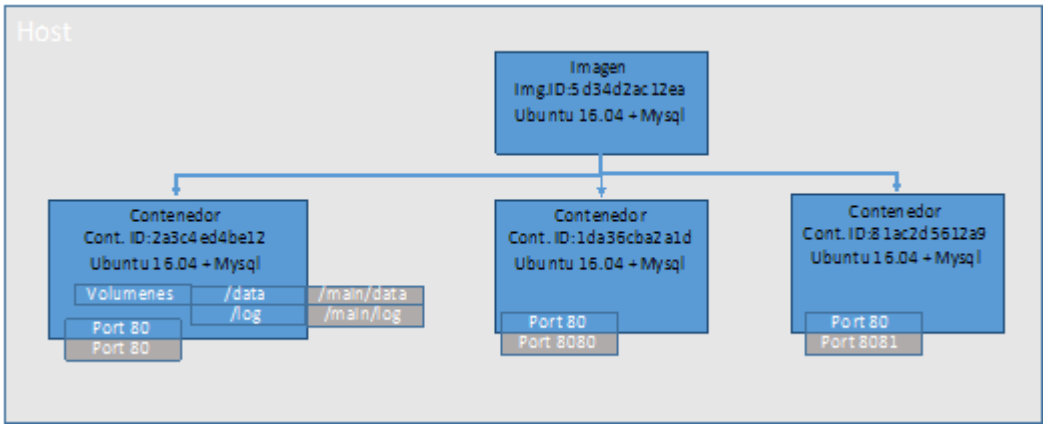

Ilustración 2. Configuración de los contenedores

Por otro lado, Docker tiene la capacidad de mantener los datos, más allá de la vida útil de los contenedores. Son referencias dentro del contenedor a ubicaciones de almacenamiento (Volumenes) fuera del él, permitiendo cambiar, destruir o reconstruir, las veces que necesitemos, los contenedores manteniendo intactos sus datos.

Docker permite establecer una frontera entre las aplicaciones y los datos, permitiendo que los contenedores sean efímeros y desechables. Los volúmenes son específicos de cada contenedor. Se pueden crear varios contenedores partiendo de una sola imagen y asociar el volumen a cada uno.

Las principales ventajas de docker son [13]:

- Instalación sencilla. La instalación de las aplicaciones y todos sus componentes se basa en imágenes y las aplicaciones están contenidas en los contenedores, que se inician desde el gestor de docker.

- Independencia de la plataforma. Las imágenes se pueden volcar en diferentes sistemas, con independencia de la plataforma a utilizar. El único requisito es que el sistema operativo soporte docker.

- Aislamiento de las aplicaciones. Cada aplicación se ejecuta de forma aislada en su contenedor, de manera que si dos aplicaciones tienen requisitos opuestos pueden ejecutarse en paralelo sobre el mismo sistema.

- Control de versiones y reutilización de componentes. Se pueden establecer diferentes versiones de los contenedores.

- Compartición de contenedores. Se pueden utilizar repositorios remotos para compartir los contenedores.

- Administración y automatización. Todos los contenedores son gestionados a través de las mismas herramientas, permitiendo la automatización de las aplicaciones, de forma sencilla y centralizada.

2.2. Introducción a Docker Swarm

Docker Swarm es una herramienta nativa, que permite construir un clúster de máquinas docker. Convierte una agrupación de nodos Docker en un único y virtual Host Docker. Debido a que, Docker Swarm sirve a la API de Docker estándar, cualquier herramienta, que ya comunique con el daemon de Docker puede usar Swarm, para escalar de forma transparente varios Host. Permite tener una gestión y orquestación de cluster incrustado en el Docker Engine.

Como en otros diferentes proyectos Docker, Docker Swarm sigue el principio de “swap, plug, and play”.

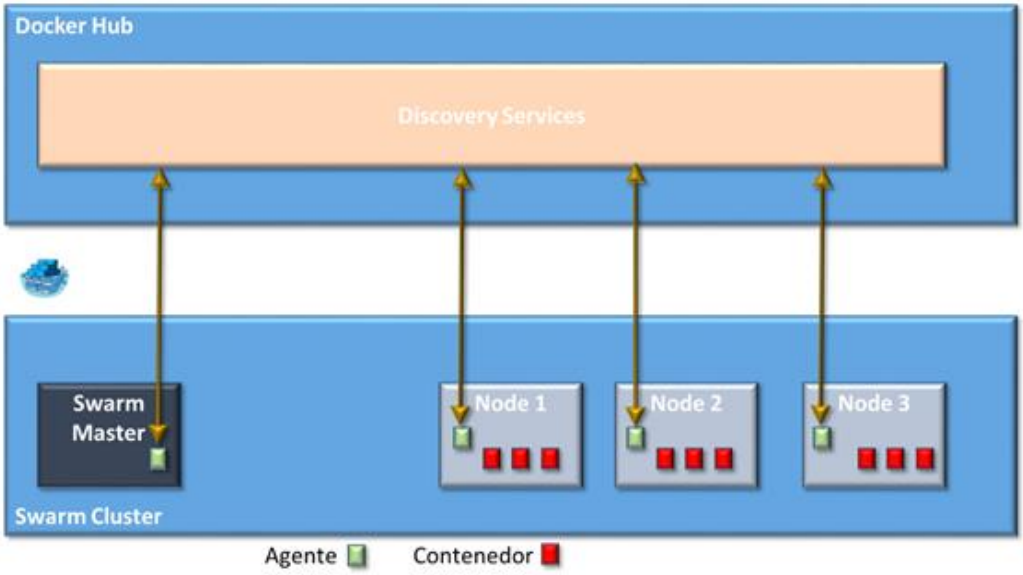

Ilustración 3. Arquitectura Docker Swarm

Las principales características son:

Gestión de cluster integrados con Docker Engine. Utiliza el Docker Engine CLI para crear un enjambre de Docker Engines, donde se puede desplegar servicios de aplicaciones. No se necesita un software de orquestación adicional para crear o gestionar un enjambre de nodos.

Diseño descentralizado. En lugar de gestionar la diferenciación entre los roles de los nodos en tiempo de despliegue, Docker Engine gestiona cualquier especialización en tiempo de ejecución. Puede desplegar ambos tipos de nodos, managers o workers, utilizando Docker Engine. Se puede construir un enjambre completo desde una sola imagen de disco.

Modelo de servicio declarativo. Docker Engine utiliza un enfoque declarativo para, conseguir una definición del estado deseado de varios servicios y la pila de aplicaciones. Por ejemplo, puede describir una

aplicación compuesta por: un servicio web front-end con servicio de colas de mensajes y un backend de base de datos.

Escalado. Para cada servicio, se puede declarar el número de tareas que se quieren ejecutar. Cuando se escala siguiendo el modelo up or down, el gestor de Swarm automáticamente se adapta para añadir o eliminar tareas de mantenimiento del estado deseado.

Reconciliación del estado deseado. El nodo gestor de Swarm constantemente monitoriza el estado del cluster y reconcilia las posibles diferencias entre el estado actual y el estado deseado. Por ejemplo, si se desea configurar un servicio ejecutándose 10 réplicas de un contenedor, y una máquina worker almacena dos de estas replicas corruptas, el gestor creará dos nuevas replicas para remplazar las réplicas corruptas. El gestor de Swarm asigna las nuevas réplicas a los workers, que están ejecutándose y disponibles.

Redes multi-host. Se puede especificar una capa de red para los servicios. El gestor de Swarm, automáticamente, asigna direcciones a los contenedores en la capa de red, cuando se inicializa o actualiza la aplicación.

Servicio de descubrimiento. Los nodos gestores de Swarm asignan cada servicio en el enjambre a un único nombre DNS y realiza un balanceo de carga en los contenedores. Se puede consultar todos los contenedores en ejecución, en el enjambre, a través del servidor DNS incrustado en el swarm.

Balanceo de carga. Se puede exponer los puertos de los servicios a un balanceador de carga externo. Internamente, el enjambre permite especificar como se distribuyen los contenedores de servicios entre los diferentes nodos.

Seguridad por defecto. Cada nodo en el enjambre fuerza, la autenticación mutua TLS y el cifrado de las comunicaciones, seguras entre ellos mismos y el resto de los nodos. Tiene la opción de utilizar certificados autofirmados o certificados de una root CA personalizada.

Actualizaciones progresivas. En tiempo de despliegue, se puede aplicar actualizaciones de servicios en los nodos de forma incremental. El gestor de Swarm permite controlar el retardo entre el despliegue del servicio de los diferentes conjuntos de nodos. Si aparece algún error, puede hacer roll-back de una tarea a una versión anterior.

Los diferentes roles que se presenta en Docker Swarm son:

- ***Swarm Master.*** Es el gestor del cluster y gestiona los recursos de los diferentes nodos Docker.

- ***Cluster Swarm*** es un cluster de nodos Docker Engine, donde se despliegan los servicios. El Docker Engine CLI y API incluyen comandos para gestionar los nodos Swarm, y desplegar y orquestar servicios a través del enjambre.

- ***Nodos Swarm.*** Es cada nodo del cluster, los cuales tienen que tener visibilidad desde el Swarm Master. Cada nodo contiene un agente de modo que registra el daemon Docker referenciado, y además, supervisa y actualiza el servicio de descubrimiento con el estado del nodo.

- ***Swarm Discovery***. Por defecto, Swarm utiliza un servicio de descubrimiento, basado en Docker Hub, a través, de un token para descubrir los nodos que conforman un clúster. También, se soportan diferentes servicios de descubrimiento como pueden ser ETCD, Consul, y Zookeeper.

- ***Swarm Strategy***: Está compuesto por múltiples estrategias de categorización de nodos. Cuando se pone en ejecución un nuevo contenedor, Swarm decide ubicarlo en el nodo con el ranking más alto establecido por la estrategia elegida. Cuenta con múltiples estrategias para la clasificación de los nodos:

    - binpack: nodo con mayor número de contenedores de funcionamiento.

    - spread: nodo con menor número de contenedores de funcionamiento (por defecto).

    - random: aleatorio.

- ***Swarm Networking***: es totalmente compatible para el nuevo modelo de red (overlay) de docker 1.9.

2.3. Introducción a Graphite.

Graphite es un sistema open-source de gráficos en tiempo real, para métricas de series de tiempo. Graphite no recolecta métricas en sí, sino que, al igual que una base de datos, recibe métricas a través su backend las cuales pueden ser consultadas, transformadas y combinadas en tiempo real. Graphite es compatible con interfaces web integrada, que permite a los usuarios consultar métricas y gráficos.

Graphite normalmente se utiliza para monitorizar métricas a nivel de infraestructura, como puede ser consumo de cpu, memoria, utilización de I/O, rendimiento y latencia de la red, aunque desempeña la misma función de la misma manera para métricas de nivel de aplicación y negocio.

Graphite, en sí, no proporciona la capacidad de gestión de alertas, cuando las métricas salen de los rangos esperados. Esta funcionalidad queda cubierta a través, de diferentes soluciones del mercado las cuales proporcionan esta característica.

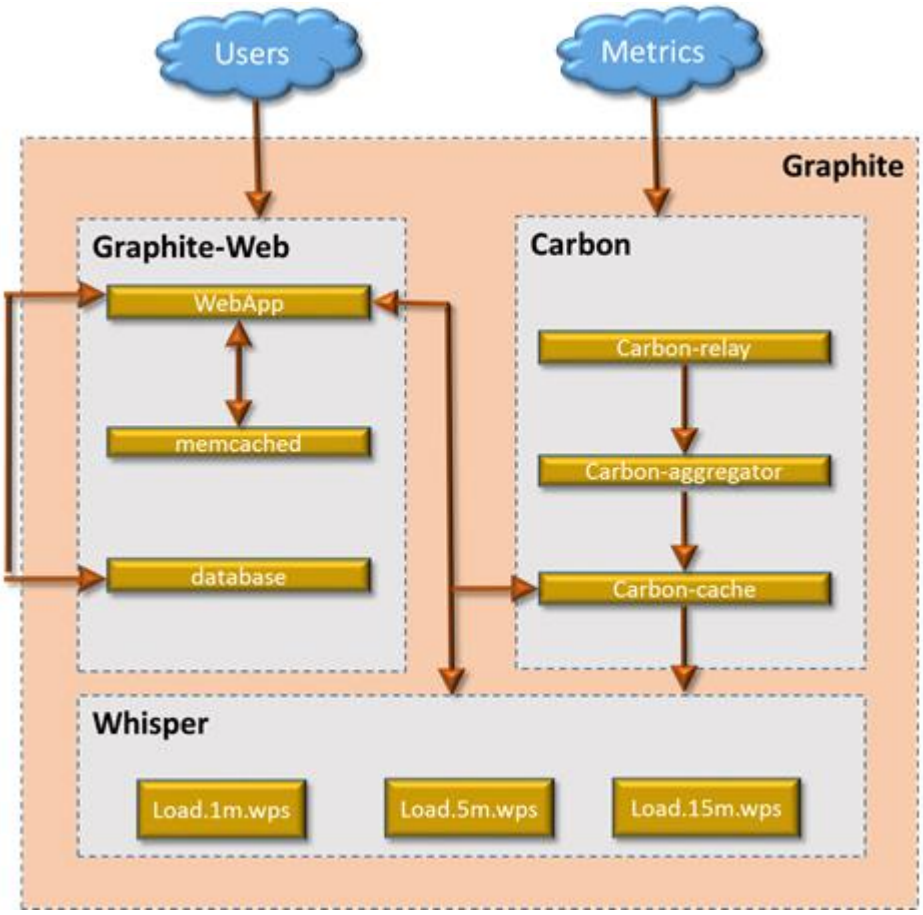

Ilustración 4. Arquitectura Graphite.

Las ventajas que proporciona Graphite respecto a otras soluciones son:

- Es muy rápida. La arquitectura es modular y escalable.
- Está implantada en el mercado, tiene una gran comunidad y tiene mucho soporte.
- Existe una gran cantidad de herramientas open-source que se integran con Graphite.
- Realiza un trabajo sencillo, realizándolo correctamente.
- Esta licenciado bajo Apache 2.0

Como desafíos que se le presentan, podemos enumerar:

- No tiene la posibilidad de compartir los datos, una solución común suele ser el tener múltiples instancias de Graphite.
- La instalación de Graphite puede ser una tarea compleja, aunque hoy en día, existen imágenes completas para instalar en docker, incluyendo todas sus dependencias.

Los componentes de Graphite.

Como hemos expuesto, Graphite es una biblioteca gráfica responsable de almacenar e incorporar representaciones visuales de datos, por lo que, requiere de otras aplicaciones para recopilar y transferir los datos.
Para ello, Graphite está compuesto de diferentes componentes, con un propósito específico y orientado.

- La aplicación web de Graphite.

Aunque no se utiliza dentro de este proyecto, es el componente más visible y dinámico de una instalación de Graphite. Es la aplicación, donde se pueden diseñar los gráficos de los datos.

Graphite proporciona un interface flexible para diseñar gráficos. Permite combinar diferentes tipos de métricas, etiquetas de control, fuentes, colores y propiedades de las líneas, y se puede cambiar el tamaño y manipular los datos a voluntad.

La idea clave es que, Graphite procesa gráficos basados en datos, que recibe y las directrices, que se le dan. No solo imprime gráficos y luego elimina los datos. Permite procesar los datos que se desee, en tiempo real.

La aplicación web, también, permite almacenar las propiedades de los gráficos y los diseños realizados, de modo, que puede promover el interfaz de monitoreo con los ajustes, que se deseen. Puede diseñar, a su elección, varias vistas del panel, esto significa que puede tener un

panel de control independiente para cada máquina o aplicación. Si necesita una correlación de los datos a través de los gráficos, solo se tiene que arrastrar y soltar los gráficos para combinar la pantalla.

- Carbon

Carbon es el backend de almacenamiento para la configuración de Graphite. Un despliegue de Graphite puede tener uno o varios demonios de Carbon, los cuales, son los responsables de gestionar los datos, que son enviados por los procesos, que los recolectan y transmiten sus estadísticas (los colectores no forman parte de Graphite).

Existe diferentes tipos de demonios de Carbon, cada uno responsable de gestionar los datos de una forma diferente. El más básico de todos se denomina carbon-cache.py. Este demonio es simple. Escucha por un puerto para recibir los datos y los escribe en el disco a medida que llegan, de manera eficiente.

Este demonio almacena los datos como vienen y a continuación lo vacía del disco, después de un periodo de tiempo predeterminado. Es importante saber, que el componente de Carbon gestiona los procedimientos de recepción y descarga de datos. No gestiona los mecanismos de almacenamiento reales. Esa responsabilidad se le da al componente whisper descrito más adelante.

El demonio de carbon-cache.py establece en que formato, protocolo y puerto se trabaja. También, se le configura, que políticas de retención de datos utiliza para su almacenamiento de datos. Estos datos se proporcionan a Whisper. Para la mayoría de las configuraciones básicas, con una instancia de carbon-cache.py es suficiente para la gestión de la recepción de datos.

Se pueden ejecutar varias instancias a la vez, a medida que se expanda el despliegue y requisitos. Este servicio puede ser balanceado a través de un demonio carbon-relay.py o carbon-aggregator.py en el frontend.

El demonio carbon-relay.py se puede utilizar para enviar peticiones a todos los demonios del backend con cierta redundancia. Además, se puede utilizar para la fragmentación de los datos en diferentes instancias de carbon-cache.py y para distribuir las cargas de lectura en diferentes ubicaciones de almacenamiento.

El demonio de carbon-aggregator.py puede almacenar datos y después descargarlos en carbon-cache.py, pasado un tiempo. Esto permite ayudar a disminuir el impacto de su procesamiento de estadísticas en el sistema a expensas de los detalles.

- Whisper

Whisper es una biblioteca de base de datos, que Graphite utiliza para almacenar la información recibida.

Whisper es muy flexible y permite almacenar datos de series de tiempo con mucho detalle. Crea diferentes archivos en diferentes niveles de detalle.

Puede almacenar un punto de datos por segundo para una determinada métrica. En Whisper, se puede establecer, que estos datos detallados se debe almacenar con un intervalo de 10 horas. Además, puede tener un fichero que almacena datos de resolución más baja. Solo puede almacenar un punto por minuto y mantenerlo por un periodo de 6 meses.

Cada punto en el fichero de baja resolución se estima a partir de los mismos datos, que se registran en los ficheros de mayor resolución. Puede tener tantos ficheros, como diferentes resoluciones y tasa de retención, como se desee. Puede configurarse como Whisper. Calcula los datos para los ficheros de baja resolución, según el tipo de métrica que se está rastreando.

Una métrica puede ser, el recuento del número de veces que se produce algún evento, en un periodo corto de tiempo. Para crear un nuevo punto para un marco de tiempo más grande con una resolución más baja, se suman los puntos de los datos del fichero de mayor resolución, para resumir los valores de los datos en el periodo de tiempo más largo.

Whisper permite calcular datos de baja resolución de otras formas, dependiendo de la naturaleza de las métricas. Por ejemplo, algunos datos se generalizan haciendo un promedio, mientras que otros pueden estar rastreando un valor máximo. Para el promedio, un valor medio real se establece a partir de los puntos de mayor resolución, para crear el punto de resolución inferior. Para el máximo, el valor más alto debe ser guardado y el resto debe descartarse para mantener el significado del número.

Whisper calcula y registra los datos de resolución más baja en el momento, que recibe los datos (una vez transcurrido el tiempo necesario para recopilar los valores necesarios). Recoge simplemente los puntos de los datos, que necesita para realizar la técnica de agregación de datos (promedio, mayor, menor…) y luego los escribe.

Graphite utiliza el fichero de más alta resolución, que contenga el marco de tiempo solicitado al consultar los datos, para procesar gráficos.

- Protocolos soportados

Existen tres protocolos diferentes, a través, de los que se puede enviar datos a Graphite.

El primero de ellos, es texto plano. Es el formato más flexible, ya que, casi cualquier aplicación o servicio puede generar un envío de texto, y esto se puede utilizar para alimentar Graphite o una herramienta intermedia.

Los mensajes de texto sin formato, incluyen información sobre el nombre de la métrica, el valor que se está proporcionando y la marca de tiempo para ese valor. Estos mensajes se pueden enviar directamente a Carbon, en un puerto establecido para el texto sin formato adicional.

Debido a que Graphite se basa en Python, este también acepta el formato de serialización de datos “pickle”. Este estándar de Python le permite almacenar y enviar múltiples valores de tiempo en una sola transacción.

Graphite, además, puede aceptar datos utilizando mensajes AMQP. Esto le permite gestionar grandes cargas de datos fácilmente. Puede volcar un gran número de estadísticas, y gestionar interrupciones en las conexiones de red entre hosts remotos sin perder datos.

2.4. Introducción a StatsD

StatsD, otro proyecto de código abierto, es un servicio de recopilación de datos de métricas muy popular. Agrupa las muestras que recibe, calcula las sumas, promedios, desviación estándar y algunas otras funciones estadísticas, y periódicamente (por ejemplo, cada 10s) los descarga a una base de datos métrica. Graphite es el backend predeterminado para StatsD.

StatsD es un demonio muy simple, que puede ser utilizado para enviar datos a Graphite. El beneficio de esta aproximación es, que se vuelve trivial desarrollar estadísticas de aplicaciones y sistemas que se encuentren desplegados.

StatsD se ejecuta escuchando en un puerto de un interfaz de paquetes UDP simple, que representa un único punto de datos. Esto significa, que puede aceptar una gran cantidad de información, sin tener conexión. A continuación puede añadir valores recibidos y enviarlos a Graphite.

Este sistema, le permite enviar estadísticas en grandes cantidades, sin preocuparse por aumentar la latencia de la aplicación. El servicio StatsD recopila todos los datos a medida, que se vayan generando, añadiéndolos y, a continuación, enviándolos a los puntos de datos. Graphite resume los datos dentro de una ventana de tiempo.

Debido a estas ventajas que ofrece, es realmente un buen intermediario para la gestión de cualquier tipo de datos enviados a Graphite, siendo la principal funcionalidad el monitoreo de las propias aplicaciones y herramientas que tenemos desplegadas.

StatsD tiene un enfoque ideal como demonio de propósito general, el cual, acepta tráfico UDP. Existe una gran cantidad de bibliotecas cliente desarrolladas en varios lenguajes de programación, que pueden enviar datos directamente a una instancia StatsD. Esto significa que las aplicaciones, que se estén desarrollando pueden enviar fácilmente datos para ser tratados.

2.5. Introducción a Graphana

Grafana proporciona un amigable cuadro de mandos, el cual permite mostrar varias métricas de Graphite, a través del navegador web. Grafana es atractivo, ya que, es simple de configurar y mantener, es fácil de utilizar y muestra las métricas en un Kibana muy amigable como visualizador de métricas. Gracias a esta funcionalidad, la solución tiene un enorme potencial y tiene la suficiente funcionalidad para poder llegar a desplegarse en entornos productivos.

Una característica importante, que ofrece esta solución, es que no se necesita preocuparse de ninguno de los detalles de los escenarios y complejidades, que conlleva su integración con los diferentes escenarios, que se pueden plantear a ejecutarse apoyándose en Graphite.

Grafana ofrece la posibilidad de gestionar alertas y avisos asociados a los gráficos.

Como conceptos básicos podemos establecer:

Data Source. Permite configurar diferentes fuentes de datos de backend para las series de datos de tiempo. Cada Data Source tiene un editor de consultas especifico el cual, esta personalizado con las características y capacidades que expone el Data Source en particular.

Oficialmente se soportan Data Sources: Graphite, InfluxDB, OpenTSDB, Prometheus, Elasticsearch y CloudWatch.

Se pueden combinar datos desde diferentes Data Sources en un único cuadro de mandos. Cada panel tiene sus Data Sources específicos asociados a las Organizaciones.

Organization. Grafana provee múltiples organizaciones para soportar gran variedad de modelos de despliegue, incluyendo la utilización de una única instancia de Grafana para dar, potencialmente, servicio a múltiples organizaciones.

Cada Organización puede tener una o más Data Sources y todos los cuadros de mando tiene su propia Organización.

User. Es una cuenta de usuario en Grafana. Un usuario puede pertenecer a una o varias organizaciones, y puede ser asignado con diferentes niveles de privilegios a través de roles.

Grafana soporta una amplia variedad de mecanismos de autenticación para los usuarios. Estos mecanismos pueden ser servidores SQL externos, servidores LDAP externos o base de datos interna.

Row. Es una división lógica de un cuadro de mando, y se utiliza para agrupar paneles. Las Rows son siempre de un ancho de 12 unidades. Estas unidades, se escalan automáticamente dependiendo de la resolución horizontal del navegador. Se puede tener el control del ancho relativo de los paneles dentro de una fila.

Panel. El panel es la visualización básica, para construir un bloque en Grafana. Cada Panel proporciona un editor de consultas, que permite extraer la perfecta visualización a mostrar en el Panel utilizando el editor de consultas.

Hay una gran variedad de estilos y opciones de formatos, que pueden exponer cada Panel permitiendo crear un cuadro de mando avanzado.

Los paneles pueden utilizar la funcionalidad “drag and drop”.

Query Editor (Editor de consultas). Expone las capacidades de los Data Source y expone la consulta de las métricas que contiene.

Dashboard (Cuadro de mando). Es el lugar donde se presenta toda la información de las métricas. El Dashboard puede configurarse a través de un conjunto de uno o más paneles organizados en una o más filas.

El periodo de tiempo para el Dashboard puede controlarse a través, de Dashboard time picker en el ángulo superior derecho del Dashboard.

3. Diseño de la solución.

El trabajo fin de grado consiste en el desarrollo de una solución que permita el despliegue ágil de un entorno de nodos de Bitcoin, basados en la distribución statoshi. Permite la captura de métricas de estos nodos, apoyándose en el agente Statsd, la herramienta de monitorización graphite y el frontend de estadísticas (dashboard) grafana.

3.1 Arquitectura de la solución.

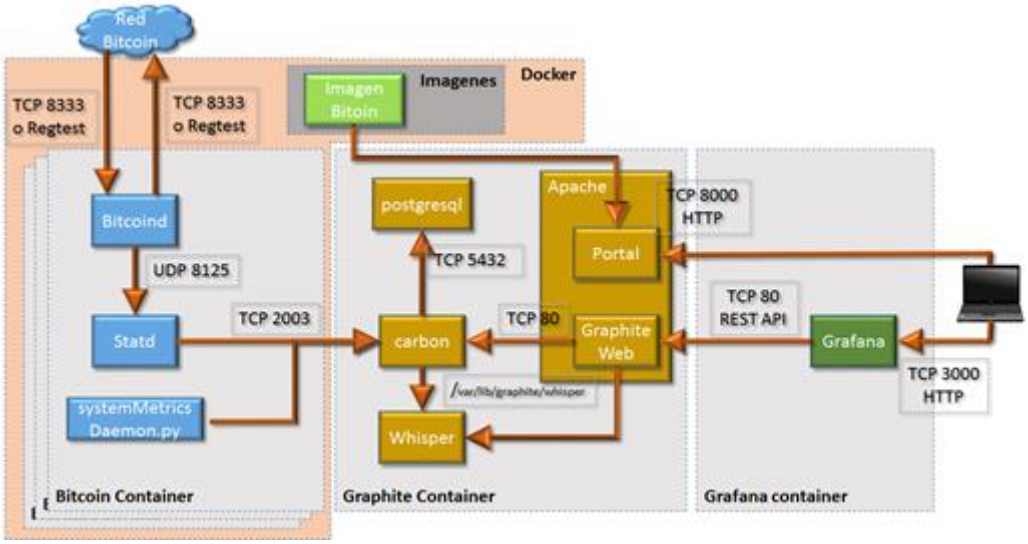

Ilustración 5. Arquitectura de la solución.

La arquitectura, que se plantea para el trabajo fin de grado, consiste en un modelo basado en las capacidades, que proporciona docker respecto a la contenización y gestión de imágenes.

Para el despliegue, se utilizarán los servicios de docker, en donde se creará una imagen basada en su respectivo Dockerfile.

La imagen permite desplegar, los que son los nodos Bitcoin con los agentes recolectores de datos, tanto de Bitcoin como del sistema. Esta imagen permite desplegar el nodo Bitcoin en modo producción, conectándose a la red Bitcoin real, como desplegar los nodos Bitcoin sobre una red de testeo regtest. La imagen contendrá el agente de monitorización Statsd, que tiene la responsabilidad de recolectar los datos de monitorización del nodo Bitcoin, y el demonio systemMetricsdaemon, que nos permite recolectar datos del sistema para tener unas métricas completas de la solución.

Además, de desplegar docker y los nodos Bitcoin desplegaremos dos componentes:

El primer componente será el correspondiente a grahite y todo su entorno. La solución de Graphite está compuesto por los siguientes componentes: Carbon es un demonio Twisted encargado de escuchar datos temporales, Whisper es una librería para almacenamiento de datos temporales, Graphite webapp es una webapp basada en Django que permite la renderización de gráficos bajo demanda utilizando Cairo, postgresql, como base de datos de apoyo para Graphite, y apache que tendrá dos funcionalidades: por un lado, expondrá graphite webapp por en un vhost y, por otro lado, expondrá un portal en otro vhost, el cual tendrá dos funcionalidades principales: por un lado, la publicación de los ficheros de configuración de los contenedores Bitcoin y, por otro lado, alojará el portal de gestión de los nodos de forma gráfica.

El segundo componente permite desplegar Grafana. Grafana es un gestor de dashboard, que permite la renderización de datos en diferentes tipos de gráficas y tablas.

3.2 Gestión de nodos.

Para la gestión de los nodos, se ha planteado la diferenciación de estos, a través de los nombres del contenedor. Se marcaran los contenedores con el nombre establecido a través de un prefijo y el número de nodo para esa red. Los prefijos establecidos son:

- Red regtest: walletreg.
- Red producción: production.
- Red testnet: testnet.

La información sobre, que red pertenece el contenedor se le proporciona, a este, a través de variables de entorno: NODETYPE Y NODENAME. NODETYPE se utiliza en la ejecución del bash para levantar el nodo Bitcoin (nodebit.check.sh). En este bash establecemos las líneas de comandos, que se va a ejecutar teniendo los tres tipos de nodos:

- production: /usr/local/bin/bitcoind -conf=/home/statoshi/.bitcoin/bitcoin.conf
- wallletreg: /usr/local/bin/bitcoind -regtest -nodnsseed -conf=/home/statoshi/.bitcoin/bitcoin.conf
- testnet: /usr/local/bin/bitcoind -tesnet -conf=/home/statoshi/.bitcoin/bitcoin.conf
- defecto: /usr/local/bin/bitcoind -regtest -nodnsseed -conf=/home/statoshi/.bitcoin/bitcoin.conf
- Para los nodos de producción se ejecuta la aplicación bitcoind, únicamente recibiendo como parámetro el fichero de configuración.

Para los nodos de regtest, se ejecuta la aplicación bitcoind con el parámetro del fichero de configuración, el parámetro regtest para indicar que se debe ejecutar en la red regtest y con el parámetro nodnsseed para evitar, que se conecte a nodos que estén fuera de la red que se está desplegando y únicamente se conecte a nodos de la red de prueba.

Para la red testnet se ejecuta la aplicación de bitcoind con el parámetro del fichero de configuración, así como, el parámetro que indica que se ejecuta en el entorno testnet.

Por defecto, se ha establecido que el nodo se ejecute en la red regtest, aunque, este nodo no podría ser gestionado por la plataforma, al no tener la información necesaria para su gestión a través de esta.

La configuración de los nodos está marcada por el fichero de configuración. Los parámetros establecidos para la gestión del nodo vía RPC son: “rpcuser” que contiene el usuario de la conexión al API JSON-RPC, el parámetro “rpcpassword” para la contraseña del usuario de conexión y el parámetro “rpcallowip” para establecer las direcciones IP desde donde se puede acceder al API JSON-RPC.

Para mantener el nodo activo utilizamos una tarea cron, establecida a través del crontab. Este crontab ejecuta periódicamente el bash (nodebit.check.sh), el cual, prepara el comando a ejecutar dependiendo del entorno del contenedor, y ejecuta la aplicación con el comando que se ha generado anteriormente, si no se encuentra el proceso bitcoind ejecutándose.

De forma similar se gestiona el nodo statsd. Ya que, si no aparece el proceso statsd como proceso en ejecución, se crea el fichero de configuración donde, se le establece el prefijo de las estadísticas cogiéndolo de la variable de entorno NODENAME, y a continuación, ejecutando la aplicación statsd.

3.3 Portal de gestión.

La plataforma permite seleccionar, que red de nodos, se quiere gestionar: walletreg (regtest), production (producción) y testnet (testnet).

Ilustración 6. Opciones de entorno de la red.

Para la gestión de los nodos desde la plataforma, se han creado tres tareas: creación de un nodo, arranque de la red y parada de la red.

Ilustración 7. Opciones sobre los nodos

Con el icono de creación de un nodo, la plataforma nos permitirá crear un contenedor de una imagen del nodo statoshi, personalizado para la plataforma, establecidas las variables de entorno NODETYPE, asociado al nombre de la red antes comentadas y NODENAME, con el nombre de la red y el número de nodo secuencial que se va a crear. Una vez creado el contenedor, la plataforma arrancará el nodo y lo dejará operativo.

La ejecución del servicio de creación del contenedor, se activa a través de una llamada Ajax realizada desde el front-end. Dicha llamada espera como respuesta un JSON que informe sobre la dirección IP en la que se ha generado el nodo.

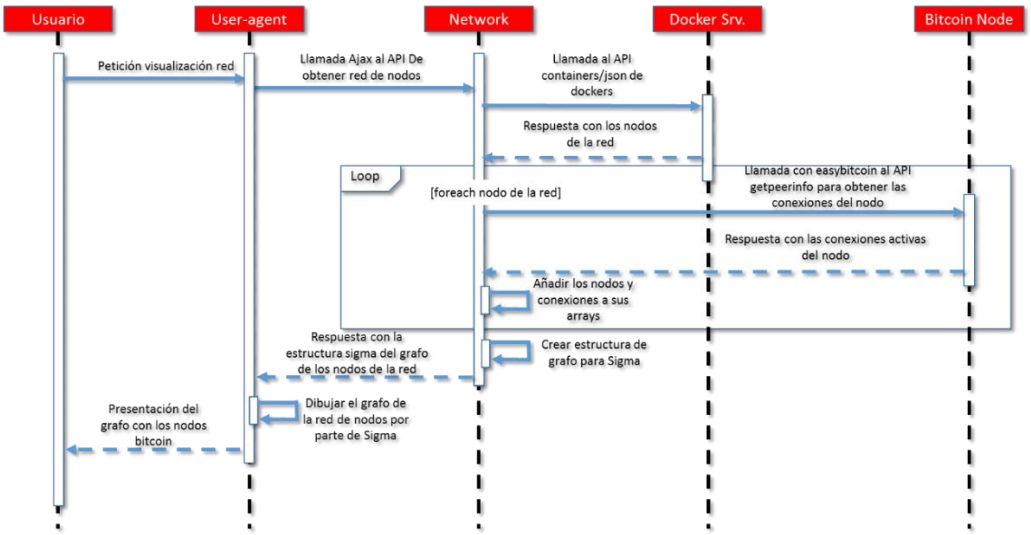

Ilustración 8. Diagrama secuencia gestión de la red.

Para su gestión se utilizarán las siguientes APIS de docker:

- “containers/create”, al que se le pasa como parámetro el nombre del contenedor y en la estructura json la imagen, en la que se va a apoyar y las variables de entorno NODETYPE y NODENAME, generadas de la forma que se ha comentado anteriormente.

- “containers/<nombre del contenedor>/start”, al que se le pasa en la URL el nombre del contenedor que se quiere arrancar. Este API lanza el nodo y lo pone en ejecución.

- “containers/<nombre del contenedor>/json”, al que se le pasa en la URL y se obtiene la información del nodo, en particular la dirección IP que se le ha asociado. Con esta información, se genera el JSON que se va a devolver por la creación y ejecución del nodo.

El servicio de arranque de la red, permite arrancar todos los nodos, o los nodos que estén parados y que pertenezcan a la red para ello se llaman a las siguientes APIs:

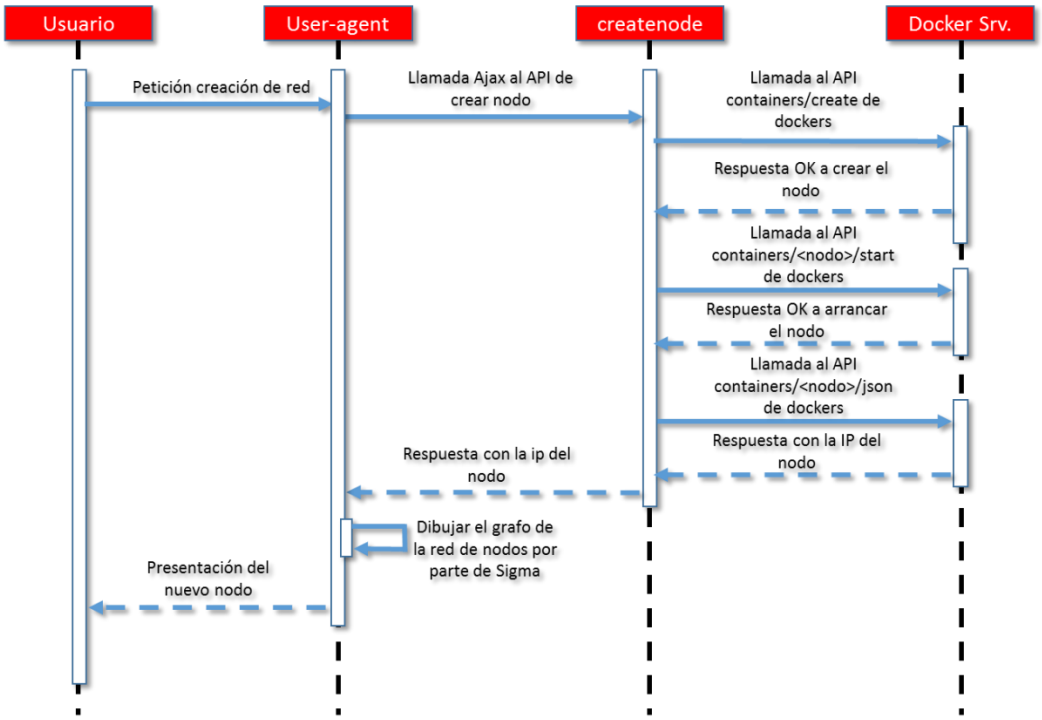

Ilustración 9. Flujo creación de nodo.

- "containers/json". Para obtener todos los nodos que pertenecen a la red seleccionada. Se reciben todos los nodos y se arrancan solo los que pertenecen a la red seleccionada.

- "containers/<nombre del contenedor>/start". Arranca todos los nodos asociados a la red que esta seleccionada.

- El servicio de parada de la red, permite parar todos los nodos, o los nodos que estén arrancados y que pertenezcan a la red, para ello, se llaman a las siguientes APIs:

- "containers/json". Para obtener todos los nodos que pertenecen a la red seleccionada. Se reciben todos los nodos y se arrancan solo los que pertenecen a la red seleccionada.

- "containers/<nombre del contenedor>/stop". Para todos los nodos asociados a la red que está seleccionada.

Tanto el arranque de los nodos de la red como la parada de estos, se realiza a través de una llamada Ajax desde el front-end, el cual devuelve una estructura JSON.

3.3.1. Obtención del modelo de la red.

Para el modelado de la red en el front-end, se ha decidido utilizar el framework de gestión de grafos para javascript "sigma.js". Este

framework nos permite tanto, la creación de grafos a partir de información, referentes a nodos y aristas. Nos permite utilizar el algoritmo de layout forceatlas2 y permite la modificación del grafo y ejecución de servicios, que estén asociados a eventos, que se producen dentro del grafo. Los eventos que se van a gestionar son:

Overnode, nos permite la ejecución de código, cuando el puntero del ratón esté ubicado sobre un nodo. En nuestro caso, nos permitirá la ejecución del servicio que devuelve información sobre el nodo, así como, la visualización de los datos en la caja de presentación de datos.

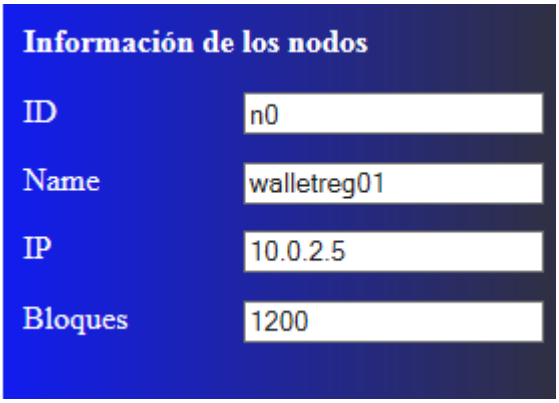

Ilustración 10. Panel Información de los nodos

rightClickNode, nos permite la ejecución de dos lógicas dependiendo de la situación, en que se encuentre. Si se realiza sobre el primer nodo, esté queda marcado para indicar, que será el nodo origen de la conexión entre nodos. Si se realiza sobre el segundo nodo, es para establecer la conexión desde el primer nodo seleccionado a este segundo nodo.

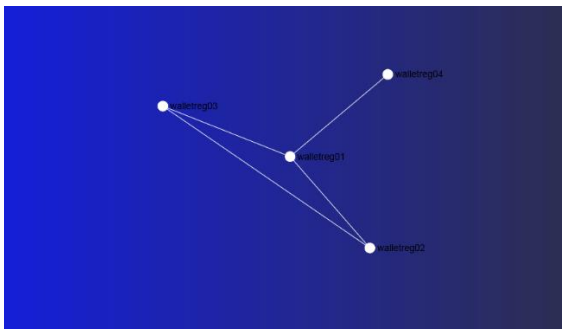

Ilustración 11. Panel Grafo de la red.

Una vez realizada la gestión de los nodos desde el front-end para conocer los nodos que pertenecen a la red, así como sus conexiones, se realiza una llamada al servicio “network”, el cual nos proporciona los nodos asociados a la red y sus conexiones a través de un mensaje JSON con el formato específico para sigma.js.

Para la obtención de los nodos de la red, se realizan dos destacadas operaciones. La primera nos proporciona los contenedores asociados a la red mediante la llamada al API de Docker containers/json. Este nos devuelve todos los contendores, que existen en la instalación Docker. Hacemos un filtro de los nodos que pertenecen a la red solicitada. Y por cada nodo de la red,se consulta: primero lo buscamos en la tabla de nodos y si aparece como nodo externo, lo marca como nodo de la red, y si no aparece se añade a la tabla de nodos como nodo interno.

A continuación, se consulta al API de Bitcoin getpeerinfo el cual nos devuelve todas las conexiones, que tiene establecidas el nodo Bitcoin, y se rellena la tabla con los nuevos nodos que se encuentran en la lista devuelta, marcándolos como nodos externos. También, se añade la conexión a la array de conexiones.

Una vez finalizada la gestión de los nodos de la red y sus conexiones se genera el mensaje JSON con el formato adecuado para sigma.js.

3.4 Visualización de Grafana.

Para la visualización de grafana se ha seleccionado las métricas establecidas en statoshi.info, y se han configurado en la instancia local de grafana. Como muestra se ha seleccionado en el documento, la página inicial:

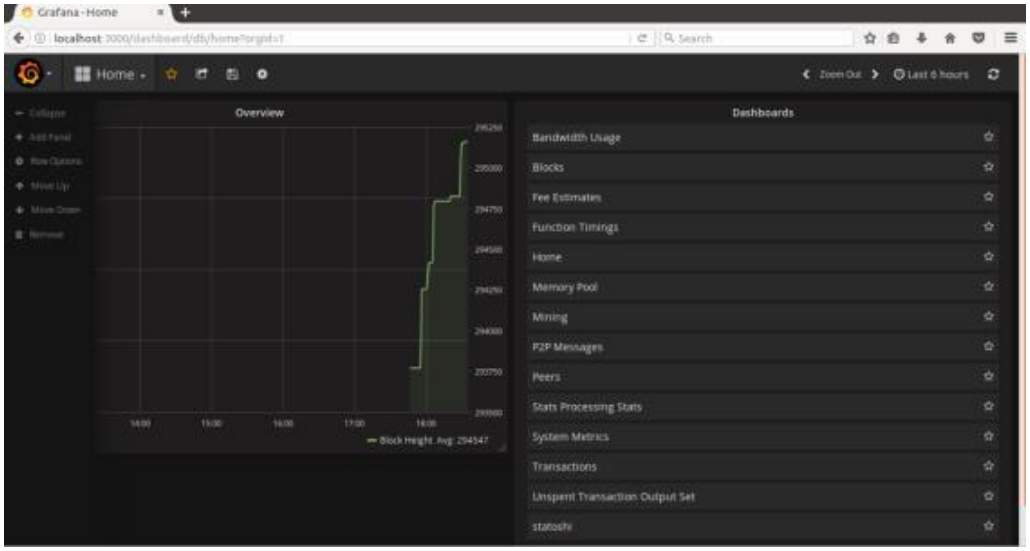

Ilustración 12.Presentación datos en Grafana

La página donde se muestran las métricas sobre los bloques:

Ilustración 13. Presentación datos en Grafana.

Y la página donde se muestran las métricas de los mensajes:

Ilustración 14. Presentación datos en Grafana.

4. Despliegue de la solución.

4.1. Descripción instalación y configuración de docker.

Para la instalación y configuración de docker se seguirán los siguientes pasos:

1. Se instalan las dependencias que requiere docker, como son la selección del repositorio.
2. A continuación, se instala la versión de docker-ce

```
sudo apt-get install docker-ce.
```

Para la ejecución de docker, debido a que nos tendremos que conectar desde otra máquina, establecemos, que para el API se ejecutará escuchando por TCP en la dirección 127.0.0.1 y el puerto por defecto, esto nos permitirá gestionar los contenedores desde el portal.

```
Dockerd –H 127.0.0.1
```

4.2 Descripción de las imágenes.

Se ha partido de la imagen basada en Ubuntu y statoshi-lite, pero con bastantes diferencias de la imagen docker de base.

La imagen de Bitcoin contiene los siguientes pasos:

1. La imagen se basa en una imagen base de Ubuntu. Se ha seleccionado esta imagen del sistema operativo por tener menos dependencias la imagen con respecto a otros sistemas operativos, como puede ser Debian.

2. Se actualiza el sistema operativo antes de empezar la carga de los módulos.

3. Se instalan componentes generales, necesarios para la instalación de los módulos de Bitcoin y statd.

4. Se crea un usuario llamado statoshi y se le asignan permisos de sudo.

5. Se crea la estructura de directorios para el usuario statoshi, y se descargan del portal de la solución desplegada en el entorno de Graphite los ficheros: bitcoin.check.sh, el cual gestiona el estado del demonio de bitcoin, statsd.check.sh, para gestionar el estado del demonio Statsd y systemmetricsd.check.sh para gestionar el estado del demonio systemmetricsd. A todos estos ficheros se les da permisos de ejecución.

6. Se descarga el fichero bitcoin.conf para la configuración del demonio bitcoin.

7. Se asigna la propiedad de los ficheros del directorio statoshi al usuario statoshi.

8. Se descargan las utilidades bitcoin-utils para poder tener en el contenedor el demonio systemmetricsd y se configura para que obtenga datos de la memoria virtual.

9. A continuación, se instalan las dependencias de la solución statoshi, las cuales son: libevent-dev, libdb4.8-dev y libdb.8++-dev para poder gestionar los datos que se almacenan en el wallet.

10. Se descarga la solución de statoshi y se configura con las siguientes opciones: --with-incompatible-bdb, --with-cli --without-gui --enable-hardening –without-miniupnpc, que nos permiten establecer las siguientes características para el demonio bitcoin:

- `--with-incompatible-bdb`, ya que, no tenemos dependencias de compatibilidad de la cartera.

- `--with-cli` instalamos el cliente cli para hacer operaciones sobre bitcoin.
- `--without-gui` no instalamos el entorno gráfico.
- `--enable-hardening`, activamos el hardening de bitocoin.
- `--without-miniupnpc` ya que no se requiere soporte UPnP.

11. A continuación, se instala la solución statoshi, a través del comando make.

12. El siguiente paso consiste en descargar Statsd con todas sus dependencias (node.js, npm y forever. Se instala node.js y npm). Se instala forever a través de npm, y se descarga statsd.

13. Se establece el fichero de configuración de Statsd. Se establece para la configuración, que la dirección IP donde se encuentra instalado Carbon de Graphite en el Host que contiene Docker.

14. Se ejecuta forever para tener activado node.js con stats.js arrancado con el fichero de configuración config.js

15. Para finalizar, configuramos crontab para poder ejecutar con cron los demonios de bitcoin, statsd y systemmetrics.

4.3 Descripción de los contenedores

4.3.1 Entorno de producción.

Para el entorno de producción se desplegarán contenedores basados en las imágenes, en las cuales, tienen desplegados los wallets y tienen mapeados los puertos. Esta configuración, nos permite conectarnos a la red de Bitcoin indicando, porque puerto va a estar publicado el nodo de Bitcoin. Se necesita esta configuración debido, a que docker necesita conocer, que puertos se van a mapear en el contenedor para su consumo desde el exterior.

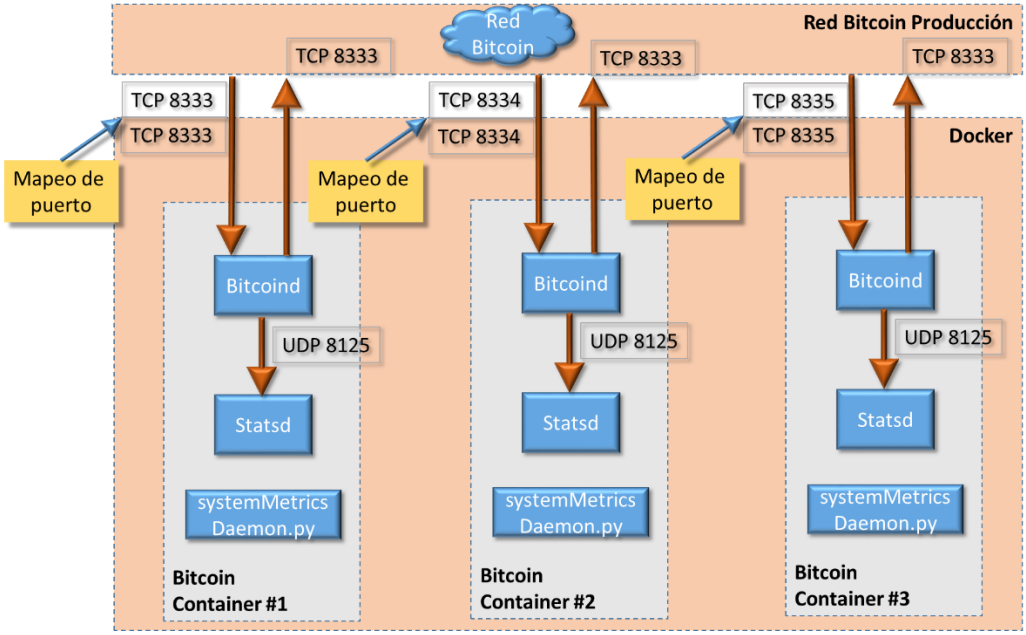

Ilustración 15. Entorno de producción

Docker expondrá diferentes puertos, por los cuales, accederán los nodos externos de la solución. Dentro del contenedor se ejecutarán dos demonios, el Statsd y el systemmetricsdaemon, para la recolección de datos y envío a graphite. La configuración de red de los nodos será del tipo bridge.

4.3.2 Entorno de regtest.

Para el entorno de regtest, se desplegarán contenedores basados en las imágenes de regtest, las cuales, tienen desplegados los wallets y tienen asignadas las direcciones IP estáticas, para poder crear la red Bitcoin, y que se conecte cada nodo del contendor con el resto de los nodos de los contenedores. Dentro del contendores se ejecutarán dos demonios, el Statsd y el systemmetricsdaemon para la recolección de datos y envío a graphite.

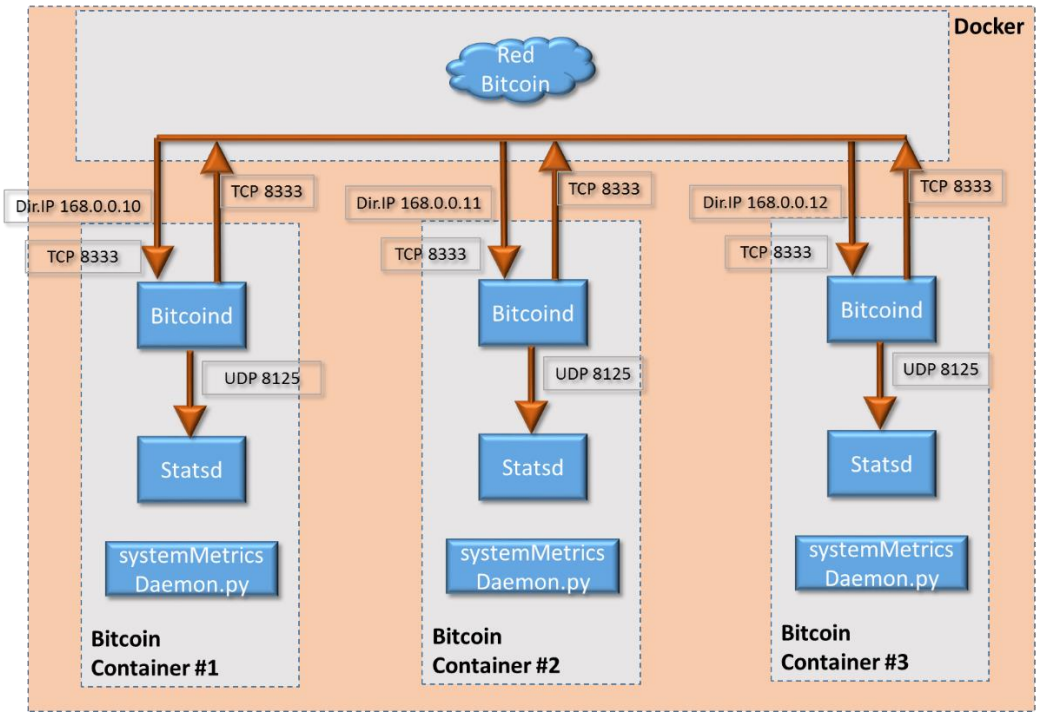

Ilustración 16. Entorno de regtest.

La configuración de red de los nodos será del tipo bridge.

Para esta solución, se creará una red virtual entre los contenedores de los nodos Bitcoin, en la que se podrá simular una red Bitcoin. Al tener desplegada la funcionalidad wallet, podremos crear bloques de la cadena block chain para simular un escenario real. Para conseguir crear cadenas de bloques, nos conectaremos al nodo a través de la opción exec de docker ejecutando bash.

- docker –H 127.0.0.1 exec -it wallet /bin/bash.

Una vez dentro del contenedor podemos ejecutar bitcoind-cli -regtest generate 101. Podremos ver el resultado a través de:

- bitcoin-cli -regtest getbalance
  50.00000000

4.4 Despliegue de Grapithe y Grafana.

A continuación, detallamos la instalación y configuración de estos dos componentes.

Para la instalación de Graphite, debemos instalar los componentes de la solución, que son: Graphite-web, Graphite-carbon y whisper, adicionalmente se instalará postgresql para dar soporte a los requerimientos de base de datos, y apache para funcionar como proxy de graphite-web y escuchar por el puerto 80. También se instalará las librerías libpq-dev y python-psycopg2.

Para la configuración de la base de datos, se ejecutará y se creará un usuario “graphite” y contraseña.

Además, se creará una base de datos llamada “graphite”. Para configurar graphite-web, se modificará el fichero local_settings.py, en el que se modificará el parámetro SECRET_KEY y el parámetro TIME_ZONE.

También, se configurará la base de datos postgresql instalada.

El siguiente paso es sincronizar la base de datos para crear la estructura correcta.

La configuración de graphite-carbon tiene dos pasos de instalación: establecer CARBON_CACHE_ENABLED a true en el fichero graphite-carbon, de esta manera, se indica que el servicio se iniciará en el arranque, y establecer ENABLE_LOGROTATION a true en el fichero carbón.conf para la rotación de los logs.

Se configurarán los esquemas de almacenamiento en el fichero storage-schemas con la siguiente configuración:

Para carbon establecemos el patrón ^carbón\. Con retenciones, `10:2160,60:10080,600:262974.`
Para stats establecemos el patrón, ^stats\\..* y retención `10:2160,60:10080,600:262974.`
Para por defecto establecemos el patrón .* con retenciones, `10:2160,60:10080,600:262974`

La configuración de los métodos de agregación no se modifica.

Para el despliegue de Apache, como proxy, el primer paso es la descarga e instalación del componente junto con el componente wsgi.

Una vez instalado, se desactiva el site por defecto ya que este escucha por el puerto 80. Se copia el fichero apache2-graphite.conf en el

directorio de los sites disponibles. La configuración de este fichero es la siguiente:

Con esta configuración se configura el WSGI, requerido para graphite, así como el alias, para el mapeo del directorio donde se encuentran los estáticos de graphite-web.

Para publicar los ficheros de configuración de las imágenes de statoshi, así como para almacenar el futuro portal de gestión de los nodos, se publicará un virtualhost que escuchará por el puerto 8000.

A continuación, se activa el site apache2-graphite y se recarga el servicio de apache.

Para la instalación y configuración de grafana se descarga a través de wget, el paquete de la versión estable y se instala. A continuación, se arranca el servicio de grafana y se configura para que se ejecute en el arranque del equipo.

- `sudo service grafana-server start`
- `sudo update-rc.d grafana-server defaults`

Para la configuración del datasource, se establece la parametrización en grafana dentro de la página de data source. En esta página estableceremos el nombre, el tipo de data source, en nuestro caso graphite, y la URL donde está ubicado graphite y el tipo de acceso que tiene, en este caso proxy.

6. Manual de usuario

6.1 Creación una red Bitocin de Regtest.

El usuario debe seleccionar en el panel de gestión de nodos el radio button de Regtest.
Una vez seleccionado, aparecerá en el panel del grafo.

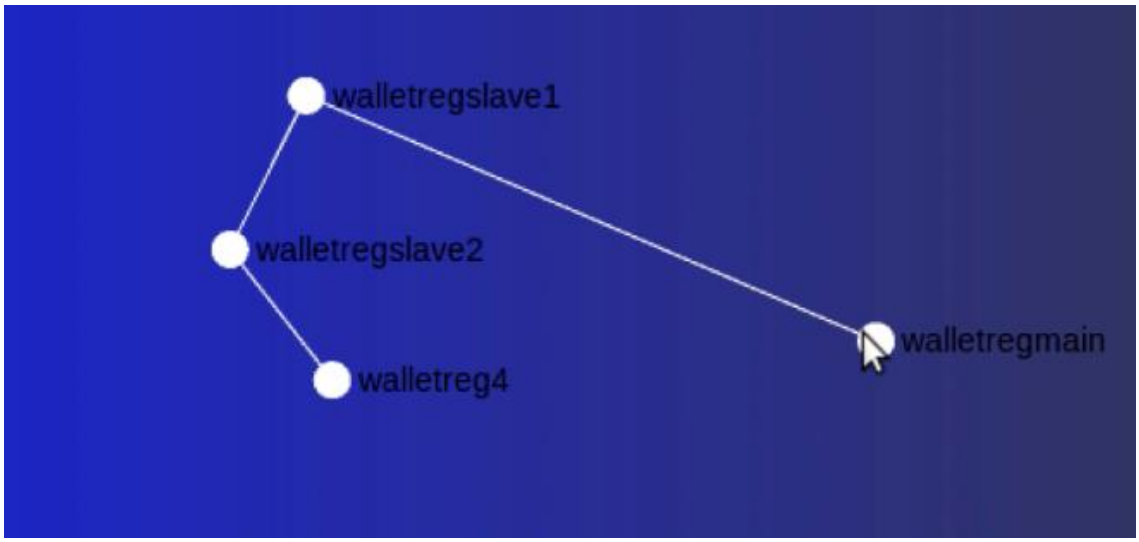

Ilustración 17. Grafo de una red regtest.

Sobre el grafo podremos realizar tres operaciones:

- Visualizar la información del nodo, que aparecerá cuando el puntero del ratón, se encuentre sobre el ratón, mostrando los datos referentes al nodo Bitcoin en el panel de información.

- Mover el nodo en el grafo para ajustarlo a la visualización deseada. Esta acción se realiza pulsando el botón izquierdo del ratón y dejándolo pulsado cuando el puntero del ratón está sobre el nodo, que se desea mover.

- Crear un enlace entre nodos Bitcoins. Para ello, se debe situar el puntero del ratón sobre el nodo en el que, se iniciará la conexión y pulsar el botón derecho del ratón. A continuación, se colocará el puntero del ratón sobre el nodo, que se quiere enlazar y pulsar de nuevo el botón derecho del ratón.

Si deseamos crear un nuevo nodo en la red Bitcoin, debemos pulsar con el ratón sobre el icono de creación de nodos.

Ilustración 18. Opción de Crear nodo.

Nos permitirá crear un nuevo nodo dentro de la red.

También, podemos arrancar todos los nodos que estén parados dentro de la red Bitcoin, a través del icono de arrancar red.

Ilustración 19. Opción de arrancar red.

A la vez, podremos parar todos los nodos, que estén arrancados en la red de Bitcoin a través del icono de parar red.

Ilustración 20. Opción de para red.

6.2. Creación de una red de producción

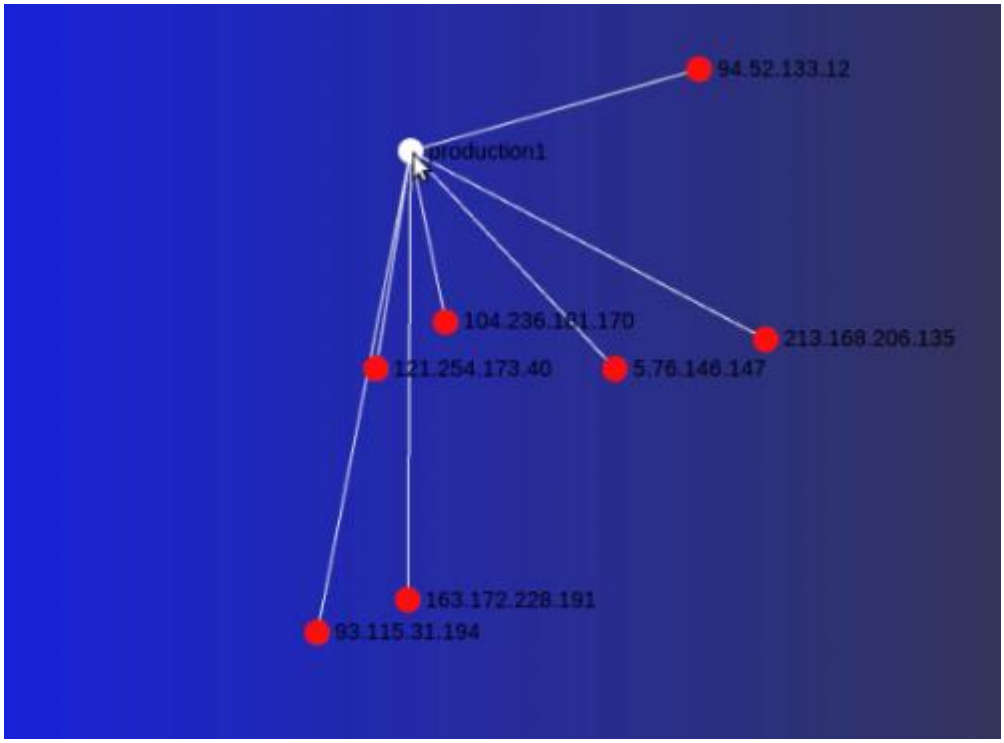

Ilustración 21. Red de mainnet.

La funcionalidad es similar a la red regtest, en cuanto a la gestión de los nodos de la red Bitcoin, exceptuando, que en esta red aparecerán nodos en color rojo, que nos indicará que son nodos externos y, que no se gestionan dentro de la red.

6.3. Estadísticas

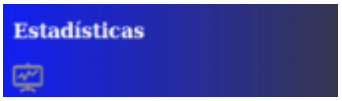

Ilustración 22. Panel de estadísticas

Para visualizar las estadísticas, se debe pulsar el icono en el panel de estadísticas el cual nos proporcionará acceso al panel de Grafana para statoshi.

7. Prueba de concepto y pruebas.

7.1 Prueba de concepto

Para validar el concepto del proyecto, se ha realizado una prueba de concepto de una red mainnet, la cual consta de dos nodos conectados con el exterior, los cuales, están interconectados entre sí. Para la evaluación del comportamiento, se ha registrado la actividad de la red durante 48 horas a través de la solución. Para no desbordar la máquina virtual donde se realizó la prueba, al tener esta una limitación de 64GB de espacio en disco duro, se ejecutó el nodo con la opción prune asignándole un valor de 550.

La red se desplego a través del panel del grafo que presenta el portal. Para la POC crean 2 nodos y los interconectan entre sí. En rojo, aparecen los nodos externos y en blanco los nodos creados.

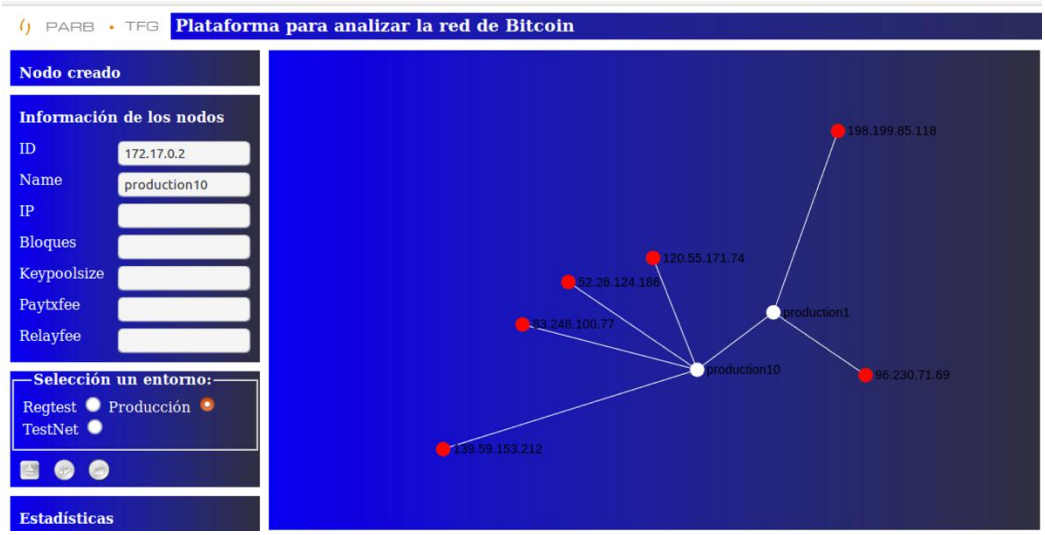

Ilustración 23. Escenario de la POC.

Como resultado del registro realizado, podemos detectar que la gestión de los bloques, se realizó correctamente y el tamaño de los bloques varió entre los 250KBytes y los 1000KBytes, quedando siempre la media equilibrada.

También, podemos detectar como se fueron incrementando el número de bloques recibidos hasta llegar al número de 37400.

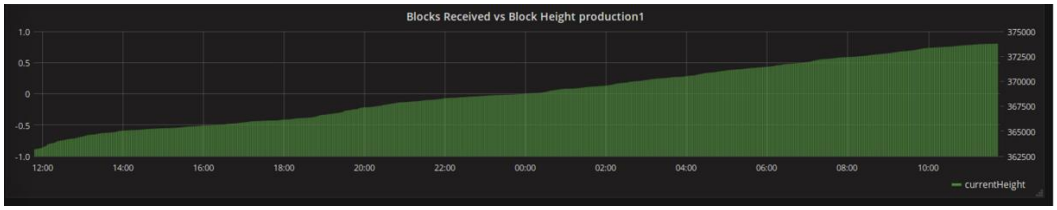

Ilustración 24. Bloques descargados.

A continuación se presenta los datos recolectados referente al tamaño de los bloques, la utilización de los bloques.

Ilustración 25. Gráfica sobre los bloques gestionados.

A continuación se presentan los datos relativos a los procesos, y su utilización del ancho de banda.

Ilustración 26. Ancho de banda por proceso.

A continuación se presenta el total del consumo de ancho de banda respecto al tiempo.

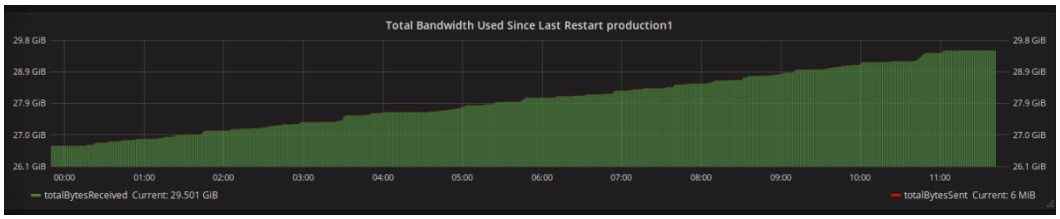

Ilustración 27. Total Ancho de banda consumido.

A continuación se presenta el tipo de conexiones a los peers y de los peers, que se han establecido en el tiempo y los ping realizados sobre los peers.

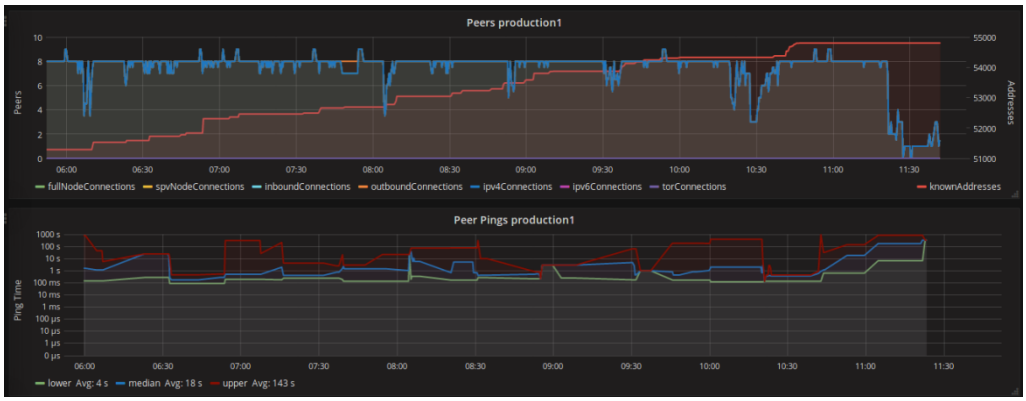

Ilustración 28. Peers conectados.

A continuación se presentan el conjunto de transacciones con el tamaño del cache.

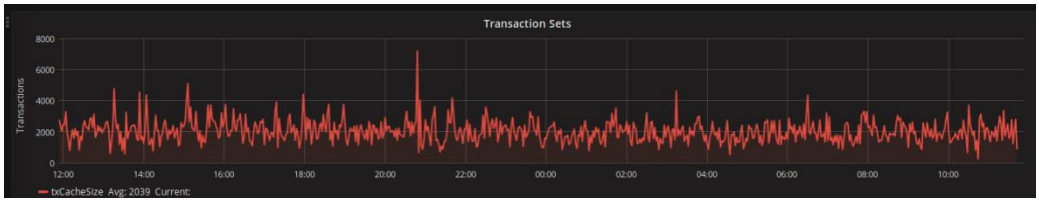

Ilustración 29. Conjunto de transacciones.

A continuación se presentan los datos referente a las entrades y las transacciones.

Ilustración 30. Control de entradas y transacciones.

7.2 Despliegue en Swarm.

Para el despliegue de la solución en un entorno Docker Swarm, se ha probado el desplegar 5 instancias de la imagen Docker de statoshi1 en una arquitectura docker con un master y dos slaves.

Para ver el modelo desplegado, solicitamos la información de los nodos desplegados.

```
sudo docker node ls
ID                            HOSTNAME       STATUS
AVAILABILITY  MANAGER STATUS
0b8pblawz1w24edbpt2gv9o6o     docker-slave1  Ready
Active
xnylhz3wgmjejjndqp739c909 *   Docker-Master  Ready
Active        Leader
xreyr9pihf188cy97mmyudw13     Docker-Slave2  Ready
Active
```

A continuación, creamos el servicio de statoshi.

```
sudo docker service create statoshi1 --containerlabel
production1
oy5cba1hkdsh1f91olfwyxq4y
```

Como siguiente paso, comprobamos que el Servicio esta activo

```
sudo docker service ls
ID            NAME             MODE        REPLICAS
IMAGE
oy5cba1hkdsh  blissful_colden  replicated  0/1
statoshi1
```

A continuación, escalamos el servio a 5 replicas:

```
sudo docker service scale blissful_colden=5
blissful_colden scaled to 5
```

El siguiente paso, comprobamos que se han replicado 5 nodos.

```
sudo docker service ls
ID            NAME             MODE        REPLICAS
IMAGE
oy5cba1hkdsh  blissful_colden  replicated  0/5
statoshi1
```

Para obtenir toda la información del Servicio realizamos una inspección del Servicio desplegado.

```
sudo docker service inspect blissful_colden
[
    {
        "ID": "oy5cba1hkdsh1f91olfwyxq4y",
        "Version": {
            "Index": 135
        },
        "CreatedAt": "2017-06-06T19:58:50.741288371Z",
        "UpdatedAt": "2017-06-06T20:00:44.137550519Z",
        "Spec": {
            "Name": "blissful_colden",
            "TaskTemplate": {
                "ContainerSpec": {
                    "Image": "statoshi1",
                    "Args": [
                        "--containerlabel",
                        "production1"
                    ],
                    "DNSConfig": {}
                },
                "Resources": {
                    "Limits": {},
                    "Reservations": {}
                },
                "RestartPolicy": {
                    "Condition": "any",
                    "MaxAttempts": 0
                },
                "Placement": {},
                "ForceUpdate": 0
            },
            "Mode": {
                "Replicated": {
                    "Replicas": 5
                }
```

```
        },
        "UpdateConfig": {
            "Parallelism": 1,
            "FailureAction": "pause",
            "MaxFailureRatio": 0
        },
        "EndpointSpec": {
            "Mode": "vip"
        }
    },
    "PreviousSpec": {
        "Name": "blissful_colden",
        "TaskTemplate": {
            "ContainerSpec": {
                "Image": "statoshi1",
                "Args": [
                    "--containerlabel",
                    "production1"
                ],
                "DNSConfig": {}
            },
            "Resources": {
                "Limits": {},
                "Reservations": {}
            },
            "RestartPolicy": {
                "Condition": "any",
                "MaxAttempts": 0
            },
            "Placement": {},
            "ForceUpdate": 0
        },
        "Mode": {
            "Replicated": {
                "Replicas": 1
            }
        },
        "UpdateConfig": {
            "Parallelism": 1,
            "FailureAction": "pause",
            "MaxFailureRatio": 0
        },
        "EndpointSpec": {
            "Mode": "vip"
        }
    },
    "Endpoint": {
        "Spec": {}
    },
    "UpdateStatus": {
        "StartedAt": "0001-01-01T00:00:00Z",
```

```
                "CompletedAt": "0001-01-01T00:00:00Z"
            }
        }
]
```

7.3 Pruebas sobre la solución.

Las pruebas realizadas sobre las diferentes redes, se pueden seguir dentro del vídeopresentación del proyecto. En este vídeo, se muestra tanto la generación de una red regtest, como una red mainnet.

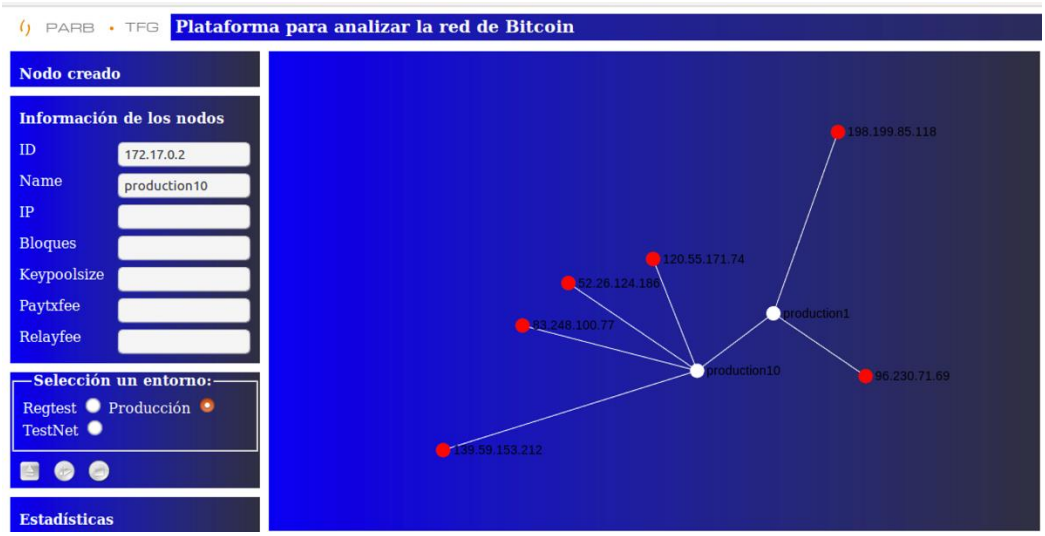

Ilustración 31. Red de producción (mainnet).

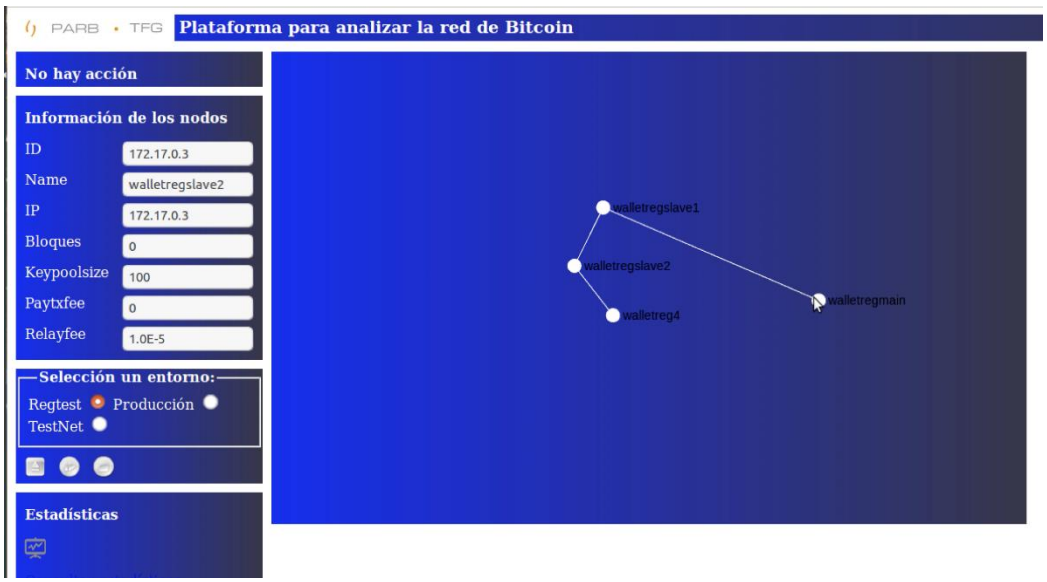

Ilustración 32. Red de regtest.

8. Conclusiones

Durante el proyecto, he podido ampliar el conocimiento sobre la arquitectura Bitcoin, y el modelo Docker de gestión de contenedores, así como los componentes de recolección de estadísticas Statds, Graphite y Grafana. He podido conocer a fondo, como se realiza la gestión de contenedores, así como, la interacción con ellos durante su ejecución. Además, he podido desarrollar mis conocimientos sobre el sistema operativo Linux, al tener que crear imágenes Docker, a través de los ficheros Dockerfile.

También, he podido ampliar el conocimiento respecto a la gestión de grafos, y su implementación a través de la librería sigma.js.

Los objetivos que se han logrado, han estado alineados con la planificación presentada, quedando pendiente la integración de Docker Swarm dentro del portal, principalmente por ajuste temporal, no por limitación técnica. He conseguido crear un portal gráfico, que gestione los nodos bitcoin desplegados en los contendores Docker, he conseguido que el portal muestre gráficamente las conexión, así como la presentación de la información asociada a los nodos. Incluso gestionando los dashboard de grafana relacionados con los nodos Bitcoin.

La planificación ha sido adecuada, excepto con la tarea de la integración de Docker Swarm con el portal. El resto de tareas se ha realizado correctamente, finalizando en la entrega de la solución integral.

Respecto a la metodología, ha sido correcta y la he seguido según lo establecido. No ha habido que incluir cambios. Si bien la tarea de gestión de nodos podía haberse realizado antes, con la idea de obtener antes el conocimiento de gestión de varios nodos en Docker y alinear mejor las tareas previas, si bien, esto se puede analizar una vez finalizado el proyecto.

Como puntos que se pueden extraer de la realización del proyecto, podemos decir que la solución tiene viabilidad, ya que, el producto permite la gestión de nodos y redes bitcoins de forma sencilla y ágil. El usuario no necesita tener conocimiento de los comandos Bitcoins. Además, la solución, nos permite acceder a sus estadísticas de forma rápida.

También, se ha podido gestionar nodos docker a través de su API, punto importante, ya que, para otras soluciones, se ofrece a los usuarios de la solución la posibilidad de gestión de nodos docker, sin requerir tener conocimiento sobre la tecnología.

Los siguientes pasos tienen que seguir las líneas de:

- Integrar Docker Swarm con el portal para poder clusterizar la solución.

- Ampliar la funcionalidad del portal respecto a la gestión de los nodos.

- Dockenización de graphite, grafana y el portal.

4. Glosario

Ancho de banda. Es la cantidad de información o de datos, que se puede enviar a través de una conexión de red en un período de tiempo dado.

Apache. Es un servidor web HTTP de código abierto, para diferentes plataformas, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual.

Bitcoin. Protocolo y red P2P, que sustenta a la moneda bitcoin.

bitcoin. Es una moneda virtual, que fue implantada en el año 1999. Esta moneda no se rige bajo la autorización de bancos o gobiernos.

Broadcast. Es la difusión masiva de información o paquetes de datos a través de redes informáticas.

Cartera. Aplicación virtual que permite guardar las cuentas bitcoin de forma segura.

Contenedor. Es una solución autocontenida en sí, que se puede llevar de un lado a otro de forma independiente, es portable.

Clusterización. Son los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes hardware y software comunes y que se comportan como si fuesen una única computadora.

Debian. Es un sistema operativo gratuito, una de las distribuciones de Linux más populares e influyentes.

Docker. Es un proyecto de código abierto, que automatiza el despliegue de aplicaciones dentro de contenedores de software.

Docker Swarm. Es una herramienta utilizada para agrupar y orquestar contenedores Docker. Con Swarm, los administradores y desarrolladores de TI pueden establecer y administrar un clúster de nodos Docker como un único sistema virtual.

Grafana. Es la aplicación que nos va a permitir visualizar y compartir los datos obtenidos de nuestros sistemas.

Graphite. Es una herramienta de monitoreo, lista para la empresa, que funciona igual correctamente en hardware barato o infraestructura de la nube.

Imagen Docker. Es una plantilla, una captura del estado de un contenedor Docker.

Latencia. Es la suma de retardos temporales dentro de una red. Un retardo es producido por la demora en la propagación y transmisión de paquetes dentro de la red.

Minería. Es el proceso por el cual, se utilizan ordenadores para recibir las solicitudes de transacciones de toda la red y recopilan una lista de las transacciones válidas.

Nodo. Es un ordenador que tiene descargado el software Bitcoin-Qt o Bitcoin Core para participar en la red entre pares.

Opensource. Es el término con el que, se conoce al software distribuido y desarrollado libremente.

Peer-to-peer (P2P). Es una red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí..

Port forwarding. Es la acción de redirigir un puerto de red de un nodo de red a otro. Esta técnica puede permitir, que un usuario externo tenga acceso a un puerto en una dirección IP privada (dentro de una LAN) desde el exterior vía un router con NAT activado.

Protocolo TCP. Es uno de los principales protocolos de la capa de transporte del modelo TCP/IP. En el nivel de aplicación, posibilita la administración de datos, que vienen del nivel más bajo del modelo, o van hacia él.

Sigma. Es una biblioteca de JavaScript dedicada al dibujo de gráficos. Facilita la publicación de redes en páginas Web y permite a los desarrolladores integrar la exploración de red en aplicaciones.

Statoshi. Es un nodo Bitcoin seguro, que permite traer la actividad que ocurre en la red del nodo de forma transparente.

Transacción. Es un trato o convenio por el cual dos partes llegan a un acuerdo comercial, generalmente de compraventa.

Ubuntu. Es un sistema operativo basado en GNU/Linux y que se distribuye como software libre, el cual incluye su propio entorno de escritorio denominado Unity.

Virtualización. Es la creación, a través de software, de una versión virtual de algún recurso tecnológico, como puede ser una plataforma de hardware, un sistema operativo, un dispositivo de almacenamiento u otros recursos de red.

5. Bibliografía

[1] Laura Shin «Susan Athey On How Digital Currency Could Transform Our Lives» [En línea]. Disponible: https://www.forbes.com/sites/laurashin/2014/11/24/susan-athey-on-how-digital-currency-could-transform-our-lives/#958fd432ad14 [último acceso: 11 03 2017].

[2] Bitcoin Price Index. Disponible: http://www.coindesk.com/price/ [último acceso: 11 03 2017].

[3] Wikipedia. Disponible: https://es.wikipedia.org/wiki/Peer-to-peer [último acceso: 05 06 2017].

[4] Couloutis G., Dolimore J., Kindberg K., Blair G. «Distributed System. Concept and Design», Pearson [2012].

[5] Docker.com. Disponible: https://docs.docker.com/engine/docker-overview/ [último acceso: 05 06 2017].

[6] Docker.com. Disponible: https://docs.docker.com/engine/swarm/ [último acceso: 05 06 2017].

[7] github.com. Disponible: https://github.com/etsy/statsd [último acceso: 05 06 2017].

[8] Grafana.com Disponible: https://grafana.com/grafana [último acceso: 05 06 2017].

[9] sigmajs.org. Disponible: http://sigmajs.org/ [último acceso: 05 06 2017].

[10] Darrow, Barb (23 de julio de 2013). «PaaS pioneer dotCloud gets new CEO in industry vet Ben Golub». GigaOM. Disponible: https://gigaom.com/2013/07/23/paas-pioneer-dotcloud-gets-new-ceo-in-industry-vet-ben-golub/ [último acceso: 05 06 2017].

[11] Bala, Raj;Chandrasekaran, Arun. Answering the 10 Biggest Questions About Containers, Microservices and Docker. Disponible: https://www.gartner.com/document/3579056?ref=solrAll&refval=182905820&qid=40db8452ad41e5f4466e7f130532871b [último acceso: 05 06 2017].

[12] Docker. Docker user guide. Disponible: https://docs.docker.com/engine/userguide/networking/default_network/binding/ [último acceso: 05 06 2017].

[13] RedHat. Product documentation Docker. Disponible: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/7.0_Release_Notes/sect-

Red_Hat_Enterprise_Linux-7.0_Release_Notes-Linux_Containers_with_Docker_Format-Advantages_of_Using_Docker.html [último acceso: 05 06 2017].

6. Anexos

6.1 Dockerfile

```
FROM ubuntu:latest
MAINTAINER Jaime Perez Diaz

# docker -H 127.0.0.1 run wallet
/usr/local/bin/bitcoind -regtest &

# Install main tools in Ubuntu

RUN apt-get update
RUN apt-get install -y nano wget
RUN apt-get install -y autoconf autotools-dev build-
essential bsdmainutils git libboost-all-dev libssl-dev
libtool pkg-config
RUN useradd -d /home/statoshi -m -s /bin/bash -c
"Statoshi" -p `openssl passwd -1 statoshi` statoshi
RUN apt-get install -y sudo
RUN echo "statoshi ALL=(ALL) ALL" >> /etc/sudoers
RUN mkdir /home/statoshi/.bitcoin && mkdir
/home/statoshi/.scripts && mkdir /home/statoshi/log
RUN cd /home/statoshi/.scripts && wget
"http://10.0.2.15:8000/conf/nodebit.check" -O
nodebit.check.sh >/dev/null 2>&1 && wget
"http://10.0.2.15:8000/conf/statsd.check" -O
statsd.check.sh >/dev/null 2>&1 && wget
"http://10.0.2.15:8000/conf/systemmetricsd.check" -O
systemmetricsd.check.sh >/dev/null 2>&1 && wget
"http://10.0.2.15:8000/conf/main.check" -O main.sh
>/dev/null 2>&1 && chmod +x *.sh
RUN wget "http://10.0.2.15:8000/conf/bitcoin.conf" -O
/home/statoshi/.bitcoin/bitcoin.conf >/dev/null 2>&1
RUN chown statoshi:statoshi -R /home/statoshi
RUN cd /home/statoshi && git clone
http://github.com/jlopp/bitcoin-utils
RUN sed -i 's/memory          =
psutil.phymem_usage()/memory          =
psutil.virtual_memory()/g' /home/statoshi/bitcoin-
utils/systemMetricsDaemon.py


RUN apt-get install -y libevent-dev
RUN apt-get install -y software-properties-common
RUN add-apt-repository ppa:bitcoin/bitcoin
RUN apt-get update
RUN apt-get install -y libdb4.8-dev libdb4.8++-dev
RUN cd /tmp && git clone
https://github.com/jlopp/statoshi
RUN cd /tmp/statoshi && ./autogen.sh
```

```
RUN cd /tmp/statoshi && ./configure --with-
incompatible-bdb --with-cli --without-gui --enable-
hardening --without-miniupnpc
RUN cd /tmp/statoshi && make
RUN cd /tmp/statoshi && make install


RUN apt-get install -y nodejs npm
RUN npm install forever -g
RUN cd /opt && git clone
https://github.com/etsy/statsd
RUN cp /opt/statsd/exampleConfig.js
/opt/statsd/config.js && sed -i 's/graphiteHost\:
\"graphite.example.com\"/graphiteHost\:
\"10.0.2.15\"/g' /opt/statsd/config.js
RUN ln -s ""$(which nodejs)"" /usr/bin/node
RUN /usr/local/bin/forever start -c /usr/bin/nodejs
/opt/statsd/stats.js /opt/statsd/config.js

RUN wget
"http://yabtcn.info/statoshi/bitcoind.conf.example" -O
/etc/default/bitcoind >/dev/null 2>&1 && wget
"http://yabtcn.info/statoshi/bitcoind.init.example" -O
/etc/init.d/bitcoind >/dev/null 2>&1
RUN chmod +x /etc/init.d/bitcoind
RUN apt-get install -y rcconf sysv-rc-conf cron
RUN update-rc.d bitcoind defaults
RUN update-rc.d cron defaults
RUN rm /etc/crontab && wget
"http://10.0.2.15:8000/conf/crontab.conf" -O
/etc/crontab >/dev/null 2>&1

CMD /home/statoshi/.scripts/main.sh && tail -F
/dev/null
```

6.2 main.sh

```
#!/bin/sh

./home/statoshi/.scripts/statsd.check.sh
./home/statoshi/.scripts/systemmetricsd.check.sh
./home/statoshi/.scripts/nodebit.check.sh
```

6.3 nodebit.check

```
#!/bin/sh

# Plain and simple check-n-start script to keep
Bitcoin Daemon alive
#
# crontab line example:
# */1 * * * *  /path/to/bitcoind.check.sh >/dev/null
2>&1
```

```
# removing the ">/dev/null 2>&1" will cause sending
you mail with info


PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sb
in:/usr/bin
PROGRAM=bitcoind
PATHLOG=/home/statoshi/log
COMMAND="/usr/local/bin/bitcoind -regtest"

case "$NODETYPE" in
"production")
    COMMAND="/usr/local/bin/bitcoind -
conf=/home/statoshi/.bitcoin/bitcoin.conf"
    ;;
"wallletreg")
    COMMAND="/usr/local/bin/bitcoind -regtest -
nodnsseed -conf=/home/statoshi/.bitcoin/bitcoin.conf"
    ;;
"testnet")
    COMMAND="/usr/local/bin/bitcoind -tesnet -
conf=/home/statoshi/.bitcoin/bitcoin.conf"
    ;;
*)
    COMMAND="/usr/local/bin/bitcoind -regtest -
nodnsseed -conf=/home/statoshi/.bitcoin/bitcoin.conf"
    ;;
esac

case "$(pgrep -c bitcoind)" in

0)  echo "Starting $PROGRAM:                  $(date)
$COMMAND" >> $PATHLOG/$PROGRAM.log
    $COMMAND & #>/dev/null 2>&1
    ;;
1)  # all ok
    # echo "$PROGRAM already running:
$(date)" >> $PATHLOG/$PROGRAM.log
    ;;
*)  echo "$PROGRAM is running multiple times: $(date)"
>> $PATHLOG/$PROGRAM.log
    ### kill $(pgrep bitcoind |awk '{print $1}')
    ;;
esac
```

6.4 statsd.check

```
# /bin/bash

# Plain and simple check-n-start script to keep StatsD
alive
```

```
#
# crontab line example:
# */1 * * * *  /path/to/statsd.check.sh >/dev/null
2>&1
# removing the ">/dev/null 2>&1" will cause sending
you mail with info


PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sb
in:/usr/bin
PROGRAM=statsd
PATHLOG=/home/statoshi/log
COMMAND="/usr/local/bin/forever start
/opt/statsd/stats.js /opt/statsd/config.js"


case "$(pgrep -c statsd)" in

0)  echo "{graphitePort: 2003, graphiteHost:
'10.0.2.15', port: 8125, backends: [
'./backends/graphite' ], graphite:{globalPrefix:
'$NODENAME', legacyNamespace: false}}" >
/opt/statsd/config.js
    echo "Starting $PROGRAM:                  $(date)"
>> $PATHLOG/$PROGRAM.log
    $COMMAND >/dev/null 2>&1
    ;;
1)  # all ok
    # echo "$PROGRAM already running:
$(date)" >> $PATHLOG/$PROGRAM.log
    ;;
*)  echo "$PROGRAM is running multiple times: $(date)"
>> $PATHLOG/$PROGRAM.log
    ### kill $(pgrep statsd |awk '{print $1}')
    ;;
esac
```

6.5 systemmetrics.check

```
# /bin/bash

# Plain and simple check-n-start script to keep
systemMetricsDaemon alive
#
# crontab line example:
# */1 * * * *  /path/to/systemmetricsd.check.sh
>/dev/null 2>&1
# removing the ">/dev/null 2>&1" will cause sending
you mail with info
```

```
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sb
in:/usr/bin
PROGRAM=systemmetricsd
PATHLOG=/home/statoshi/log
COMMAND="/usr/bin/python /home/statoshi/bitcoin-
utils/systemMetricsDaemon.py"


case "$(ps aux |grep "systemMetricsDaemon.py" |wc -l)"
in

1)  echo "Starting $PROGRAM:                   $(date)"
>> $PATHLOG/$PROGRAM.log
    $COMMAND & >/dev/null 2>&1
    ;;
2)  # all ok
    # echo "$PROGRAM already running:
$(date)" >> $PATHLOG/$PROGRAM.log
    ;;
*)  echo "$PROGRAM is running multiple times: $(date)"
>> $PATHLOG/$PROGRAM.log
    ### kill $(ps aux |grep "systemMetricsDaemon.py"
|awk 'NR==1{print $2}')
    ;;
esac
```

6.6 local_settings.py

```
DATABASES = {
'default': {
'NAME': 'graphite',
'ENGINE': 'django.db.backends.postgresql_psycopg2',
'USER': 'graphite',
'PASSWORD': 'password',
'HOST': '127.0.0.1',
'PORT': ''
}
}
```

6.7 apache2-graphite.conf

```
<VirtualHost *:80>
WSGIDaemonProcess _graphite processes=5 threads=5
display-name='%{GROUP}' inactivity-timeout=120
user=_graphite group=_graphite
WSGIProcessGroup _graphite
WSGIImportScript /usr/share/graphite-web/graphite.wsgi
process-group=_graphite application-group=%{GLOBAL}
WSGIScriptAlias / /usr/share/graphite-
web/graphite.wsgi
```

```
Alias /content/ /usr/share/graphite-web/static/
<Location "/content/">
SetHandler None
</Location>
ErrorLog ${APACHE_LOG_DIR}/graphite-web_error.log
# Possible values include: debug, info, notice, warn,
error, crit,
# alert, emerg.
LogLevel warn
CustomLog ${APACHE_LOG_DIR}/graphite-web_access.log
combined
</VirtualHost>
```

6.8 portal.conf

```
<VirtualHost *:8000>
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html
ErrorLog ${APACHE_LOG_DIR}/config.error.log
CustomLog ${APACHE_LOG_DIR}/config.access.log combined
</VirtualHost>
```

6.9 storage-schemas.conf

```
[carbon]
pattern = ^carbón\.
Retentions = 10:2160,60:10080,600:262974
[stats]
priority = 110
pattern = ^stats\\..*
retentions = 10:2160,60:10080,600:262974
[default]
pattern = .*
retentions = 10:2160,60:10080,600:262974
```