

IMPLANTACIÓN DE LA TECNOLOGÍA BLOCKCHAIN EN INFRAESTRUCTURA SERVERLESS

Deployment of Blockchain technology in Serverless infrastructure

DANIEL ORTIZ SÁNCHEZ

MÁSTER EN INGENIERÍA INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID



Trabajo Fin Máster en Ingeniería Informática

Curso académico 2018-2019

Convocatoria junio 2019

Calificación: 8 Notable

Director:

José Luis Vázquez-Poletti

Autorización de difusión

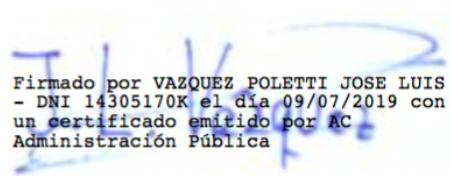
Daniel Ortiz Sánchez

8 de julio de 2019

El/la abajo firmante, matriculado/a en el Máster en Ingeniería en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Implantación de la tecnología Blockchain en infraestructura serverless", realizado durante el curso académico 2018-2019 bajo la dirección de José Luis Vázquez-Poletti en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.



DANIEL ORTIZ SÁNCHEZ



Firmado por VAZQUEZ POLETTI JOSE LUIS
- DNI 14305170K el día 09/07/2019 con
un certificado emitido por AC
Administración Pública

Implantación de la tecnología Blockchain en infraestructura Serverless

El objetivo de este proyecto es el de conseguir desarrollar una solución blockchain externalizando parte de su procesamiento en una plataforma de servicios en la nube.

La solución final consiste en el desarrollo de un banco digital cuyo soporte es una red blockchain propia. Los usuarios podrán conectarse a dicha red mediante un cliente e interactuar con su cuenta del banco. Dentro del cliente los usuarios podrán comprobar el estado de su cuenta, del resto de cuentas existentes en el banco y realizar transferencias a cualquiera de ellas realizando una transacción por la blockchain.

Tanto el flujo del programa del cliente como las operaciones y actualizaciones de la red blockchain y las cuentas del banco se lleva a cabo en el código local de cada cliente. Por otra parte, todo el control de las conexiones entre los distintos usuarios conectados se realiza mediante computación en la nube.

Además, se pone a disposición, mediante un servicio de almacenamiento en la nube y de forma pública, los estados actuales tanto de las cuentas del banco como el histórico de la blockchain directamente para ser consumidas por cualquier aplicación. En este mismo almacenamiento también se guarda el un archivo de log de los movimientos de la última iteración de la blockchain que puede ser usado para su posterior visualización animada.

Palabras clave

1. Blockchain
2. Serverless
3. P2P
4. AWS
5. Banco
6. Criptomoneda
7. Computación
8. Lambda
9. Trazabilidad
10. Seguridad

Deployment of Blockchain technology in Serverless infrastructure

The objective of this project is to develop a blockchain solution by externalising part of its processing to a cloud services platform.

The final solution consists in the development of a digital bank whose support is an own network blockchain. Users will be able to connect to this network through a client and interact with their bank account. Within the client, users will be able to check the status of their account, the rest of existing accounts in the bank and make transfers to any of them by making a transaction through the blockchain.

Both the flow of the client's program and the operations and updates of the blockchain network and bank accounts are carried out in each client's local code. On the other hand, all the control of the connections between the different connected users is carried out through cloud computing.

In addition, it is made available, through a cloud storage service and publicly, the current status of bank accounts and the history of the blockchain directly to be consumed by any application. In this same storage is also saved a log file of the movements of the last iteration of the blockchain that can be used for later animated visualization.

Keywords

1. Blockchain
2. Serverless
3. P2P
4. AWS
5. Bank
6. Cryptocurrency
7. Computing
8. Lambda
9. Traceability
10. Security

Índice general

Índice	I
Agradecimientos	III
1. Introducción	1
2. Objetivos	3
2.1. Objetivos principales	3
2.2. Objetivos secundarios	3
3. Estado del Arte	5
3.1. Blockchain	5
3.1.1. ¿Qué es Blockchain?	5
3.1.2. Características principales	6
3.1.3. Tecnologías Blockchain principales	7
3.1.4. Base de datos distribuida	8
3.1.5. Sistema descentralizado	8
3.1.6. Robustez de datos	9
3.1.7. Participantes	10
3.1.8. Hashing	10
3.1.9. Hash en la cadena de bloques y Proof of Work	11
3.1.10. Minado	12
3.1.11. Profundizando en Ethereum	14
3.2. Serverless	17
3.2.1. ¿Qué es Serverless?	17
3.2.2. ¿De donde viene? Evolución desde las *aaS	17
4. Descripción de la solución	24
4.1. Tecnologías usadas	24
4.1.1. Golang	24
4.1.2. LibP2P	26
4.1.3. Node.js	27
4.1.4. AWS	28
4.1.5. Gource	31
4.1.6. Postman	32
4.2. Esquema	33
4.3. Amazon Web Services	34

4.3.1. IAM	34
4.3.2. Amazon S3	35
4.3.3. Lambda	40
4.4. Cliente Blockchain	43
4.4.1. Estructura de los datos	43
4.4.2. Funcionamiento interno	45
4.4.3. Menú inicial	47
4.4.4. Menú de usuario	48
4.4.5. Instalación y despliegue	50
4.5. Salidas	53
4.5.1. Postman	53
4.5.2. Gource	55
4.6. Alcance y límites	58
5. Ejemplos y casos de prueba	61
5.1. Servidor local	61
5.1.1. Descripción	61
5.1.2. Ejecución	62
5.2. Red local	64
5.2.1. Descripción	64
5.2.2. Ejecución	65
6. Conclusiones	68
7. Trabajo Futuro	70
8. Introduction	72
9. Conclusions	74
Bibliografía	78
A. Prueba inicial con Ethereum	79
A.1. Blockchain mediante Geth en entorno virtual	79
A.1.1. Descripción	79
A.1.2. Objetivos	79
A.1.3. Requisitos	80
A.1.4. Creación del génesis	81
A.1.5. Creación de los nodos	81
A.1.6. Conexión de los nodos	83
A.1.7. Minado de los nodos	84

Agradecimientos

Me gustaría agradecer a mi familia todo el apoyo (tanto anímico como económico) recibido durante la realización de este máster durante los dos últimos años. Sin ellos esto sería imposible.

También me gustaría agradecer a mi director, José Luis, la libertad de decisión que me ha dado en todo momento y el optimismo con el que ha afrontado todos los problemas que nos hemos ido encontrando. Sin duda ha hecho que la realización de este TFM sea mucho más fácil y llevadera.

Capítulo 1

Introducción

En este TFM se tratan dos tecnologías novedosas y que, a priori, no deberían porque tener nada en común. Por un lado, tenemos la tecnología blockchain. En unos tiempos en los que la corrupción y la transparencia son dos temas que están todos los días presentes estudiar una tecnología que los trata de lleno se vuelve muy estimulante. Por otra parte, el recorrido laboral del autor le ha llevado a trabajar construyendo aplicaciones web, arquitecturas y microservicios. Animado por el director de este trabajo cuanto más descubría sobre plataformas en la nube y posibilidades como las funciones lambda más cree que el futuro es ir hacia esa dirección. Quedaba comprobar si ambas tecnologías se pueden combinar y beneficiarse.

El objetivo de este proyecto es comprobar si de verdad se puede abstraer funcionalidad de la tecnología blockchain usando la computación sin servidor haciéndola más ligera, pero manteniendo todos los beneficios intrínsecos de su propia tecnología. La realización de la solución ha supuesto el estudio a fondo de ambas tecnologías y multitud de pruebas y casos de prueba hasta dar con una solución satisfactoria. Para poder demostrar como funcionan ambas tecnologías el proyecto se ha basado en un ejemplo de aplicación bancaria con transacciones y cuentas simulando a otras plataformas blockchain existentes como Bitcoin o Ethereum. El ejemplo siempre debe servir como un vehículo para conseguir demostrar de manera más sencilla como se pueden combinar ambas tecnologías.

La solución propuesta se debe entender como un punto de partida para avanzar en esta dirección y como prueba de concepto de cómo estas dos tecnologías pueden funcionar conjuntamente, más adelante se explicara cuáles han sido los límites y el porqué de abstraer unas determinadas funcionalidades.

Como se puede comprobar en la bibliografía existen multitud de alusiones a ambas tecnologías por separado, pero cuando se juntan ambos términos se han tenido más dificultades a la hora de encontrar referencias.

En el tercer capítulo se puede ver un marco teórico del estado del arte de las tecnologías. Se realiza un estudio de los conceptos fundamentales de blockchain explicando su funcionamiento y sus principales plataformas. También se detalla que significa la tecnología serverless, sus diferentes formas y se proporcionan ejemplos de cada una de sus variantes. Una vez comprendidos todos los puntos se pasa a explicar la solución en el cuarto capítulo. Se comienza mostrando la arquitectura global (la cual aquí cobra gran importancia), los detalles de sus partes y sus conexiones. Se analizan de forma individual detallando las tecnologías usadas y las razones que han llevado a elegirlas. Se continua con los límites y el alcance de la solución, así como los resultados esperados tras probarla.

En el quinto capítulo se muestran los casos de prueba llevados a cabo y si han sido satisfactorios o no.

Y se finaliza con los capítulos sexto y séptimo comprobando si se han cumplidos los objetivos propuestos, los problemas encontrados, las conclusiones obtenidas y el trabajo futuro a realizar.

Capítulo 2

Objetivos

2.1. Objetivos principales

Como se puede deducir por el apartado anterior el objetivo principal de este TFM es el de combinar las tecnologías blockchain y serverless manteniendo las virtudes de ambas con el fin de demostrar todo su potencial usadas conjuntamente.

A su vez también se tiene que conseguir desarrollar un ejemplo real con el que sea fácil de demostrar el despliegue completo.

Antes de nada se tendrá que ganar un conocimiento profundo de ambas tecnologías. Esto será necesario para en el caso de blockchain tomar la decisión de elegir una tecnología existente o desarrollar una desde cero. Respecto a la serverless las decisiones importantes serán qué plataformas se usan y qué funcionalidades se pueden abstraer.

2.2. Objetivos secundarios

A parte de los objetivos principales existen otros que hay que cumplir para satisfacer completamente el proyecto y que son necesarios para cumplir al cien por cien los objetivos principales:

1. Mantenerse dentro de la capa gratuita de los servicios serverless. Esto es importante ya que ayuda a demostrar que los objetivos se cumplen para la plataforma escogida y sería fácilmente traspasable a otra.

2. Comprobar las ventajas de despliegues de blockchain de forma tradicional frente a hacerlo con serverless. Es necesario verificar si de verdad el despliegue conjunto que propone el proyecto aporta beneficios.
3. Conseguir mostrar de forma visual el funcionamiento interno y movimientos de blockchain. De forma teórica se puede llegar a entender lo que realmente hace la tecnología blockchain, pero conseguir mostrarlo a nivel de usuario es importante para demostrar con más convencimiento sus bondades.

Según se vaya recorriendo el documento se verá como se va tratando cada objetivo en profundidad y en el apartado de conclusiones se podrá comprobar si se han cumplido del todo.

Capítulo 3

Estado del Arte

3.1. Blockchain

A continuación se explicará en que consiste la tecnología blockchain, sus características y beneficios y sus tecnologías principales.

3.1.1. ¿Qué es Blockchain?

¿De dónde viene?

La tecnología blockchain surgió en el año 2008 a través de un manifiesto firmado por Satoshi Nakamoto, donde este nombre es un pseudónimo.

Este artículo sirve para dar a conocer al Bitcoin y explicarlo como un sistema de efectivo electrónico en redes p2p (más adelante veremos en qué consiste una red p2p). El texto detalla cómo usar una red p2p de ordenadores para crear una red de transacciones digitales. Esto ya es por tanto lo que conocemos hoy como la cadena de bloques o blockchain.

¿Qué es exactamente?

Básicamente es un libro mayor de transacciones conocido como Ledger que puede registrar movimientos no solo financieros si no que representen cualquier tipo de valor. En este Ledger se puede escribir nueva información, pero la información existente una vez almacenada en sus bloques no puede ser eliminada o editada jamás.

Se puede decir que blockchain es un libro de registros perfectos cuya información es infalsificable. Por tanto, lo que conocemos como la cadena de bloques representa una base de datos distribuida y segura que almacena transacciones y las agrupa en bloques. Cada bloque de la cadena está relacionado con el anterior utilizando procesos criptográficos y contiene el hash de su contenido. Gracias a esta relación entre bloques se verifica que la información de los bloques es consistente y no ha sido modificada. Si un actor malintencionado tratara de cambiar el contenido de un bloque su hash cambiaría y su relación con todos los bloques siguientes sería inválida. Por tanto, un bloque almacena un número de transacciones y los bloques están relacionados entre sí de tal forma que sean inmodificables.

3.1.2. Características principales

Es una red p2p

En las redes p2p un usuario utiliza y compone la red al mismo tiempo. Cada nodo es considerado igual antes los demás pudiendo cambiar en todo caso el rol que desempeña. En este tipo de redes no hay un nodo central de almacenamientos si no que esta se comparte constantemente entre sus participantes. Un ejemplo de red p2p sería el protocolo BitTorrent.

Es distribuido

Esto significa que los datos no se encuentran centralizados en un punto, sino que todos los participantes de la red tienen una copia exacta de los mismos. Esto asegura que los datos no pueden ser manipulados ya que existen multitud de copias que pueden repudiar la información falseada y que la red nunca pueda dejar de ser accesible ante un ataque o desastre informático.

Es descentralizado

No existe un nodo central que se encargue de manejar y llevar a cabo las operaciones de la red. Las transacciones son recogidas por todos los nodos de la red y cualquier cambio tiene

que ser confirmados por la inmensa mayoría de usuarios para ser reconocido como legítimo. Al mismo tiempo consigue que corromper los datos sea prácticamente imposible.

Inmutabilidad de los datos

En una base de datos tradicional se puede modificar cualquier dato del pasado o incluso borrarla. En blockchain una vez los datos han sido escritos nadie puede cambiarlos o reescribir su historia. Esto tienen unos beneficios claros de auditoría.

3.1.3. Tecnologías Blockchain principales

Bitcoin

Primera cripto-moneda virtual que nació teóricamente junto con el manifiesto del blockchain creado por Satoshi Nakamoto. Aparte de lo mencionado en el apartado inicial del Estado del arte es importante resaltar que al ser la primera tecnología que usa blockchain todas las posteriores parten de ella¹⁵. Es decir, muchas de sus características son heredadas o retocadas por posteriores tecnologías como las que veremos a continuación. Se presentó al mundo como una alternativa al dinero “real” ya que supone una moneda que no está controlada por ninguna organización o entidad financiera como si están las demás monedas. Pero la realidad es que se ha encontrado con problemas que están frenando las expectativas en ese sentido y que hacen que no termine de funcionar su enfoque práctico⁸. Problemas tales como: tiempos de minado cada vez más superiores, confianza de la sociedad en la moneda, corrupción de mineros para romper la descentralización...

Ethereum

Es una plataforma open-source que proporciona todo lo necesario para poder facilitar la construcción aplicaciones en tecnología blockchain y todo lo que se puede hacer en ellas: como facilitar las transacciones de divisa de las que se habla en Bitcoin³²⁴. Cuenta con su propia moneda (ether) y lo más importante el gran avance que introduce frente a Bitcoin es el concepto de Smart Contracts.

Importante: posteriormente se verá un caso práctico utilizando Ethereum por lo que se explicará más detalladamente.

Hyperledger

Proyecto open-source de la Fundación Linux creado a finales de 2015 - principios de 2016.⁴

El objetivo es dar un paso más en la tecnología blockchain orientando el proyecto hacia el mundo empresarial y las redes privadas. Además, incluye todo lo necesario para realizar smart contracts que por otro punto encajan muy bien con sus objetivos. El proyecto cuenta con Hyperledger Fabric (¹) que es donde se van desarrollando nuevas funcionalidades. Algunas de estas nuevas características son: integración con docker, distintos tipos de contratos, frameworks y SDK (en Node.js), soporte para APIs...

3.1.4. Base de datos distribuida

Cuando hablamos de un sistema de datos centralizados todos los nodos de la red necesitan conectar a un nodo central para acceder a los datos , el problema es que si ese nodo central no puede ser accedido el sistema completo entra en parada. si la información de este nodo central pudiera ser eliminada y no tuviera procesos de copias de seguridad para mitigar los daños los datos podrían perderse para siempre. En un sistema descentralizado tenemos múltiples copias de los datos y si alguno de los nodos de la red falla podemos seguir accediendo a los datos sin problema. En blockchain cada nodo tiene una copia completa de todos los datos desde su origen. De esta manera cualquier nodo puede desconectarse en cualquier momento y otros nodos pueden sumarse a la red sincronizar los datos y constituirse rápidamente como un miembro más.

3.1.5. Sistema descentralizado

Blockchain es un sistema descentralizado²⁵. En un sistema centralizado todos los nodos confían en un nodo central que es el responsable de llevar a cabo las operaciones sobre los

¹github.com/hyperledger/fabric/wiki/Fabric-Releases

datos. ¿Pero qué ocurriría si ese sistema central sufriera un ataque o determinadas personas con interés dispares llevan a cabo modificaciones maliciosas sobre los datos saltándose todas las reglas? Pensemos en un banco y cualquier podría ser su impacto si un hacker o alguien de dentro de la organización decide modificar registros en ese nodo central y comenzar a desviarse fondos. Esto hace ver el inmenso poder que tienen este tipo de nodos en las redes centralizadas. Como hemos dicho en el punto anterior cualquier cambio en el Ledger o libro de transacciones tendría que ser validado por los usuarios encargados de ello. Si alguien intentara alterar los datos de alguna transacción sus esfuerzos se verían frustrado por el resto de miembros de la red además de que siempre quedarían grabados sus intentos.

3.1.6. Robustez de datos

Ateniéndonos al punto anterior blockchain es una red descentralizada donde todos los nodos pueden modificar y validar la información. Pero esto nos lleva al siguiente problema: ¿cómo se puede alcanzar la consistencia de los datos y el acuerdo entre todos los participantes? Tampoco se podría tener un nodo responsable en elegir qué nodo es el siguiente en modificar porque volveríamos al caso del exceso de poder y de la posible corrupción.

Esto se resuelve gracias al algoritmo de consenso. Tanto en Bitcoin como en Ethereum el algoritmo de consenso usado se llama Proof of Work (prueba de trabajo). Los mineros utilizan todo el poder computacional del hardware de sus máquinas para resolver un problema matemático⁶ utilizando criptografía a través de fuerza bruta. Ethereum se encuentra en vías de migrar a otro algoritmo llamado Proof of Stake que elimina la necesidad del poder computacional que consume amplia cantidad de energía. En este nuevo modelo básicamente al nodo que va a validar un bloque se le retiene una cierta unidad de divisa si la validación del bloque ha sido válida y no existe fraude. Este modelo debería ser implementado a lo largo de 2019.

3.1.7. Participantes

- Desarrolladores: Son los grupos encargados de desarrollar el software que implementa el protocolo y esto lo hacen siguiendo un modelo colaborativo. Estos desarrollos son opensource y pueden seguirse en Github:
 - Bitcoin: ²
 - Ethereum: ³
- Usuarios: ejecutan un software en su máquina mediante el cual se les asigna una cuenta y con ella pueden enviar y recibir divisa y ejecutar transacciones y sincronizar sus datos con el resto de la red.
- Mineros: activan en su software la capacidad de minado. Esto les distingue de los usuarios normales. Esta capacidad de minado les permite participar en la verificación de bloques a través de la competición de prueba de trabajo y ganar así recompensa en forma de divisa. Por ejemplo, la red de Bitcoin recompensa a los mineros con moneda bitcoins y en la red Ethereum la recompensa para los mineros es a través de ether.

3.1.8. Hashing

En blockchain se utilizan distintos mecanismos de criptografía que hacen posible la tecnología y la dotan de gran seguridad. Pero el más destacado son las funciones hashing que convierten cualquier texto a un hash.

Características principales de las funciones de hashing:

- No importa el tamaño del texto, el tamaño del hash resultante será siempre idéntico.
- Son deterministas. Esto quiere decir que para un mismo texto o entrada de datos siempre recibimos el mismo hash de salida. Un cambio en un solo bit puede hacer que nuestro hash cambie por completo.

²<https://github.com/bitcoin/bitcoin>

³<https://github.com/ethereum/go-ethereum>

- No son reversibles. Nunca a partir de un hash podremos encontrar el texto original. Lo único que podemos hacer para revertirlo es aplicar la función de hashing a un texto y comprobar si su hash de salida coincide con el original.
- Son fáciles para trabajar. Puesto que un hash puede ser calculado para textos extremadamente largos y su salida siempre tiene el mismo tamaño esto hace que sea mucho más fácil trabajar con hashes que con contenidos largos.

3.1.9. Hash en la cadena de bloques y Proof of Work

Los hashes son muy importantes dentro de la tecnología blockchain ya que se usan para enlazar a unos bloques con otros. Por ejemplo, Bitcoin utiliza funciones hash criptográficas SHA-256¹⁷.

Cada bloque contiene un hash único que identifica todos los datos que tiene dicho bloque y además contiene el hash del contenido del bloque anterior. Esto significa que si cambiáramos cualquier contenido de cualquier transacción de un bloque esto invalidaría el hash del bloque previo que también contiene que posee y esta invalidación se propagaría hasta el final de la cadena.

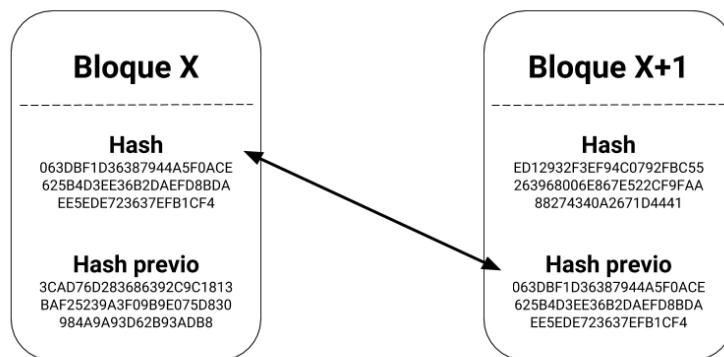


Figura 3.1: Bloques enlazados

Se podría pensar que lo único que se tiene que hacer para romper la seguridad es recalcular todos los hashes de los bloques porque calcular un hash es un proceso muy

rápido pero hay otra limitación:

El hash de un bloque tiene que ser más pequeño que un valor llamado difficulty target o dificultad objetivo que se establece para cada bloque.

Además cada bloque tiene un número arbitrario llamado nonce.

La única manera de calcular el siguiente hash es ir cambiando el valor del nonce hasta que el valor del hash resultante es menor que la dificultad objetivo. Cuando se cumple esto habremos calculado un bloque válido. Cuanto más pequeño sea el valor de la dificultad más difícil será calcular el valor del hash y es así como funciona el algoritmo de consenso

Proof of Work.

$$\text{hash}(\text{bloque} + \text{nonce}) < \text{dificultad} \rightarrow \text{bloque correcto}$$

Este es el trabajo que realizan los mineros cada vez que hay que verificar un bloque. El ganador de la competición emitirá cual es el número nonce con el que ha conseguido resolverlo y gracias a esto el resto de nodos utilizando dicho número nonce podrán verificar que el valor es correcto y por tanto el minador ha hecho la prueba de trabajo.

En resumen lo que hacen los mineros es ejecutar una de las funciones de hashing en las cuales proporcionan el contenido del bloque y un número que van cambiando de manera aleatoria hasta que el hash resultante es menor que un número que viene indicado por la dificultad del bloque. Este proceso requiere un gran esfuerzo computacional y gracias a esto vemos que no es tan sencillo minar bloques y por tanto corromper la red.

3.1.10. Minado

Recompensas

Recompensa que obtiene un minero por conseguir verificar o minar un bloque. Como hemos visto antes una vez que un minero descubre el número nonce que le hace ganar la competición del algoritmo de consenso lo envía a la red para que el resto de nodos lo verifique. Si el valor del nonce es correcto dicho minero tiene derecho a introducir una

primera transacción en el bloque donde se le transmitirá una cantidad de divisa a modo de recompensa por participar y ganar en la competición.

A día de hoy en Bitcoin la cantidad de recompensa es 12.5 bitcoins y en Ethereum 3 ethers. Aparte de recibir una cantidad fija de divisa el minero también recibe las comisiones que se han cobrado a los usuarios que han ejecutado transacciones en dicho bloque. Estas comisiones en Ethereum se denominan Gas. Son las comisiones que se les cobra a los usuarios por las transacciones.

Recompensar a los mineros es la manera que tienen las redes blockchain de generar divisa y de así asegurarse de que no existen malas intenciones a la hora de verificar los datos. Ya que si los mineros intentaran falsear información afectarían a la red y el valor de la divisa con la que son recompensados disminuiría lo cual sería como pegarse un tiro en el pie.

Tiempos

En blockchain existe un tiempo por defecto para poder generar un nuevo bloque. Es decir, cada implementación tiene un intervalo de tiempo establecido que ha de pasar para que se genere un nuevo bloque desde que el anterior fue creado. En Bitcoin está establecido en 10 minutos y en Ethereum de 15 segundos.

Este tiempo establecido es así por diseño y el sistema se encarga automáticamente de incrementar la dificultad del siguiente bloque si los tiempos de resolución de un conjunto de bloques baja o bien de disminuir dicha dificultad si los tiempos de minados de un conjunto de bloques se encuentran por encima del tiempo establecido en cada implementación. Incrementar o disminuir la dificultad de un bloque afecta directamente al tiempo que necesitarán los mineros para encontrar el hash con la solución. Es el mecanismo que se tiene para poder controlar el tiempo de minado de los bloques. Debido a esto no se recomienda blockchain si lo que se quiere es una aplicación con una gran velocidad de transacciones.

3.1.11. Profundizando en Ethereum

Transacciones

En blockchain los usuarios tienen un identificador que representa su cuenta desde la cual pueden enviar y recibir divisa. Puesto que blockchain es totalmente anónimo son los poseedores de una clave privada. Proteger la clave privada es vital, si alguien consiguiera acceder a ella sería capaz de hacer desaparecer todos los fondos disponibles en la cuenta.

Los usuarios pueden firmar transacciones utilizando su clave privada. El contenido de la transacción que se está enviando será encriptado con la clave privada dando lugar a la firma.

$$\text{Firma} = \text{clave privada} + \text{mensaje}$$

Será fácilmente verificable por terceros para comprobar que efectivamente esa transacción fue emitida por dicho usuario.

Las transacciones que se ejecutan normalmente en Ethereum son de divisa entre cuentas, transacción de usuario a usuario. Sin embargo, se diferencia de otras implementaciones porque también podemos enviar transacciones a smart contracts (que se explican en un punto más adelante).



Figura 3.2: Transferencia de cuenta a cuenta

Formato de las transacciones:

- TxHash: hash de la transacción que lo identifica de manera única.
- Timestamp: fecha y hora en la cual se emitió dicha transacción.

- Block: representa el número de bloque donde se ha incluido la transacción.
- Gas limit: indica la cantidad máxima que el usuario está dispuesto a pagar a modo de comisión por ejecutar una determinada transacción.
- Gas used: indica la cantidad final que gastó la transacción de dicha transacción.
- Gas price: precio/unidad de gas que se encontraba vigente en ese bloque. Es importante tener en cuenta que si un usuario que va a ejecutar una transacción establece un Gas limit inferior al gas price la transacción se abortará y todos los cambios que se hayan realizado en dicha transacción se revertirán.
- Input data: campo de datos que cuando se haga un envío de divisa irá vacío, pero cuando se realice una transacción a smart contracts representarán el código del smart contract al que van dirigidos o los datos que enviamos para poder interactuar con él.

Bloques

Campos más importantes de los bloques:

- Timestamp: representa la fecha y la hora en la cual se generó el bloque.
- Transactions: representa el listado de transacciones incluidas en dicho bloque.
- Hash: Identificador hash único del bloque.
- Prev Hash: Identificador del bloque anterior.
- Mined by: dirección de eth del minador que ganó el algoritmo de consenso.
- Difficulty: representa el valor de la dificultad del bloque utilizado para ajustar la dificultad de minado.
- Gas used: gas total gastado en comisiones por todas las transacciones incluidas en este bloque.

- Gas price: límite de gas máximo que tenía establecido el bloque.
- Nonce: solución que aportó el minero para encontrar un hash válido. Es incluido en el bloque para que todos los mineros puedan verificar su validez.
- Block reward: cantidad de eth que ganó el minero en total por validar el bloque (3 ethers + total de las comisiones que han pagado todos los usuarios).

Hay que tener en cuenta que existe un estado **génesis o estado inicial** que hace referencia al primer bloque de la cadena y en el que los campos pueden tomar un valor especial.

Smart Contracts

En Ethereum un smart contract (contrato inteligente) es un programa que vive en la cadena de bloques y del cual todos los nodos tienen una copia.

Es capaz de ejecutarse y hacerse cumplir por sí mismo de manera autónoma y automática sin intermediarios ni mediadores de manera totalmente descentralizada. Gracias a ser contratos digitales que especifican sus reglas utilizando un lenguaje de programación no existe la posibilidad de malas intenciones al no ser algo verbal o escrito.

Para poder desarrollar smart contract en Ethereum se utiliza el lenguaje solidity. Es un lenguaje tipado con una sintaxis muy parecida a javascript pero que tiene capacidades adicionales relacionadas con el manejo de transacciones y divisa en su propio diseño. Una vez compilamos un smart contract desarrollado con solidity este se convertirá a código máquina también llamado bytecode. El código bytecode se enviará en una transacción a la cadena de bloques de Ethereum creándose así una instancia de dicho contrato.

Todos los nodos tienen en su software la máquina virtual de Ethereum también conocida como EVM. Esta máquina virtual es capaz de ejecutar el código bytecode de los smart contracts.

3.2. Serverless

En el siguiente apartado se expone que se entiende por serverless, de donde viene el término y porque es tan importante actualmente.

3.2.1. ¿Qué es Serverless?

La computación sin servidores o serverless podemos entenderla como un modelo de computación en la nube cuyo objetivo es abstraer de los desarrolladores la gestión de servidores y las decisiones de infraestructura a bajo nivel. En este modelo, la asignación de recursos y todas las decisiones de infraestructura son gestionadas por el proveedor de la nube en lugar de por el arquitecto o desarrollador de la aplicación, lo que puede aportar algunos beneficios importantes. Por tanto el objetivo del serverless es hacer lo que dice su nombre: permitir desarrollar las aplicaciones sin preocuparse por implementar, ajustar o escalar un servidor, tan solo preocuparse del código.

3.2.2. ¿De donde viene? Evolución desde las *aaS

IaaS

IaaS (infraestructura como servicio) es la provisión de una infraestructura informática proporcionada por un tercero con el objetivo de que se pueda construir una aplicación sobre ella. Proporciona toda una infraestructura que contiene todo el hardware necesario (servidor, almacenamiento, imágenes de máquinas virtuales, red...) y todo el software o bibliotecas asociadas. Se administra a través de la conexión a la nube a través de Internet. Permite reducir o escalar verticalmente los recursos con rapidez para ajustarlos a la demanda necesaria por el cliente en ese momento, normalmente los gastos van relacionados con el uso que se haga de ella.

IaaS¹⁰ evita todo el gasto y la complejidad que suponen la adquisición y administración de equipos físicos propios y otras infraestructuras que se tengan como por ejemplo centros de datos. Cada recurso se ofrece como un componente de servicio independiente, y el cliente

solo tiene que contratar un recurso concreto durante el tiempo que lo necesite. El proveedor de servicios en la nube se encarga de administrar la infraestructura, mientras que el cliente se encargaría de la compra, instalación, configuración y administración de su propio software (sistemas operativos, middleware o aplicaciones). Aunque el proveedor no hace apenas gestión de los servicios que ofrece y es el cliente el que debe gestionar los servicios contratados.

Entre las ventajas de IaaS se encuentran:

- IaaS evita el gasto inicial de configurar y administrar equipos locales, por lo que constituye una opción económica para empresas de reciente creación o que quieran probar ideas nuevas. Por supuesto también se traduce en un ahorro tanto en la compatibilidad como en la mantenibilidad. No hay necesidad de mantener y actualizar el software y el hardware, ni de solucionar problemas en los equipos. Aparte del ahorro económico también se produce un gran ahorro de tiempo que hace que el cliente se pueda centrar en su actividad principal y se produzca un aumento de su productividad.
- Mejora la continuidad y la recuperación ante desastres externos comparado con el sistema tradicional. Lograr tener una alta disponibilidad y continuidad y recuperación ante desastres de forma interna puede resultar caro para el cliente ya que requiere una cantidad importante de tecnología y personal. Pero, a través de las IaaS se puede reducir este presupuesto y permitir el acceso a aplicaciones y datos con normalidad durante un momento en el que haya problemas de disponibilidad (como una catástrofe).
- Mejora el tiempo de la innovación frente a los modelos tradicionales. Tan pronto como el cliente se decida a comercializar o poner público un nuevo producto o un servicio, la infraestructura total necesaria puede estar actualizada en tiempos muy inferiores frente a lo que se tardaría en configurar la arquitectura tradicional. A su

vez esta característica también se podría utilizar en el caso en el que se quisiera escalar verticalmente los recursos con rapidez. Esto se podría dar por ejemplo ante un aumento de la demanda de los servicios ofrecidos en una situación puntual (navidad, vacaciones...). Por supuesto, también se puede escalar en el sentido inverso en el caso de que la demanda caiga drásticamente y nos encontremos con que nos sobran recursos.

Uno de sus ejemplos más conocidos es **AWS EC2**¹² que promete mayor flexibilidad, facilidad de implementación, escalabilidad instantánea y un vasto ecosistema de servicios de terceros, aunque requiere algo de configuración para hacerlo funcionar como un servidor tradicional. También existe la opción de **Google Compute Engine**¹¹ que destaca por su buen precio durante usos prolongados, por lo tanto pensado para alojar servicios que no son puntuales. Y por último **DigitalOcean**⁴ que es buena opción para servicios con mucha frecuencia ya que se basa en una tarifa fija.

PaaS

PaaS (Plataforma como servicio) es un modelo de negocio que proporciona una plataforma de desarrollo e implementación para los desarrolladores. Proporcionando a los desarrolladores los recursos para escribir y ejecutar sus propias aplicaciones en la plataforma. También ayuda a los desarrolladores a almacenar datos y gestionar el servidor.

Al igual que en IaaS el cliente compra solo los recursos que necesita al proveedor de servicios en la nube. La gran diferencia con PaaS es el modelo económico. En PaaS el cliente solo paga por el uso que hace de estos servicios. Es una capa que normalmente engloba a IaaS por tanto incluye infraestructura (servidores, almacenamiento, redes...), pero también incluye middleware, herramientas de desarrollo, servicios de inteligencia empresarial (BI), sistemas de administración de bases de datos, etc.

PaaS está diseñado con el objetivo de cubrir el ciclo de vida completo de las aplicaciones: compilación, pruebas, implementación, administración, entrega y actualización. Facilita el

⁴<https://www.digitalocean.com/>

desarrollo y la implementación sin el coste y la complejidad de comprar y gestionar la infraestructura subyacentes.

Entre sus ventajas se encuentran:

- Usar herramientas modernas y actualizadas a un precio asequible. El modelo de pago por uso permite que los clientes puedan usar software de desarrollo puntero si no necesitan de comprarlo lo que supone un avance frente al modelo tradicional.
- Administrar y gestionar el ciclo de vida de las aplicaciones de un modo más optimizado. Desde PaaS se proporcionan herramientas específicas para el control del ciclo de vida de las aplicaciones.
- Rapidez tanto en el desarrollo como en el despliegue. Automatiza mucho todo el proceso de puesta en el mercado de la aplicación del cliente frente al modelo tradicional.

Entre sus ejemplos más conocidos se encuentra **OpenShift**¹³, comúnmente utilizado para el despliegue y la integración con soporte para la mayoría de lenguajes y tecnologías. AWS cuenta con **Elastic Beanstalk**¹⁶ donde destaca su escalado automático y su monitorización. Por último encontramos **Force.com**⁵ que destaca por su rapidez en la configuración.

SaaS

SaaS (software como servicio) permite a los clientes/usuarios conectarse a aplicaciones que están alojadas en la nube y poder usarlas. Aquí el usuario no tiene que hacer ningún desarrollo o programación, pero es posible que el software si requiera una configuración aunque suelen ser muy flexibles, personalizables y orientadas hacia un tipo de usuario que no tiene porque ser desarrollador o poseer altos conocimientos de informática.

Aunque cualquier usuario a modo personal ha podido usar estas aplicaciones ya anteriormente (como por ejemplo aplicaciones de correo, notas o ofimática), más orientado

⁵<https://developer.salesforce.com/platform/force.com>

al mundo empresarial nos encontraríamos con aplicaciones empresariales sofisticadas: como CRM (administración de las relaciones con el cliente), ERP (planeamiento de recursos empresariales) o administración de documentos. El proveedor de SaaS ofrece una solución software alojada en la nube normalmente mediante el modelo de pago por uso como el que se ha explicado en las soluciones superiores. El cliente alquila el uso de una aplicación para que su organización, empleados y los usuarios puedan conectarse y hacer uso de ella.

Dentro del modelo de pago se suelen tener en cuenta los números de usuarios o funcionalidades disponibles.

Toda la infraestructura, el middleware, el software, los datos o cualquier otra información de las aplicaciones se encuentran con el proveedor y son responsabilidad suya. Desde el contrato entre el proveedor se pueden garantizar características como la disponibilidad de las aplicaciones ofrecidas o la seguridad de los datos, ya que pueden transmitir información relevante (por ejemplo, aplicaciones de correo electrónico). Al igual que en las soluciones anteriores, destaca por su capacidad para ejecutar aplicaciones a partir de un coste mucho menor que con el modelo tradicional.

Entre sus ventajas cuenta con:

- Modelo de pago por uso. Permite escalar rápidamente los recursos en función de los parámetros de uso.
- Usar software de forma gratuita. Los usuarios pueden ejecutar la mayoría de las soluciones SaaS directamente desde un navegador (en caso de ser aplicación web) o desde una aplicación gratuita (en caso de ir dirigido a plataforma móvil).
- Acceso a la solución desde cualquier lugar. Ya que no suelen requerir ninguna plataforma especial ni ninguna configuración determinada es muy fácil para los usuarios disponer de los datos casi en cualquier lugar donde se tenga un ordenador o un dispositivo móvil con navegadores que lo soporten.

- El cliente no tiene que preocuparse por ningún aspecto del desarrollo, configuración o con nada relativo al almacenamiento de los datos. Todo esto es responsabilidad del proveedor de servicios.

Quizá el ejemplo más conocido podría ser el paquete de Google **GSuite** (el cuál incluye multitud de servicios de Google como Gmail, Drive...). Otro ejemplo distinto sería la herramienta de comunicación **Slack**⁶ el cuál ha ganado mucha popularidad entre los desarrolladores. Distinto a los anteriores tendríamos **Salesforce**: uno de los CRM más populares hasta la fecha ⁷

FaaS

FaaS (función como servicio)¹⁹ es un concepto tecnológico que a través de dividir el software en funciones independientes permite a los desarrolladores ejecutar código en respuesta a eventos o peticiones de red sin necesidad de construir o mantener una infraestructura compleja. Esto significa que se puede delegar partes de la funcionalidad de una aplicación en la nube y que se ejecute de manera independiente. Suele estar muy relacionado con la construcción de microservicios. Es un término relativamente nuevo ya que apareció por primera vez en 2014 por hook.io (⁸) y poco después apareció en las demás plataformas principales. Surge para hacer frente al modelo tradicional monolítico de servidor REST el cual es mucho más difícil de escalar o de modularizar.

Estas funciones normalmente no tienen ningún estado y solo tienen ámbito local. Es decir que su ámbito se pierde al terminar su ejecución. Por lo que se suelen usar juntamente con otros servicios externos como servicios de almacenamiento o bases de datos para conseguir persistencia. Debido a esto la mayoría de las plataformas con servicios FaaS proveen de grandes facilidades para relacionar sus servicios y permite invocarlos conjuntamente.

Suelen funcionar como servicios bajo demanda. Esto significa que sólo gasta mientras el código esté ejecutándose, por tanto, estarán apagados todo el resto del tiempo. Esto hace se

⁶<https://slack.com/intl/es-es/>

⁷<https://www.salesforce.com/es/>

⁸<http://hook.io/>

pueda reducir el coste frente al modelo tradicional de un único servidor ya que no incurrirá en ningún gasto mientras no esté en uso.

Algunos de sus casos de usos típicos son funciones de procesamiento de datos, procesamientos por lotes, procesamientos ETL, servicios de IoT o microservicios aplicaciones web o móviles. Debido a este último caso el término FaaS se puede encontrar relacionado con el término Baas (backend as a service). Su ejemplo más conocidos serían las **Cloud Functions**¹⁴ y pionero de éstas sería **AWS Lambda** de la que se hablará más adelante. Similar a Lambda está **Azure Functions** que aparte de ejecutarse en Windows cuentan con la peculiaridad de que la memoria se asigna dinámicamente a la función.

También existe **Google Cloud Functions** la cual se diferencia de las demás por su rendimiento variante a las demás poseen rendimiento fijo. Similares a las anteriores también se encontraría la propuesta de IBM **Open Whisk**. Cabe destacar frente a las demás la propuesta de **Open Lambda**⁹ ya que no se encuentra dentro de ninguna gran plataforma. Su implementación se puede encontrar en su repositorio bajo licencia Apache.

⁹<http://www.open-lambda.org/>

Capítulo 4

Descripción de la solución

En el actual apartado se describirá la solución aportada a todo detalle, se nombrará la tecnología utilizada, se detallarán sus alcances y limitaciones y se aportaran ejemplos de su uso.

4.1. Tecnologías usadas

4.1.1. Golang

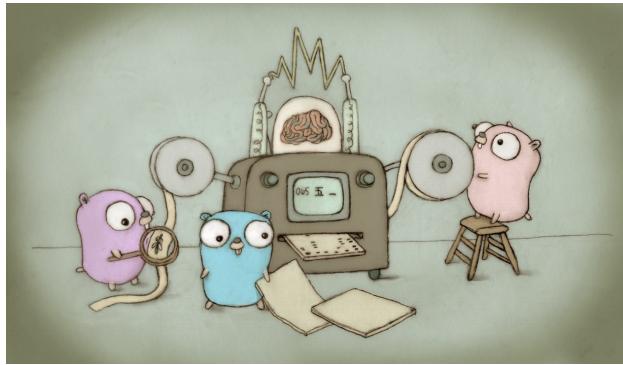


Figura 4.1: Imagen de Renee Grench bajo la licencia Creative Commons 3.0.

Go⁷ es un lenguaje de programación de código abierto que fue concebido en septiembre de 2007 por Robert Griesemer, Rob Pike y Ken Thompson, todos en Google, y fue anunciado en noviembre de 2009. Los objetivos del lenguaje y sus herramientas de acompañamiento eran ser expresivos, eficientes tanto en la compilación como en la ejecución, y eficaces en la

redacción de programas fiables y robustos.

Go¹⁸ es un lenguaje que puede parecer similar a C, pero no es solo una versión actualizada de este. Toma prestadas y adapta buenas ideas de muchos otros lenguajes, a la vez que evita características que han llevado a la complejidad y a la falta de fiabilidad del código. Suponen una novedad sus facilidades para la concurrencia, su enfoque para la abstracción de datos y la flexible programación orientada a objetos que proporciona. Además, dispone de recolector de basura.^automático.

Go está especialmente recomendado para la construcción de infraestructura como servidores en red, servicios backend y todo el mundo DevOps pero no hay que olvidar que es un lenguaje de propósito general. En los últimos años se ha vuelto popular como sustituto de los lenguajes de programación como Java, C o C++ ya que permite la concurrencia de forma simultánea y efectiva con múltiples procesos al igual que ellos, pero de una forma mucho más fácil gracias a sus rutinas y canales. Esto último es un punto importante para decidir usar este lenguaje en el desarrollo.

Como se ha dicho antes Go es un proyecto de código abierto, por lo que su código fuente, librerías y herramientas están disponible gratuitamente. Las contribuciones al proyecto provienen de la comunidad, aquí su repositorio en Github. Funciona en todos los principales sistemas operativos y la mayoría de sus programas no necesitan de modificar para funcionar en uno o en otro.

4.1.2. LibP2P



Figura 4.2: Logo de libp2p proveniente del repositorio de su implementación en go.

Libp2p¹ es un framework de red que permite desarrollar aplicaciones descentralizadas peer-to-peer (p2p). Originalmente formaba parte del proyecto del protocolo de red de IPFS, más adelante fue extraído para ser tratado como proyecto independiente y dar soporte a cada una de sus implementaciones. En este proyecto se ha utilizado su implementación con el lenguaje Go¹ bajo licencia MIT.

Libp2p ofrece ventajas indiscutibles ya que se encarga de la detección de los pares y de los protocolos de "handshake" para construir las conexiones entre ellos. Además, en un mundo donde los clientes también actúan como servidores, inevitablemente habrá una variedad de hardware, sistemas operativos y protocolos de comunicación entre los nodos. Libp2p admite tanto protocolos no cifrados (TCP, UDP) como cifrados (TLS).

Libp2p ha sido diseñado desde el principio para ser muy modular, de modo que puede ser implementado en muchos proyectos con redes p2p diferentes. Mientras que los nodos en las aplicaciones p2p tradicionales se denominan con una combinación de dirección IP y puerto, libp2p utiliza en su lugar el concepto de multidirección. No todos los proyectos que utilizan libp2p necesitan soportar todos los protocolos. El concepto de multidirección

¹<https://github.com/libp2p/go-libp2p>

existe para hacer posible ampliar libp2p con nuevos protocolos. Esto hace que en el futuro podremos, por ejemplo, añadir Bluetooth como protocolo de transporte.

El segundo aspecto principal de la modularidad de libp2p es su proceso de negociación de protocolos. Una vez que se ha establecido una conexión entre dos pares, lo único que maneja libp2p es negociar los protocolos que se usan en esa conexión y hace muy fácil al desarrollador poder trabajar con ella. Aunque se anima a los nodos a usar con un conjunto específico de protocolos comunes, ninguno de ellos es técnicamente obligatorio. Esto hace posible experimentar fácilmente con nuevos protocolos o nuevas ideas, y desplegar nuevas versiones de los protocolos sin dejar de ser compatible.

4.1.3. Node.js



Figura 4.3: Logo de Node.js obtenida del banco de imágenes Pixabay

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Orientado a eventos asíncronos lo hace ideal para construir fácilmente aplicaciones de red rápidas y escalables. Utiliza un modelo de E/S basado en eventos, sin bloqueo, que lo hace ligero y eficiente, perfecto para aplicaciones de datos intensivos en tiempo real que se ejecutan en dispositivos distribuidos.

Es de código abierto, y su repositorio es uno de los más visitados de Github. Su creador es Ryan Lienhart Dahl aunque la marca pertenece a la empresa Jeyent (en la que trabaja Dahl). Quien realmente mantiene la plataforma y el repositorio es el grupo de colaboradores de Node autorizados por su Comité Directivo Técnico. Cuenta además con

más de 15000 módulos comunitarios publicados en NPM², el gestor de paquetes con el que suele ir unido.

V8 le da a Node⁵ un gran impulso en rendimiento porque elimina al intermediario, prefiriendo la compilación automática en código máquina en lugar de la ejecución bytecode o utilizando un intérprete. También debido a que Node utiliza JavaScript en el servidor, hay también otros beneficios como que los desarrolladores pueden escribir aplicaciones web en un solo lenguaje. Esto sin duda ayuda a reducir el coste de cambiar entre los contextos de desarrollo del cliente y el del servidor, y permite compartir códigos entre el cliente y el servidor.

Node también suele ir ligado con el formato de intercambio de datos JSON³, el cual es nativo en JavaScript. JavaScript también es el lenguaje utilizado en las terminales de varias bases de datos NoSQL (como MongoDB).

4.1.4. AWS



Figura 4.4: Logo de Amazon Web Services

Dentro de los servicios de AWS se han usado tres fundamentalmente que pasan a explicarse a continuación:

²<http://npmjs.org>

³<https://www.json.org/>

IAM

AWS Identity and Access Management (IAM)²¹ es el servicio que permite crear varios usuarios y gestionar los permisos para cada uno de ellos dentro de la cuenta de AWS. Un usuario es una identidad o rol con credenciales de seguridad únicas que pueden utilizarse para acceder al resto de los servicios de AWS con unos permisos u otros. IAM elimina la necesidad de compartir contraseñas o claves de acceso y facilita la activación o desactivación del acceso de un usuario según las necesidades.

Además, IAM permite implementar las mejores prácticas en seguridad mediante la concesión de credenciales únicas a cada usuario dentro de su cuenta de AWS. Solo le permite acceder a los servicios y recursos de AWS necesarios que necesite para realizar su trabajo, dejando restringir incluso en funcionalidades por servicio. Es seguro de forma predeterminada ya que los nuevos usuarios no tienen acceso a ningún servicio hasta que se le concedan de forma explícita, esto le hace realmente útil para grupos de trabajo en los que un administrador de servicios vaya dando acceso a los desarrolladores/usuarios según se vaya necesitando. Aunque IAM es relativamente nuevo está integrado de forma nativa en la mayoría de los servicios de AWS.

S3

Amazon Simple Storage Service (Amazon S3)²² es el sistema de almacenamiento simple de Amazon. Conceptualmente, es un almacén infinito para objetos de tamaño variable (mínimo 1 Byte, máximo 5 GB). Un objeto es simplemente un contenedor de bytes que se identifica por una URI (a la que tenemos acceso desde la interfaz de S3).

Los clientes pueden leer y actualizar objetos⁹ en S3 de forma remota utilizando las librerías que proporciona con funciones similares a los protocolos SOAP o REST. Por ejemplo la operación *get(uri)* devuelve un objeto y *put(uri, bytestream)* escribe una nueva versión del objeto. El cubo(bucket) aparte de almacenar los archivos además contiene las versiones previas de dichos archivos lo que hace que podamos llevar un control de

versiones. Esto hace que por medio de cabeceras especiales en las operaciones se pueden incluir parámetros especiales que permiten, por ejemplo, recuperar la nueva versión de un objeto sólo si el objeto ha cambiado desde la marca de tiempo especificada. Además, los meta datos definidos por el usuario pueden asociarse a un objeto y pueden ser leídos y actualizados independientemente del resto del objeto, por ejemplo, para registrar una marca de tiempo del último cambio. Cuando un usuario crea un nuevo objeto, el usuario especifica en qué cubo debe colocarse el nuevo objeto. S3 proporciona varias formas de obtener u operar a través de un cubo. Por ejemplo, un usuario puede recuperar todos los objetos de un cubo o sólo aquellos cuyos URIs coinciden con un prefijo específico.

Muy importante es que el cubo puede ser tratado como una parte fundamental de la seguridad: los usuarios pueden conceder autorización de lectura y escritura a otros usuarios para cubos enteros, o bien se pueden conceder privilegios de acceso a objetos individuales. Además, se puede hacer que los objetos o el cubo sean totalmente públicos y cualquiera apuntando a la URI del cubo o objeto sea capaz de obtenerlo.

Lambda

AWS Lambda es el servicio que permite implementar arquitecturas de microservicios²² sin necesidad de gestionar servidores (bajo arquitectura serverless). De este modo, facilita la creación de funciones (es decir, microservicios) que pueden ser fácilmente desplegadas y escaladas automáticamente, y también ayuda a reducir los costes de infraestructura y operaciones para la computación. Este servicio está diseñado para ofrecer una estructura de costes por solicitud²³, lo que significa que los desarrolladores sólo tienen que preocuparse de escribir funciones individuales para implementar cada microservicio y luego desplegarlas en AWS Lambda.

AWS Lambda permite escribir las funciones en los lenguajes Node.js (JavaScript), Python, Java (compatible con Java 8) y C# (.NET Core). Este tipo de arquitectura es adecuada para aplicaciones que requieren de gran computación de datos a altas velocidades o computaciones paralelas. Estas funciones lambda son invocadas mediante las bibliotecas

proporcionadas por AWS al igual que otros servicios (como S3). Son funciones que no tienen contexto, es decir, los únicos datos de los que poseen son los que vienen incluidos en la llamada por la que son invocadas. Ha recibido ciertas críticas porque esta última característica puede requerir mayor habilidad por parte de los desarrolladores para crear las funciones sin embargo es alabada por su gran utilidad a la hora de procesar grandes cantidades de datos. Además, el problema del contexto se puede subsanar conectando las funciones Lambda con otros sistemas de almacenamiento de AWS ya que están disponibles directamente desde las funciones.

4.1.5. Gource

Gource⁴ es una herramienta de visualización por la cual los proyectos de software son mostrados como un árbol animado con el directorio raíz del proyecto en su centro. Los directorios aparecen como ramas y los archivos como hojas. Se puede ver a los desarrolladores trabajando en el árbol en los momentos en que contribuyeron al proyecto. El repositorio de código de Gource está disponible en su página de Github bajo licencia GNU. Aparte de la instalación disponible en el repositorio Gource también está disponible en gestores de paquetes como Homebrewer.



Figura 4.5: Captura de pantalla de Gource obtenido de su web.

Analiza los archivos de registro (logs) generados en el repositorio Git. También posee soporte para Mercurial, Bazaar, SVN y también puede analizar los registros producidos por varias herramientas de terceros para repositorios CVS. Gource genera un video que es ejecutado al momento siguiendo los movimientos que se encuentra en el archivo de log del

⁴<https://gource.io/>

repositorio. Dispone de cantidad de opciones de personalización como cambiar los iconos, el título, la velocidad de reproducción etc.

4.1.6. Postman

Postman²⁰ es una extensión de navegador Chrome para hacer peticiones HTTP. Ofrece una gran cantidad de características que hacen que sea fácil desarrollar, probar y documentar una API REST. También existe una versión de escritorio de la aplicación que proporciona características adicionales como la carga masiva que no están disponibles en la extensión del navegador. Puede ser descargado e instalado desde la Chrome Web Store o desde su sitio web⁵. Además, es de licencia gratuita para uso individual y grupos pequeños.

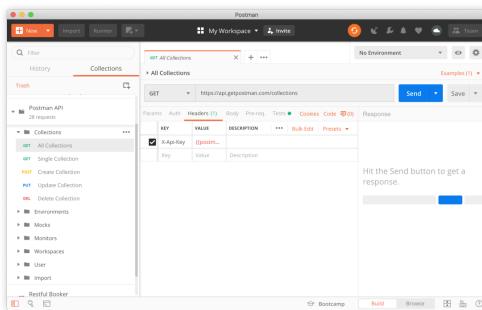


Figura 4.6: Ejemplo de uso de Postman obtenido de su web.

Postman proporciona una interfaz de usuario limpia e intuitiva para componer una solicitud HTTP, enviarla a un servidor y ver la respuesta HTTP. También guarda automáticamente las solicitudes, que están disponibles para futuras ejecuciones. Se puede ver la solicitud guardada en la sección de historial de la barra lateral izquierda. Postman facilita la agrupación de las llamadas a la API en colecciones. Es posible tener subcolecciones de peticiones bajo una colección.

⁵<https://www.getpostman.com/>

4.2. Esquema

En la siguiente imagen se puede ver el esquema general de la solución:

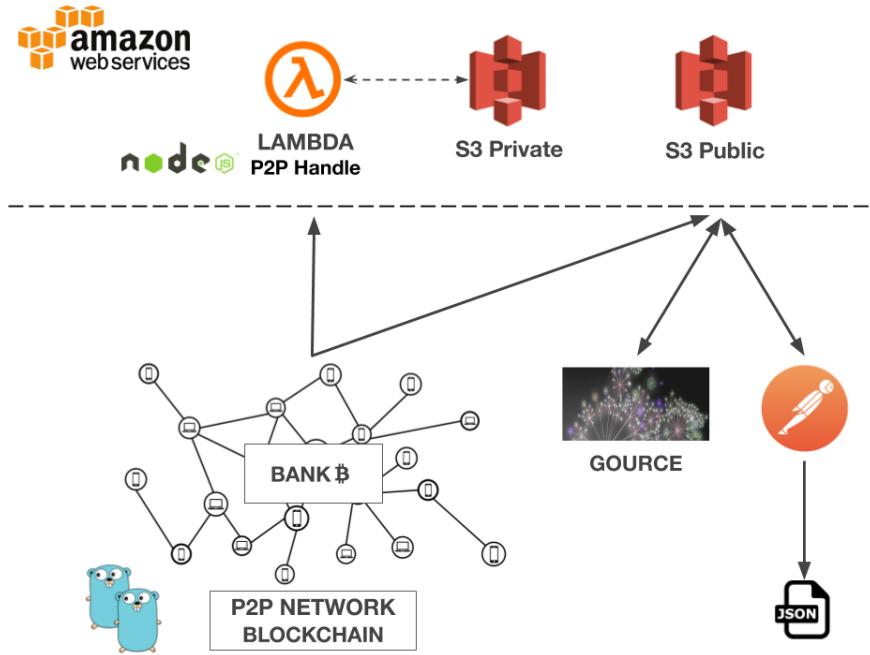


Figura 4.7: Esquema de la solución

Como se puede ver en el esquema superior la solución propuesta consiste en la creación de un cliente el cual interactúa con una aplicación de banca electrónica desarrollada bajo una arquitectura blockchain. Esta aplicación se conecta con el servicio AWS Lambda para manejar la gestión de nodos y conexiones relativos a la red P2P de la blockchain. A su vez, AWS Lambda se conecta con el servicio de AWS S3 para conseguir persistencia de datos y almacenar el estado de la red.

La aplicación también se conecta con los servicios de S3 para guardar en un bucket público los estados actuales de las cuentas bancarias y de las transacciones de la red blockchain. También almacena en el mismo bucket público los movimientos realizados por los nodos de la red en un archivo de log.

Por tanto, este bucket público contendrá dos (banco y blockchain) archivos en formato

JSON listos para ser “consumidos” por cualquier aplicación externa con los datos actualizados del estado actual de la aplicación y un archivo de log que irá almacenando todos los movimientos que se produzcan internamente en la red blockchain para que sea compatible con la herramienta de visualización Gource.

El código de la solución se encuentra disponible en el siguiente repositorio de Github bajo licencia MIT⁶. Url del repositorio del proyecto:

<https://github.com/daniOrtiz11/blockchain-serverless>

En los apartados siguientes se explica el funcionamiento de cada parte, así como su instalación y la puesta en marcha necesaria para ejecutar la solución.

4.3. Amazon Web Services

En esta sección se va a ver como se han usado los servicios de AWS dentro de la solución. El único requisito para la configuración de los servicios fue crearse una cuenta de AWS. Como se dijo en secciones anteriores, todos los servicios usados se han mantenido dentro de la capa gratuita.

Importante: dentro del repositorio de la solución se encuentra una copia de todas las funciones Lambda y archivos de ejemplo de los buckets de S3.

4.3.1. IAM

El servicio de IAM se ha necesitado usar para dar acceso desde las funciones Lambda (que se explican en la siguiente sección) a los buckets de S3 necesarios para la recuperación y persistencia del estado de los nodos.

Para ello se necesita crear un rol de ejecución de Lambda hacia S3. Basta con ir a la consola de IAM, elegir la opción de crear un rol, seleccionamos el tipo de entidad **Servicio**

⁶<https://github.com/daniOrtiz11/blockchain-serverless/blob/master/LICENSE>

de AWS y servicio Lambda. En los permisos se selecciona **AWSLambdaExecute**, tal como podemos ver en la imagen:

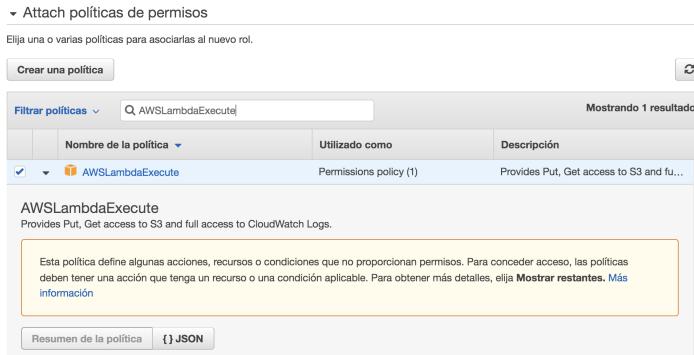


Figura 4.8: Permisos a seleccionar para crear el rol

4.3.2. Amazon S3

El servicio de S3 ha sido utilizado de dos formas distintas en la solución:

1. Usado conjuntamente junto con las funciones Lambda para gestionar los nodos de la red blockchain.
2. Como repositorio público para almacenar los archivos de salida.

Respecto al primero de los bucket cuando entramos en el podemos ver dos archivos:



Figura 4.9: Bucket privado

Su única peculiaridad es que es un bucket completamente privado, característica que se configura fácilmente durante la fase de creación. En la imagen podemos ver dos archivos

distintos: **p2pstate.json** y **p2pstateTest.json**. El segundo únicamente es utilizado en la fase de pruebas y no tiene ninguna relevancia en la solución final, el primero lo explicamos a continuación.

En el archivo **p2pstate.json** se almacena el estado actual de las conexiones entre nodos de la red blockchain. Es el archivo que sirve para persistir los resultados y del que parten las funciones Lambda para actualizar dicho estado. Las únicas peticiones que llegan a este archivo y a este bucket serán las que vengan de dichas funciones Lambda. Por tanto, se puede concluir que es un archivo de trabajo y que va a estar actualizándose constántemente. Al funcionar blockchain como una red p2p en este archivo nos encontramos como se están conectando los nodos entre sí y que pares se han formado.

Por dentro sigue la siguiente estructura:

```
[  
  {  
    "Ip": "10.10.0.1",  
    "Port": "1000",  
    "Key": "xxxxxxxxxxxxxx1",  
    "PrevKey": "",  
    "Configure": true  
  },  
  {  
    "Ip": "10.10.0.2",  
    "Port": "1001",  
    "Key": "xxxxxxxxxxxxxx2",  
    "PrevKey": "xxxxxxxxxxxxxx1",  
    "Configure": true  
  },  
  {  
    "Ip": "",  
    "Port": "",  
    "Key": "",  
    "PrevKey": "xxxxxxxxxxxxxx2",  
    "Configure": false  
  }  
]
```

Listing 4.1: *p2pstate.json*

Como se puede observar dentro del documento JSON se almacena un array con los datos de los nodos actuales en la red. Los parámetros que se guardan son: la dirección IP, el puerto, la clave, la clave del nodo al que está conectado y si está configurado o no. Este array de datos es con el que trabajan las funciones Lambda.

Se pueden dar dos casos especiales dentro del array:

1. Nodo inicial: es el primer nodo que se conecta a la red y por tanto su atributo *PrevKey* estará vacío.
2. Último nodo: es un nodo que aún no existe pero se deja preparado el hueco en el array con el atributo *PrevKey* del nodo al que se va a conectar y el atributo *Configure* con el valor *false*.

Al finalizar la conexión de todos los nodos de la red el archivo se debe encontrar únicamente con un array vacío. Esto indica que estaría preparado para la siguiente ejecución de la red.

Respecto al segundo de los bucket llamado **blcserverlessbucket** recordamos que es un bucket público que sirve para almacenar las salidas de la aplicación y su archivo de log. Es importante recalcar que solo se ha habilitado como público las peticiones para acceder a objetos o crear listas de ellos, las peticiones para eliminar o insertar siguen siendo privadas. El bucket está disponible a través de la siguiente url:

<https://s3.us-east-2.amazonaws.com/blcserverlessbucket>

Al ser público cualquier petición de obtención de los archivos o del estado del bucket será aceptada pero únicamente el Cliente de la aplicación blockchain tendrá acceso de escritura, por lo tanto es el único que puede modificar los archivos existentes (aparte, por supuesto, del creador del bucket a través de la consola de AWS).

Podemos ver el aspecto en las siguientes imágenes:



Figura 4.10: Bucket público

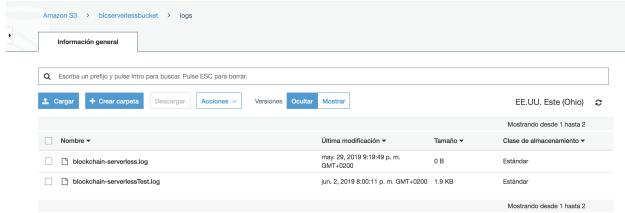


Figura 4.11: Carpeta de logs

Dentro de la carpeta principal podemos ver dos archivos: **bank.json**, **blc.json**. El objetivo de que estos dos archivos estén públicos y en formato JSON es que puedan ser consumidos por una aplicación externa para así poder mostrar el estado de la blockchain y de la aplicación bancaria.

Dentro del archivo de banco se tiene la siguiente estructura:

```
[  
  {  
    "PublicID": "yyyyyyyyyyyyyy1",  
    "PrivateID": "yyyyyyyyyyyyyy2",  
    "Name": "Jon",  
    "Amount": 90  
  },  
  {  
    "PublicID": "yyyyyyyyyyyyyy3",  
    "PrivateID": "yyyyyyyyyyyyyy4",  
    "Name": "Sansa",  
    "Amount": 110  
  }  
]
```

Listing 4.2: bank.json

Se pueden observar los campos de la clave pública y privada (más adelante se explicará como se usan), el nombre del usuario de la cuenta y la cantidad de monedas que posee.

Dentro del archivo de blockchain se tiene la siguiente estructura:

```
[
  {
    "Index": 0,
    "Timestamp": "2019-05-04 20:26:49.390042 +0200 CEST m=+0.024890473",
    "Transaction": {
      "SourceID": "sourceGenesis",
      "TargetID": "targetGenesis",
      "Amount": -2
    },
    "Hash": "zzzzzzzzzzzzz1",
    "PrevHash": ""
  },
  {
    "Index": 1,
    "Timestamp": "2019-05-04 20:27:23.681185 +0200 CEST m=+34.315518368",
    "Transaction": {
      "SourceID": "yyyyyyyyyyyyy1",
      "TargetID": "Jon",
      "Amount": -1
    },
    "Hash": "zzzzzzzzzzzzz2",
    "PrevHash": "zzzzzzzzzzzzz1"
  },
  {
    "Index": 2,
    "Timestamp": "2019-05-04 20:27:24.403669 +0200 CEST m=+35.037991021",
    "Transaction": {
      "SourceID": "yyyyyyyyyyyyy1",
      "TargetID": "yyyyyyyyyyyyy2",
      "Amount": 0
    },
    "Hash": "zzzzzzzzzzzzz3",
    "PrevHash": "zzzzzzzzzzzzz2"
  },
  {
    "Index": 3,
    "Timestamp": "2019-05-04 20:27:37.6989 +0200 CEST m=+40.292842138",
    "Transaction": {
      "SourceID": "yyyyyyyyyyyyy3",
      "TargetID": "Sansa",
      "Amount": -1
    },
    "Hash": "zzzzzzzzzzzzz4",
    "PrevHash": "zzzzzzzzzzzzz3"
  },
  {
    "Index": 4,
    "Timestamp": "2019-05-04 20:27:38.408993 +0200 CEST m=+41.002923791",
    "Transaction": {
      "SourceID": "yyyyyyyyyyyyy3",
      "TargetID": "yyyyyyyyyyyyy4",
      "Amount": 0
    },
    "Hash": "zzzzzzzzzzzzz5",
    "PrevHash": "zzzzzzzzzzzzz4"
  },
  {
    "Index": 5,
    "Timestamp": "2019-05-04 20:28:21.464959 +0200 CEST m=+92.098375070",
    "Transaction": {
      "SourceID": "yyyyyyyyyyyyy1",
      "TargetID": "yyyyyyyyyyyyy3",
      "Amount": 10
    },
    "Hash": "zzzzzzzzzzzzz6",
    "PrevHash": "zzzzzzzzzzzzz5"
  }
]
```

Listing 4.3: *blc.json*

En este archivo vemos todos los bloques que se han creado en la red blockchain. Cada bloque tiene los campos de índice, fecha de creación, transacción, clave y clave anterior.

Se pueden adivinar distintos formatos de bloque si se observan las transacciones y estos son: formato de bloque inicial de la blockchain, formatos para la creación de las cuentas (uno para transmitir el nombre del titular y para su clave privada) y formato para las transacciones entre cuentas. Hay que tener en cuenta que las transacciones entre cuentas se realizan sirviéndose como identificador de la clave pública de las cuentas, por lo que si se dispone de este archivo se puede llegar a obtener el estado actual de todas las cuentas bancarias.

Respecto a los archivos en la carpeta de logs podemos encontrar, como en otros casos, el archivo de log de la aplicación y un archivo de test a modo de ejemplo.

El archivo de log tiene la siguiente estructura:

```
20190515154656|127.0.0.1:10000|A|blockchain|#FFFFFF  
20190515154706|127.0.0.1:10000|A|blockchain|#FFFFFF  
20190515154707|127.0.0.1:10001|A|blockchain|#FFFFFF  
20190515154710|127.0.0.1:10000|A|127.0.0.1:10000/bank/Jon|#FFFFFF  
20190515154712|127.0.0.1:10001|M|blockchain|#FFFFFF
```

Listing 4.4: *blockchain-serverless.log*

Se puede diferenciar una línea por cada log que incluye tiempo, autor y recurso afectado. Hay que tener en cuenta que el formato del archivo ha sido adaptado para ser leído por la herramienta Gource y puede llegar a ser largo y complicado de leer. Ha sido adaptado siguiendo las indicaciones de su repositorio.

Como se ha dicho antes estos archivos se van actualizando según se van realizando movimientos en la red blockchain, por lo que siempre estarán actualizados. Hay que tener en cuenta que en caso de desconectar todos los nodos de la red estos archivos no se borran por lo que tendrán el último estado de la aplicación. Al comenzar una nueva ejecución de la aplicación cuando se conecte el primer nodo estos archivos ya volverán a estar actualizados.

4.3.3. Lambda

El servicio de AWS Lambda se ha usado para cumplir dos tareas: manejar la estructura de los nodos en la red y para actualizar el archivo de logs de la aplicación.

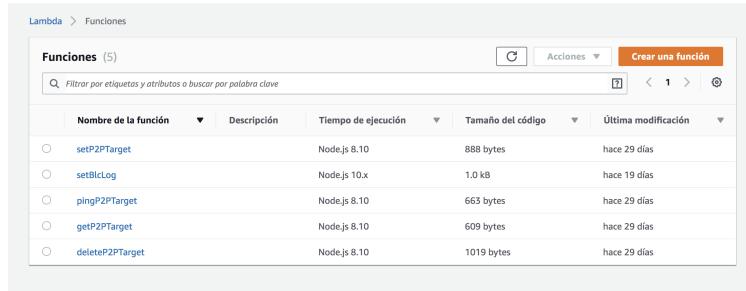


Figura 4.12: Consola de AWS Lambda actual

Para ello se han creado cinco funciones. Estas funciones se han programado bajo el lenguaje Node.js. Cuatro corresponden al manejo de los nodos y una al log. Es importante recordar que al crear las funciones se ha seleccionado el rol previamente creado en el servicio de IAM, para poder tener acceso a los archivos existentes en S3 desde estas funciones.

Las funciones referentes al control de los nodos son:

- **pingP2PTarget:** comprobación de si el nodo sigue estando estable dentro de la red. Es posible que su par esté en estado inestable o haya sido desconectado y él haya sido eliminado, en ese caso se comunica.
- **getP2PTarget:** obtención del nodo al que debe conectarse el nodo nuevo que se quiere unir a la red.
- **setP2PTarget:** configuración del nuevo nodo en la red. Deja establecido el nuevo nodo y deja preparado el siguiente para la próxima llamada.
- **deleteP2PTarget:** Elimina el nodo de la red. En caso de tener nodos conectados a él también los elimina de la red, así como a los "hijos" de estos. Posteriormente deja preparado el último nodo para la siguiente conexión.

Aunque cada una realiza una función distinta todas siguen el mismo patrón:

1. Formateo de parámetros recibidos por eventos.

2. Conexión con S3 y recuperación de datos.
3. Función particular de cada uno y actualización de S3 si fuera necesario.
4. Devolución de respuesta al cliente, siempre controlada.

Respecto a la función **setBlcLog** es la encargada de establecer el nuevo log que llega por evento al archivo de log. Se encarga de adecuar la nueva entrada al formato de Gource.

Hay que destacar que también se ha hecho uso del servicio de AWS **CloudWatch** para ayudar a depurar las funciones Lambda sirviendo para mostrar los resultados de estas.

4.4. Cliente Blockchain

En este apartado se va a explicar en qué consiste el cliente blockchain, la estructura de los datos, su instalación y su funcionamiento.

4.4.1. Estructura de los datos

El cliente blockchain está desarrollado usando el lenguaje Go ayudándose (aparte de por sus bibliotecas más comunes) por LibP2P para el manejo de la red p2p y por el SDK de AWS para Go.⁷ para las comunicaciones con éste. Los datos que usa el cliente para el correcto funcionamiento de la aplicación blockchain se detallan a continuación.

Estructuras y variables

Los bloques que conforman la red blockchain están definidos de la siguiente manera:

```
// Transaction struct
type Transaction struct {
    SourceID string
    TargetID string
    Amount    int
}

// Block struct
type Block struct {
    Index        int
    Timestamp   string
    Transaction Transaction
    Hash         string
    PrevHash    string
}

//Blockchain is a series of Blocks
var Blockchain [] Block
```

Listing 4.5: *Estructura de datos*

Como se pudo ver cuando se analizan los archivos JSON en el apartado de Amazon S3 cada bloque cuenta con un índice, la fecha de su creación, su hash, el hash del bloque anterior y

⁷<https://github.com/aws/aws-sdk-go>

un objeto de transacción. Este objeto de transacción está formado por dos identificadores de cuentas y una cantidad a transferir. Por tanto la variable *Blockchain* es una *Slice* (estructura de Go equivalente a un array de tamaño variable) de bloques.

Las cuentas de la aplicación bancaria están estructuradas de la siguiente manera:

```
// Account struct
type Account struct {
    PublicID  string
    PrivateID string
    Name       string
    Amount     int
}
//Bank is a series of Accounts
var Bank []Account

//account is the account of user logged
var account Account
```

Listing 4.6: Estructura de datos

Cada cuenta está formada por el nombre del titular, la cantidad de divisa que tiene y dos claves: clave pública y clave privada. La variable *Bank* hace referencia al banco y está compuesto por una colección de cuentas. Además, el nodo también guarda en la variable *account* la cuenta del usuario que ha iniciado sesión.

Para controlar las conexiones entre nodos se usa lo siguiente:

```
//LocalP2P struct
type LocalP2P struct {
    Ipdir   string
    Port    string
    Key     string
    PrevKey string
}

//localP2P is the local dir node
var localP2P LocalP2P

var targetP2P string
```

Listing 4.7: Estructura de datos

Cada dirección en la red p2p estará formada por una dirección ip, un puerto, una clave y la clave del nodo al que está conectado. Cada nodo almacena su dirección local y la clave del nodo al que debe conectarse.

Además, cuenta con las siguientes variables para su correcto funcionamiento:

```
var mutex = &sync.Mutex{}
var listenF *int
var seed *int64
var logged bool
```

Listing 4.8: Estructura de datos

Hacen referencia a un mutex para prevenir accesos concurrentes mientras se editan variables críticas, el puerto al que desea conectarse el cliente, la semilla para los algoritmos de generación de hashes (en caso de usarse) y una variable para saber si el nodo cuenta con un usuario conectado.

4.4.2. Funcionamiento interno

Todos los nodos que arranquen el cliente seguirán inicialmente el mismo camino de ejecución. El ejecutable se arranca desde una consola de comandos. Las únicas opciones que se permiten al arrancar son los parámetros opcionales *-l* referente al puerto que se quiere usar o *-seed* si se quiere usar semilla.

```
blockchain-serverless -l 8989 -seed 23
```

O sin ningún parámetro:

```
blockchain-serverless
```

En caso de no indicar puerto el cliente correrá por defecto en el puerto 10000. Pasos que siguen todos los nodos al arrancar:

- Inicialización de todas las estructuras y creación de un bloque génesis que se incluye dentro de su blockchain.

- Obtención de dirección apropiada por medio de LibP2P y creación de host para formar parte de la red y almacenamiento de su propia dirección. Se indica como función controladora del host a las funciones de lectura y escritura (que se explican más adelante). A estas funciones se les pasa un buffer creado a partir de la red para permitir la comunicación entre nodos.
- Obtención (por medio de invocación a Lambda) del estado actual de la red y de la dirección a la que debe conectarse. Dependiendo de si son el nodo inicial o no siguen distintos caminos:
 - En caso de ser el nodo inicial abre el canal de streaming al que deben conectarse los siguientes nodos y establece su dirección en el almacenamiento de S3.
 - En caso contrario también abre el canal streaming realiza la configuración a través de LibP2P del nodo al que conectarse (obtenida su dirección anteriormente). Realiza las comprobaciones necesarias, como que la dirección no esté repetida, fija su dirección en S3 y termina de establecer la conexión con el nodo al que está conectado. Termina yendo de nuevo a las funciones controladoras de la conexión.

Función de lectura

La función de lectura (por medio del puntero al buffer de la conexión) se encarga de comprobar si ha habido cambios entre la blockchain de un nodo y otro.

Esta función se está ejecutando constantemente mientras ese nodo esté vivo. Comienza comprobando si se ha escrito algo en el buffer entre los dos nodos que conecta. En caso de que se haya escrito algo, si se detecta que se trata de una estructura de bloques, comienza a tratarse. Esto significa que el nodo, en la otra punta de la conexión, ha **actualizado la cadena de bloques**. Esta es la manera que tienen de notificarse los extremos actualizaciones en los datos.

Verifica las longitudes de las dos cadenas (la suya y la entrante del buffer) y en caso de que la nueva tenga una longitud mayor se actualiza. Además, a partir de los datos de la nueva blockchain actualiza también el banco con sus respectivas cuentas y movimientos.

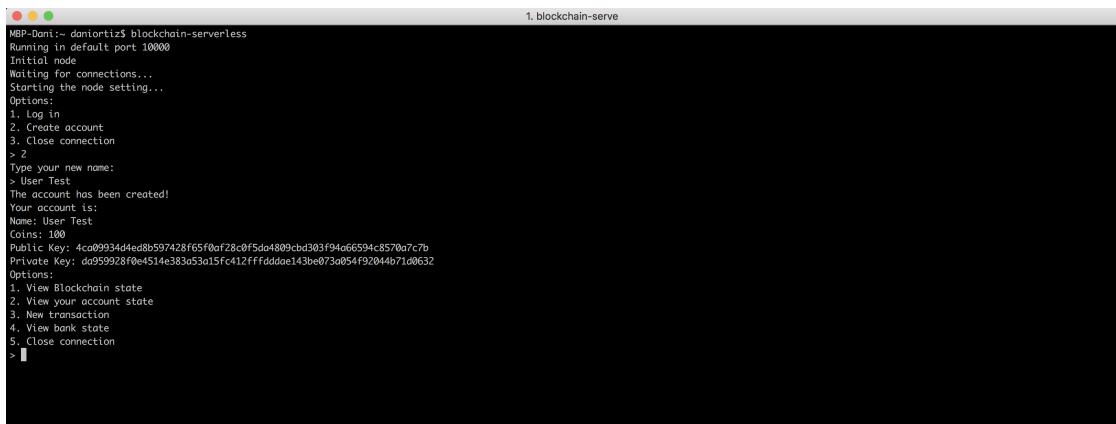
Función de escritura

La función de escritura es la función principal de interacción con el usuario. Es la encargada de las operaciones y de escribir los cambios en el buffer de la conexión.

Esta función está dividida en dos partes: actualizaciones y menús. Por un lado, la primera parte es la que se encarga de actualizar el buffer cada cinco segundos escribiendo en él la cadena de bloques. Esta blockchain es la que se comparará en la función de lectura del otro nodo. Se estará ejecutando siempre, de tal modo que si hay algún cambio en la blockchain tarde como máximo cinco segundos en transmitirse al nodo al otro lado de la conexión. La otra parte consiste en el menú y las funciones principales.

4.4.3. Menú inicial

El primer paso tras conectarse el nodo a la red es pasar por el menú principal.



```
MBP-Dani:~ danioritz$ blockchain-serverless
Running in default port 10000
Initial node
Waiting for connections...
Starting the node setting...
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> User Test
The account has been created!
Your account is:
Name: User Test
Name: User Test
Coins: 100
Public Key: 4cd09934d4ed8b597428f65f0af28c0f5da4809cb303f94a66594c8570a7c7b
Private Key: da959928f0e4514e383a53a15fc412fffdaddae143be073a054f9204671a0632
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> |
```

Figura 4.13: Menú inicial y creación de cuenta

En el podemos encontrar las opciones de inicio de sesión, creación de cuenta y cerrar conexión.

En la opción de creación de cuenta se pedirá que se proporcione el nombre de usuario que se desee y con él se creará la cuenta. Se recibirán las claves pública y privada y las monedas con los que se crea la cuenta. Es importante guardar las claves ya que las necesitaremos para otras operaciones. Esta creación de cuenta también se verá reflejada con nuevos bloques para propagar la nueva cuenta al resto de nodos.

En la opción de inicio de sesión se requerirá introducir la clave privada para poder acceder al control de la cuenta.

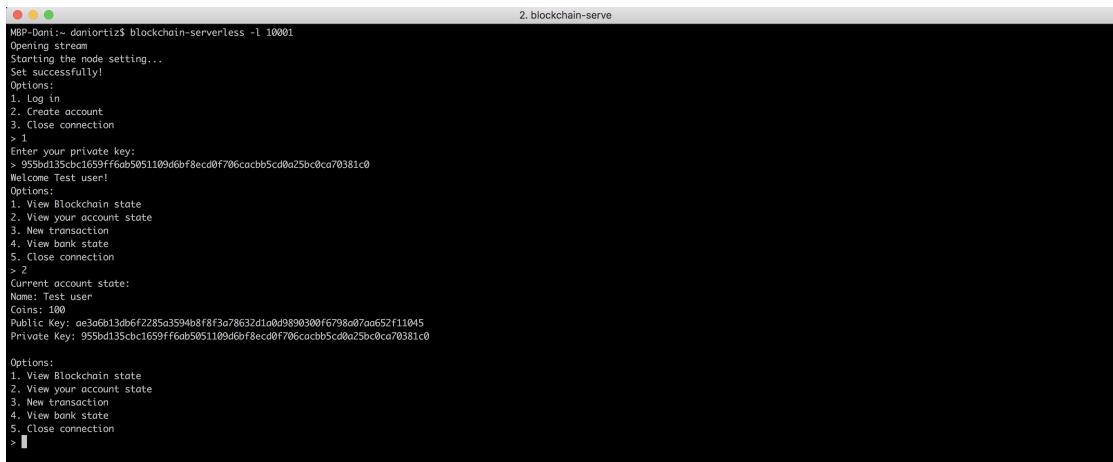
La opción de cerrar conexión (que también dispondrá el siguiente menú) consiste en cerrar la conexión e invocar la función Lambda para eliminar el nodo del estado de la red. Esta opción también se invoca si se produce alguna señal que interrumpe la entrada y cierre abruptamente el cliente (típicamente Ctrl-C).

4.4.4. Menú de usuario

El menú de usuario consta de las opciones de visualización de la cuenta propia, el estado del banco, el estado de la blockchain o la realización de una nueva transacción. Aparte del cierre de conexión explicado anteriormente.

Opciones de visualización

Las opciones de visualización muestran por consola el estado actual del banco y de la blockchain (similar a los archivos almacenados en la salida de S3). Además, la visualización de la cuenta permite comprobar las claves pública y privada y las monedas disponibles.



```
M&P-Dani:~ danioritz$ blockchain-serverless -l 10001
Opening stream
Starting the mode setting...
Set successfully!
Options:
1. Log in
2. Create account
3. Close connection
> 1
Enter your private key:
> 955bd135cbc1659ff6db5051109d6bf8ec0f706cacbb5ca0a25bc0ca70381c0
Welcome Test user!
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 2
Current account state:
Name: Test user
Coins: 100
Public Key: ae3a6b13db6f2285a3594b8f8f3a78632d1a0d989030f6798a07ad652f11045
Private Key: 955bd135cbc1659ff6db5051109d6bf8ec0f706cacbb5ca0a25bc0ca70381c0
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> |
```

Figura 4.14: Función de login y visualización de cuenta

Transacciones

Es la opción principal para trabajar con los bloques de la cadena. Trata de asemejarse a transacciones de un banco por lo que consiste en que una cuenta transfiera monedas a otra. Desde un cliente se solicitará la cantidad de monedas a transferir y la clave pública de la cuenta (se puede buscar desde la opción de visibilización del banco) a la que queremos transferir. En caso de que se tenga monedas suficientes y la clave corresponda a una cuenta se hará una transferencia de las monedas. Para ello se incluye un nuevo bloque en la blockchain indicando la cuenta de origen, destino e importe de la transferencia.

The image shows two terminal windows side-by-side. The left window, titled '1. blockchain-server', displays a user interface for a blockchain application. It lists several options: 1. View Blockchain state, 2. View your account state, 3. New transaction, 4. View bank state, 5. Close connection, and a help message > 4. Below this, it shows the current bank state with two entries: 'User Test 1 100' and 'User Test 2 100'. The right window, titled '2. blockchain-server', shows the command-line interface for the blockchain server. It starts with 'Starting the node setting...', followed by 'Set successfully!', then 'Options: 1. Log in, 2. Create account, 3. Close connection, > 2'. It then prompts for a new name ('Type your new name:'), showing the input '> User Test 2'. It then says 'The account has been created!' and shows the updated account information: 'Name: User Test 2', 'Coins: 100', 'Public Key: d3150b0037f5c2d84b1a07b9af39680c46a16b63b1a54fa8ec2293872c24242b', and 'Private Key: 57c2f6e721adef7b92b08665520c44727e5f33a0a05bb32d285a139706389faf'. The process is repeated for another user test account.

Figura 4.15: Transferencia de User Test 1 a User Test 2

En la imagen inmediatamente superior se puede ver como *User Test 1* realiza una transferencia de 2 monedas a *User Test 2* desde el cliente de la izquierda y las cuentas quedan actualizadas en el cliente de la derecha, demostrando así que se ha transmitido la transferencia por medio de la red.

4.4.5. Instalación y despliegue

Como requisitos previos para la instalación se asume que se tiene instalado Go⁸, que se tiene construido su espacio de trabajo (directorio con directorios *src* y *bin*)⁹ y que el directorio *bin* del espacio de trabajo está añadido a la variable *\$PATH* del sistema (dependiendo de cada sistema operativo) con el fin de hacer accesibles desde cualquier lugar del sistema los ejecutables de Go.

Dentro de la carpeta *src* se clona el repositorio:

```
git clone https://github.com/daniOrtiz11/blockchain-serverless
```

⁸<https://golang.org/doc/install>

⁹<https://golang.org/doc/code.html#Workspaces>

Una vez incluido el código fuente faltan las dependencias que se incluyen con los siguientes comandos:

```
go get github.com/libp2p/go-libp2p  
go get github.com/davecgh/go-spew/spew  
go get github.com/aws/aws-sdk-go
```

Una vez incluidas las dependencias dinámicas faltan incluir las estáticas de la librería LibP2P. Se ha decidido incluir librerías estáticas ya que esta librería se encuentra actualmente en actualización de manera que se ha querido asegurar el correcto funcionamiento en el proyecto.

Dentro del repositorio se encuentra el directorio *gx-dependencies*. En él se pueden encontrar dos zip: *gx.zip* y *go-libp2p.zip*.

El primero se debe descomprimir y mover a la altura de los directorios *src* y *bin* del espacio de trabajo de Go:

```
cp -r $GOPATH/src/blockchain-serverless/gx-dependencies/gx $GOPATH/
```

Respecto al segundo también se debe descomprimir y el directorio de *go-libp2p* debe reemplazar al existente en las dependencias:

```
rm -r $GOPATH/src/github.com/libp2p/go-libp2p  
cp -r $GOPATH/src/blockchain-serverless/gx-dependencies/go-libp2p  
$GOPATH/src/github.com/libp2p/go-libp2p
```

Después de las operaciones el directorio debe quedar:

Nombre		Fecha de modificación	Tamaño	Clase
bin		hoy 20:18	--	Carpeta
pkg		1 mar 2019 11:37	--	Carpeta
src		hoy 23:50	--	Carpeta
blockchain-serverless		hoy 20:50	--	Carpeta
actions.go		12 may 2019 12:33	7 KB	Documento, .io Code
aws.go		15 may 2019 16:20	6 KB	Documento, .io Code
bank.json		5 may 2019 13:14	433 bytes	JSON
blockchain.go		hoy 20:50	--	Carpeta
blk.json		5 may 2019 13:14	3 KB	JSON
constants.go		15 may 2019 12:50	4 KB	Documento, .io Code
credentials.go		15 may 2019 16:21	924 bytes	Documento, .io Code
debug		5 may 2019 13:14	20,7 MB	Ejecutable Unix
gx-dependencies		hoy 21:20	--	Carpeta
nodejserver.go		hoy 20:50	5 KB	Documento, .io Code
tombola		16 may 2019 16:59	--	Carpeta
LICENSE		10 feb 2019 12:07	1 KB	Documento, extExt
main.go		hoy 20:30	4 KB	Documento, .io Code
obfuscatedcredentials.go		15 may 2019 16:33	501 bytes	Documento, .io Code
README.md		5 may 2019 20:51	308 bytes	Markdown
SS		15 may 2019 16:57	--	Carpeta
utils.go		15 may 2019 16:59	3 KB	Documento, .io Code
github.com		hoy 21:39	--	Carpeta
galang.org		11 feb 2019 0:33	--	Carpeta
gx		hoy 19:44	--	Carpeta
ipfs		3 mar 2019 19:30	--	Carpeta

Figura 4.16: Ejemplo de directorio de trabajo de Go

Una vez acabadas de incluir las dependencias se instala el cliente:

```
go install blockchain-serverless
```

Se puede encontrar dentro del directorio *bin* del espacio de trabajo de Go y en cualquier lugar del sistema en caso de que se haya añadido el directorio al \$PATH como se indicaba al comienzo del apartado.

4.5. Salidas

Como se ha comentado en los apartados anteriores se pueden obtener salidas de dos formas distintas: Postman y Gource. Vamos a ver como se producen ambas en a continuación.

4.5.1. Postman

En el apartado referente a **Amazon S3** se ha afirmado a que el bucket **blcserverlessbucket** referente a las salidas estará público para aceptar peticiones. Se recuerda que solo está público para operaciones de acceso a archivos, no se podrá borrar o insertar ninguno nuevo.

Una vez que se sabe que el bucket está público podemos acceder a él y a su lista de archivos. Para ello se utiliza la herramienta Postman que previamente se ha explicado.

Desde Postman resulta muy sencillo realizar peticiones HTTP contra servicios como el REST API de S3 de AWS. Desde la documentación de AWS ¹⁰ se puede comprobar que operaciones están permitidas. Podemos encontrar las más típicas y ya que queremos acceder a los objetos se usa la operación GET.

Resulta tan sencillo como crearse una colección y posteriormente una nueva petición (dentro de la colección) con los nombres que se quiera.

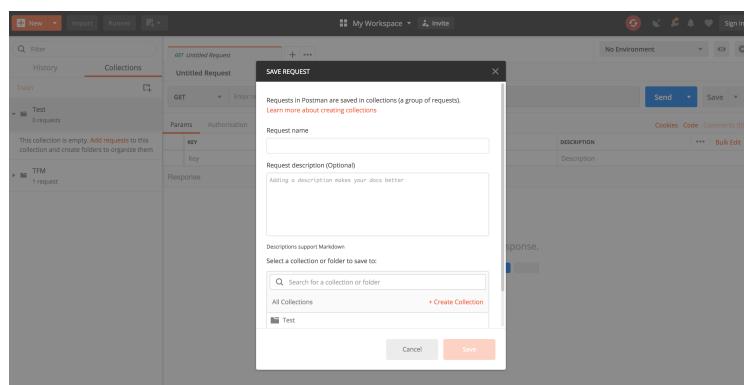


Figura 4.17: Creación de petición en Postman

¹⁰<https://docs.aws.amazon.com/AmazonS3/latest/API>Welcome.html>

Una vez creada la petición se tiene que seleccionar la operación GET y apuntar mediante la url apropiada al repositorio público.

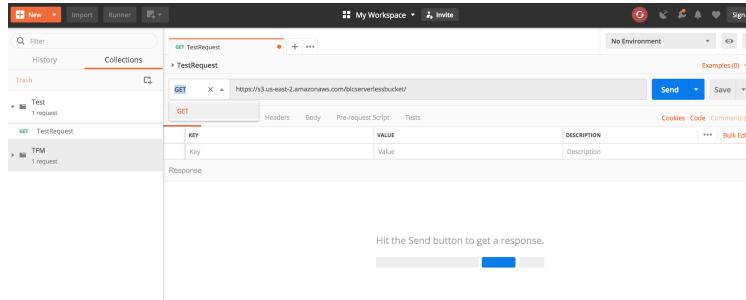


Figura 4.18: Configuración de petición en Postman

Es importante recalcar que la herramienta Postman posee multitudes de configuraciones para realizar distintas llamadas. Al estar el repositorio público y solo para operaciones de obtención de objetos hace que coincida que justo las operaciones permitidas sean las más sencillas de configurar.

Volviendo a la petición que se está viendo, con lanzar la petición pulsando en el botón *SEND* ya obtenemos los resultados esperados.

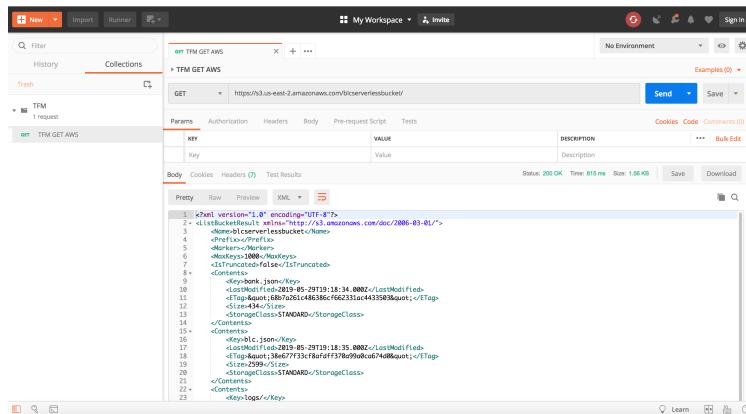


Figura 4.19: Petición de Get satisfactoria en Postman

Se puede observar como el *Status* de la petición ha sido fijado al resultado 200 OK. Lo que significa que la consulta ha devuelto el código de respuesta 200 de petición satisfactoria. También se pueden observar valores como el tiempo que ha tardado o el tamaño de la

respuesta. Además en la mitad inferior podemos observar tanto el cuerpo como las cabeceras de la respuesta. En las cabeceras se puede encontrar desde el servidor de origen (AmazonS3 en este caso) como el formato del contenido de la respuesta. En este caso como no se está accediendo a un archivo en concreto si no que se ha apuntado al bucket en general la respuesta viene en formato XML ¹¹. Observando el cuerpo de la respuesta podemos observar como comienza a verse la lista de archivos como meta datos relativos a estos.

Para acceder a un objeto tan solo hay que añadir a la url su nombre y ya lo obtendríamos.

```

1 [ {
2   "Index": 0,
3   "Timestamp": "2019-05-29 21:16:56-138174 +0200 CEST m+=0.023904856",
4   "Type": "File",
5   "SourceID": "sourceGenesis",
6   "TargetID": "sourceGenesis",
7   "Amount": 1,
8   "Hash": "6e3489cff37a989c544e6b780a2c78901a5fb33738768511a36517ef0d1d",
9   "Preshash": "*",
10 },
11 ],
12 },
13 ],
14 ],
15 ],
16 ],
17 ],
18 ],
19 ],
20 ],
21 ],
22 ],
23 ]
  
```

Figura 4.20: Petición de objeto Get satisfactoria en Postman

En este caso ya se puede observar como el cuerpo de la respuesta ha devuelto el contenido del archivo y en formato JSON tal como se quería.

En siguientes apartados se podrán ver más ejemplos para recuperar salidas de la aplicación utilizando la herramienta de Postman.

4.5.2. Gource

Respecto a Gource se ha afirmado en apartados superiores que se creaba un archivo de log con el formato adecuado a la herramienta. Para acceder a este archivo podemos usar la herramienta Postman, similar a como se usa para acceder a los archivos JSON.

¹¹<https://www.w3.org/TR/xml/>

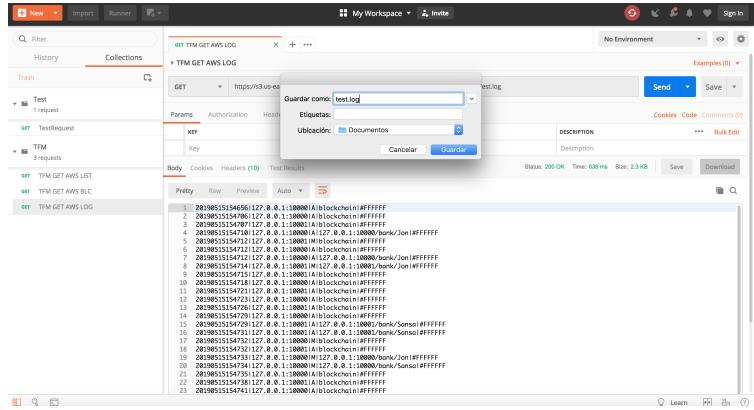


Figura 4.21: Petición y guardado de objeto de log en Postman

En este caso una vez hecha la petición debemos guardar el resultado, botón *Download*, con la extensión .log.

Una vez obtenido el archivo ya se puede proceder a utilizarlo con la herramienta Gource.

Como se dijo en apartados superiores la herramienta Gource parte de un archivo de log (típicamente de un repositorio aunque aquí no es el caso) para que se pueda visualizar el histórico de movimientos que se ha producido. Una vez instalado correctamente Gource y añadido a nuestro PATH el comando necesario para lanzar Gource es el siguiente:

```
gource test.log --title "TFM Blockchain Serverless" --realtime -e 1
--highlight-users --highlight-colour FFFFFF --hide date
```

Los parámetros añadidos en el comando hacen referencia a la velocidad de reproducción y al aspecto de la visualización. Nada más ejecutarlo si el formato del archivo log ha sido el correcto se lanzará una pantalla como esta:

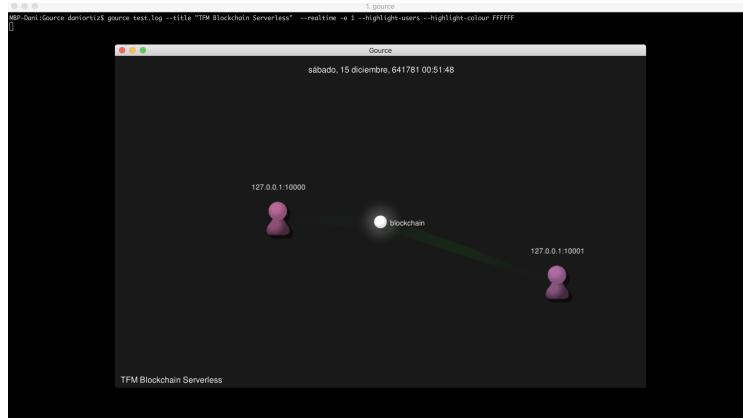


Figura 4.22: Ejecución de Gource

Gource se ejecuta en forma de vídeo recorriendo los tiempos de sus entradas de logs.

Mediante la modificación de los parámetros se puede conseguir visualizaciones distintas pero siempre se verán los mismos elementos.

Se pueden ver a "usuarios" moviéndose alrededor de objetos que están creando/editando.

Estos usuarios interactúan con los objetos mediante flashes de luces de distintos colores. Si la luz es de color verde se indica que el objeto se está creando mientras que si es de color naranja se indica que se está modificando.

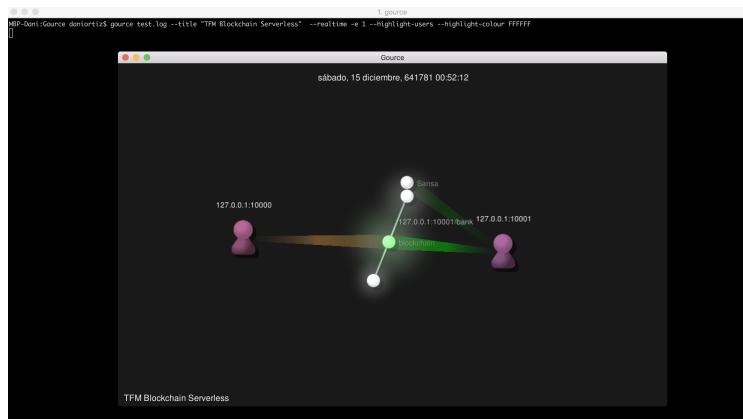


Figura 4.23: Ejecución de Gource con más objetos

Además, los objetos permiten tener sub-objetos lo que beneficia que se puedan mostrar distintas copias de los objetos (una por usuario, replicando el funcionamiento de la

blockchain). En los apartados posteriores se podrán ver más ejecuciones con distintos parámetros y resultados.

4.6. Alcance y límites

En este apartado se va a explicar los límites técnicos de la solución propuesta, hasta donde se ha llegado, que es capaz de hacer y que no el proyecto.

Se tomó la decisión de centrar la solución en conseguir el objetivo fundamental del proyecto: desarrollar una solución blockchain utilizando servicios de computación en la nube. Por ello se optó por desarrollar una solución blockchain propia. Esto aporta beneficios ya que facilita el entendimiento de la tecnología blockchain y facilita la inclusión de tecnología serverless en cualquiera de sus fases. Por contra también desventajas como la pérdida de fiabilidad frente a una solución ya existente o el tiempo o la dificultad del desarrollo. Por esta razón se decidió no incluir en la solución algoritmos de minado. A pesar de haber investigado sobre ellos como se ve en el Estado del Arte, al tratarse el proyecto de una prueba de concepto se decidió que no era necesario para demostrar el resultado esperado.

A su vez por el mismo motivo se decidió solo manejar y permitir direcciones ip locales y privadas en los nodos. Se entiende que la configuración de redes queda fuera del ámbito de la solución, por eso se decidió centrar los esfuerzos contando con redes locales y privadas.

Sin embargo, se entiende que ambas restricciones (redes y minado) son útiles y complementarias a la solución existente por eso van incluidas en el apartado de Trabajo Futuro.

En el capítulo de Objetivos se hace referencia a querer mantener la solución dentro de la capa gratuita de AWS. Se puede afirmar que se ha conseguido, pero con matices.

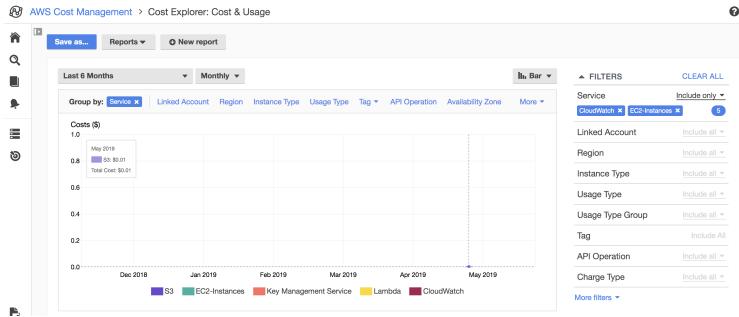


Figura 4.24: Resumen de gastos por meses desde Amazon Cost Reports

Como se observa salvo el último mes, que se realiza un gasto de 0.01\$, nunca se sobrepasa el límite de la capa gratuita. Se puede ver que el servicio culpable de gasto es S3.

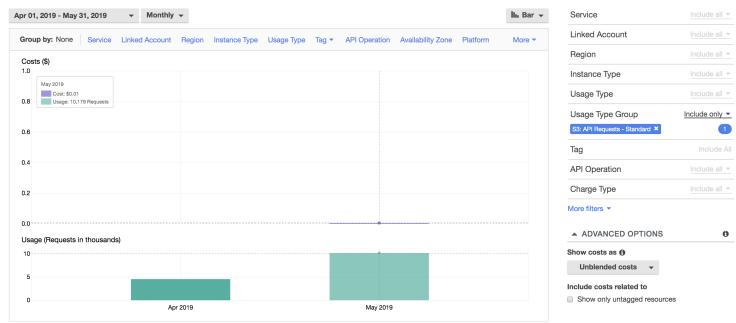


Figura 4.25: Gasto por servicio en los 2 últimos meses

Viendo el uso de los dos últimos meses puede comprobar un aumento importante en el uso respecto a los últimos meses. Además se puede identificar que el apartado de **API: Request** es el que recibe el aumento de uso. Si se desglosa este apartado se puede ver aún con un filtro más específico:

‣ Data Transfer	\$0.00
‣ Elastic Compute Cloud	\$0.00
‣ Key Management Service	\$0.00
‣ Lambda	\$0.00
‣ US East (Ohio)	\$0.00
AWS Lambda USE2-Lambda-GB-Second	\$0.00
AWS Lambda - Compute Free Tier - 400,000 GB-Seconds - US East (Ohio)	140,588 Lambda-GB-Second
AWS Lambda USE2-Request	\$0.00
AWS Lambda - Requests Free Tier - 1,000,000 Requests - US East (Ohio)	4,011 Request
‣ Simple Notification Service	\$0.00
‣ Simple Storage Service	\$0.01
‣ US East (N. Virginia)	\$0.00
‣ US East (Ohio)	\$0.01
Amazon Simple Storage Service USE2-Requests-Tier1	\$0.01
\$0.00 per request - PUT, COPY, POST, or LIST requests under the monthly global free tier	1,798 Requests
\$0.005 per 1,000 PUT, COPY, POST, or LIST requests	1,028 Requests
Amazon Simple Storage Service USE2-Requests-Tier2	\$0.00
\$0.00 per request - GET and all other requests under the monthly global free tier	6,608 Requests
Amazon Simple Storage Service USE2-TimedStorage-ByteHrs	\$0.00
\$0.000 per GB - storage under the monthly global free tier	0.002 GB-Mo
	\$0.00

Figura 4.26: Gasto de S3 y Lambda desglosado

Ahora ya se puede averiguar que el gasto proviene de peticiones desde la API de S3 que son de operaciones de modificación/inserción de objetos. Por tanto, se puede concluir que proviene de las operaciones realizadas desde las funciones Lambda a S3 o desde el cliente a S3.

Si se observa la estructura de la solución se puede llegar a concluir que la operación que más frecuentemente realiza invocaciones es la de gestión de log (desde su función Lambda). Esta operación actualiza su archivo en el bucket de S3 cada vez que se produce una entrada de log nueva. Si tenemos en cuenta que cada vez que los nodos comprueban si están actualizados se produce una salida de log estamos hablando de una nueva entrada cada 5 segundos por nodo. Lo que hace que se sobrepase con relativa facilidad el límite de la capa gratuita en ese apartado. Dado que desde el mes de abril la funcionalidad de log ha estado operativa se puede entender de donde procede el gasto.

Ahora bien, si excluimos la funcionalidad de logs (no necesaria imprescindiblemente para el funcionamiento de la solución, solo para su visualización) si que se ha cumplido el objetivo de no sobrepasarse de la capa gratuita. Hay que tener en cuenta que el uso que se ha dado a la solución no es un uso de entorno de producción si no de prueba de concepto, lo que conlleva demostraciones, casos de usos y todas las fases de desarrollo. En caso de querer realizar un uso intensivo de la solución se tendría que realizar un plan de negocio para aproximarse al coste real. Se puede encontrar un resumen de los precios de AWS aquí.

Capítulo 5

Ejemplos y casos de prueba

Para ambos ejemplos se asume que la aplicación está instalada completamente y se tienen los programas y el cliente disponibles.

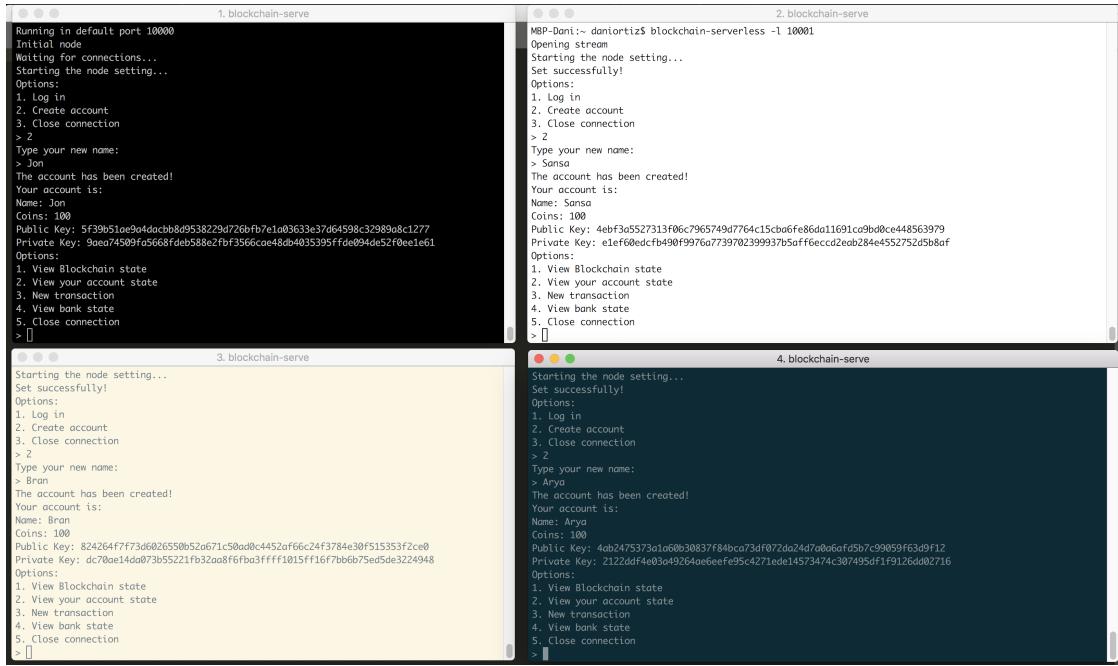
5.1. Servidor local

5.1.1. Descripción

En este primer ejemplo se lanza la aplicación utilizando una misma máquina por lo que cada nodo estará corriendo en la misma dirección pero cada uno en un puerto distinto. Se muestran cuatro consolas distintas (cada una un color) haciendo referencia cada una a un nodo. En cada nodo se creará una cuenta. Se realizan dos transferencias: desde el nodo 1 al nodo 4 y desde el nodo 2 al nodo 3. Posteriormente se volverá a mostrar el estado de las cuentas para comprobar sus efectos. A continuación el nodo 3 se desconectará, lo que hará que el nodo 4 (que estaba conectado al nodo 3) se quede descolgado y tenga que reconectarse a la red. Una vez reconectado se hará dos transferencias a los nodos 1 y 2.

5.1.2. Ejecución

El estado de la red una vez creadas cuatro cuentas, una en cada nodo es el siguiente:



The image shows four terminal windows labeled 1, 2, 3, and 4, each representing a blockchain node. Node 1 (top left) shows the creation of an account named 'Jon' with coins 100. Node 2 (top right) shows the creation of an account named 'Sansa' with coins 100. Node 3 (bottom left) shows the creation of an account named 'Bran' with coins 100. Node 4 (bottom right) shows the creation of an account named 'Arya' with coins 100. Each window displays the public and private keys for the newly created accounts.

```
1. blockchain-server
Running in default port 10000
Initial node
Waiting for connections...
Starting the node setting...
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> Jon
The account has been created!
Your account is:
Name: Jon
Coins: 100
Public Key: 5f39b51ee904dcbb8d953829d72bfb7e1a03633e37d64598c32989a81277
Private Key: 9ea074509f0568fdeb588e2fbf3566cae48db4035395ffde094de52f0e1e61
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []

2. blockchain-server
Starting the node setting...
Set successfully!
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> Sansa
The account has been created!
Your account is:
Name: Sansa
Coins: 100
Public Key: 4ebf3d5527313f8c7965749d7764c15cb06fe86dd11691ca9bd0cc448563979
Private Key: e1ef08edcfb490f99f76a7739702399937b5af6ecc02edb284e4552752d5b8af
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []

3. blockchain-server
Starting the node setting...
Set successfully!
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> Bran
The account has been created!
Your account is:
Name: Bran
Coins: 100
Public Key: 824264f7f73d6026550b52a671c50ad80c4452af66c24f3784e30f515353f2ce0
Private Key: dc70ea14dd073b55221fb32aa8f6fba3ffff1015f16f7bb6b75ed5de3224948
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []

4. blockchain-server
Starting the node setting...
Set successfully!
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> Arya
The account has been created!
Your account is:
Name: Arya
Coins: 100
public Key: 4ab2475373a1a6b30837f84bca73df072da24d7a0e6af5b7c99059f63d9f12
private Key: 2122ddff4e03a49264a66effe95c4271nde14573474c307495df1f9126dd02716
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []
```

Se realizan las transferencias desde nodo 1 al nodo 4 y desde el nodo 2 al nodo 3 y el estado de las cuentas es el siguiente:

```

1. blockchain-server
Type the user id:
> 4ab2475373a1a60b30837f84bca73df072da2d7a0a6fd5b7c99059f63d9f12
Sending the transfer to Arya
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 2
Current account state:
Name: Jon
Coins: 95
Public Key: 5f39b51a0e9a4dcbb8d9538229d726fb7e1a93633e37d64598c32989a8c1277
Private Key: b2cede154679bafcfa3ce22c1391b74f83367433860ed59731df9edccb6f02db6

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []

2. blockchain-server
Type the user id:
> 824264f7f73d6026550b52a671c50a0d0c4452af66c24f3784e30f515353f2ce0
Sending the transfer to Bran
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 2
Current account state:
Name: Sansa
Coins: 92
Public Key: 4ebf3ad527313f06c7965749d7764c15cb06fe86d0a11691c09b0ce448563979
Private Key: 9dd132e6748c906c1d304f026d90005c11e5643bb1e61630e617c0013432a1

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []

3. blockchain-server
Public Key: 824264f7f73d6026550b52a671c50a0d0c4452af66c24f3784e30f515353f2ce0
Private Key: 0a7da81d3a15f6a3983b7593f9adob307e965f18dd61768805ab1d252e35fab5
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 2
Current account state:
Name: Bran
Coins: 108
Public Key: 824264f7f73d6026550b52a671c50a0d0c4452af66c24f3784e30f515353f2ce0
Private Key: 0a7da81d3a15f6a3983b7593f9adob307e965f18dd61768805ab1d252e35fab5

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []

4. blockchain-server
Public Key: 4ab2475373a1a60b30837f84bca73df072da2d7a0a6fd5b7c99059f63d9f12
Private Key: 6291a06d1a7b3d2ab9e1b68695bbfe684f0b5195a8d7552d5d3f7ece5a10d
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 2
Current account state:
Name: Arya
Coins: 105
Public Key: 6291a06d1a7b3d2ab9e1b68695bbfe684f0b5195a8d7552d5d3f7ece5a10d
Private Key: 6291a06d1a7b3d2ab9e1b68695bbfe684f0b5195a8d7552d5d3f7ece5a10d

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []

```

Con esto se puede comprobar que la red está funcionando correctamente. A continuación el nodo 3 (correspondiente a la cuenta de Bran) se desconecta y se intenta realizar una operación desde el nodo 4 (Arya):

```

1. blockchain-server
Type the user id:
> 4ab2475373a1a60b30837f84bca73df072da2d7a0a6fd5b7c99059f63d9f12
Sending the transfer to Arya
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 2
Current account state:
Name: Jon
Coins: 95
Public Key: 5f39b51a0e9a4dcbb8d9538229d726fb7e1a936333e37d64598c32989a8c1277
Private Key: b2cede154679bafcfa3ce22c1391b74f83367433860ed59731df9edccb6f02db6

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []

2. blockchain-server
Type the user id:
> 824264f7f73d6026550b52a671c50a0d0c4452af66c24f3784e30f515353f2ce0
Sending the transfer to Bran
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 2
Current account state:
Name: Sansa
Coins: 92
Public Key: 4ebf3ad527313f06c7965749d7764c15cb06fe86d0a11691c09b0ce448563979
Private Key: 9dd132e6748c906c1d304f026d90005c11e5643bb1e61630e617c0013432a1

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []

3. blockchain-server
Public Key: 824264f7f73d6026550b52a671c50a0d0c4452af66c24f3784e30f515353f2ce0
Private Key: 0a7da81d3a15f6a3983b7593f9adob307e965f18dd61768805ab1d252e35fab5
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 2
Current account state:
Name: Bran
Coins: 108
Public Key: 824264f7f73d6026550b52a671c50a0d0c4452af66c24f3784e30f515353f2ce0
Private Key: 0a7da81d3a15f6a3983b7593f9adob307e965f18dd61768805ab1d252e35fab5

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 2
Your node is currently disconnected
Reconnecting...
Opening stream
Starting the node setting...
Set successfully!
Options:
1. Log in
2. Create account
3. Close connection
> 1
Enter your private key:
> 6291a06d1a7b3d2ab9e1b68695bbfe684f0b5195a8d7552d5d3f7ece5a10d
Welcome Arya!
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> []

```

El nodo 4 ha detectado que está en un estado inestable, se ha vuelto a conectar a la red y vuelve a iniciar la conexión automáticamente. Una vez ahí el usuario inicia sesión con su clave privada. Por último se realizan la últimas transferencias desde el nodo 4 al nodo 1 y 2:

```

1. blockchain-server
2. Sansa 97 4ebf3a5527313f06c7965749d7764c15cba6fe86dd11691ca9bd0ce448563979
3. Bron 108 824264f7f73d6026550b52a671c50ad0c4452af66c24f3784e30f515353f2ce0
4. Arya 97 4ab2475373a1a60030837f84bc73df0726024d7a0aaf5db7c99059f63d9f12

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 4
Current bank state:
1. Jon 98 5f39b51ea94dcbb8d9538229d726bf7e1a03633e37d4598c32989a0c1277
2. Sansa 97 4ebf3a5527313f06c7965749d7764c15cba6fe86dd11691ca9bd0ce448563979
3. Bron 108 824264f7f73d6026550b52a671c50ad0c4452af66c24f3784e30f515353f2ce0
4. Arya 97 4ab2475373a1a60030837f84bc73df0726024d7a0aaf5db7c99059f63d9f12

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 1
Current account state:
Name: Sansa
Coins: 97
Public Key: 4ebf3a5527313f06c7965749d7764c15cba6fe86dd11691ca9bd0ce448563979
Private Key: 9dd1132e6748c906c1d304f026e90005c11c5643bb1e6130e617c0013432a1

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 1
Current account state:
Name: Sansa
Coins: 97
Public Key: 4ebf3a5527313f06c7965749d7764c15cba6fe86dd11691ca9bd0ce448563979
Private Key: 9dd1132e6748c906c1d304f026e90005c11c5643bb1e6130e617c0013432a1

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 1
Current account state:
Name: Sansa
Coins: 97
Public Key: 4ebf3a5527313f06c7965749d7764c15cba6fe86dd11691ca9bd0ce448563979
Private Key: 6291a066df1a7b30d2a9e1b68695b0f6e684f0b5195a0d7752d53f7ece5a10d

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 3
Enter the number of coins to transfer:
> 5
Type the user id:
4ebf3a5527313f06c7965749d7764c15cba6fe86dd11691ca9bd0ce448563979
Sending the transfer to Sansa

Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 1

```

Se puede comprobar que el estado de las cuentas final es el esperado y que la reconexión del nodo 4 ha sido correcta, por tanto se puede dar el ejemplo como satisfactorio.

5.2. Red local

5.2.1. Descripción

En este segundo ejemplo se lanza la aplicación utilizando tres máquinas virtuales que estarán conectadas a la misma red. Cada máquina contiene un nodo. El nodo inicial es mv1, mv2 se conecta a mv1 y mv3 se conecta a mv2.

Se realizan dos transferencias: desde la mv1 a la mv2 y desde la mv3 a la mv2. Después de finalizar todas las sesiones, desde la máquina local y mediante Postman se recuperan los archivos de salida *blc.json*, *bank.json* y *blockchain-serverless.log*. Éste último se descarga y se ejecuta con Gource.

5.2.2. Ejecución

Una vez creadas las cuentas del banco en los extremos (mv1 y mv3) por lo que la red está funcionando correctamente:

```

mv1 [Corriendo]
osboxes@osboxes:~/go/bin$ ./blockchain-serverless
Running in default port 10000
Initial node
Waiting for connections...
Starting the node setting...
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> nodemv1
The account has been created!
Your account is:
Name: nodemv1
Coins: 100
Public Key: 87a0f0d416683dda527a84d0183eb3056a9eee442b554499b00
55f932ada25d0
Private Key: a5220ec80f47184290c8d0f368cd648580e42f1fcce9d3e
0f8065310376
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 4
Current bank state:
1. nodemv1 100 87a0f0d416683dda527a84d0183eb3056a9eee442b554499
b005f932ada25d0
2. nodemv2 100 ab6acd8f1851a28f0ad16528258ada77aaf04ab342b4840e
3fa4080333fb0bf
3. nodemv3 100 b09c32086b319a2343dbeff1d5501f5ed0c864c5b9cdde2
9c3bfff884969f5b
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 1

```



```

mv2 [Corriendo]
osboxes@osboxes:~/go/bin$ ./blockchain-serverless
Running in default port 10000
Initial node
Waiting for connections...
Starting the node setting...
Set successfully!
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> nodemv2
The account has been created!
Your account is:
Name: nodemv2
Coins: 100
Public Key: ab6acd8f1851a28f0ad16528258ada77aaf04ab342b4840e3f
9cf4a9f6426784fb5ad15f44b2770fb9d234f290f9a9564e
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 4
Current bank state:
1. nodemv1 100 87a0f0d416683dda527a84d0183eb3056a9eee442b554499
b005f932ada25d0
2. nodemv2 100 ab6acd8f1851a28f0ad16528258ada77aaf04ab342b4840e
3fa4080333fb0bf
3. nodemv3 100 b09c32086b319a2343dbeff1d5501f5ed0c864c5b9cdde2
9c3bfff884969f5b
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 1

```



```

mv3 [Corriendo]
osboxes@osboxes:~/go/bin$ ./blockchain-serverless
Running in default port 10000
Initial node
Waiting for connections...
Starting the node setting...
Set successfully!
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> nodemv3
The account has been created!
Your account is:
Name: nodemv3
Coins: 100
Public Key: b09c32086b319a2343dbeff1f1d5501f5ed0c864c5b9cdde2
9cf4a9f6426784fb5ad15f44b2770fb9d234f290f9a9564e
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 4
Current bank state:
1. nodemv1 100 87a0f0d416683dda527a84d0183eb3056a9eee442b554499
b005f932ada25d0
2. nodemv2 100 ab6acd8f1851a28f0ad16528258ada77aaf04ab342b4840e
3fa4080333fb0bf
3. nodemv3 100 b09c32086b319a2343dbeff1d5501f5ed0c864c5b9cdde2
9c3bfff884969f5b
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 1

```

Se realiza la primera transferencia desde mv1 a mv2 y se puede ver como se ha detectado en mv3:

```

mv1 [Corriendo]
osboxes@osboxes:~/go/bin$ ./blockchain-serverless
Running in default port 10000
Initial node
Waiting for connections...
Starting the node setting...
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> nodemv1
The account has been created!
Your account is:
Name: nodemv1
Coins: 100
Public Key: 87a0f0d416683dda527a84d0183eb3056a9eee442b554499b00
55f932ada25d0
Private Key: a5220ec80f47184290c8d0f368cd648580e42f1fcce9d3e
0f8065310376
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 4
Current bank state:
1. nodemv1 100 87a0f0d416683dda527a84d0183eb3056a9eee442b554499
b005f932ada25d0
2. nodemv2 100 ab6acd8f1851a28f0ad16528258ada77aaf04ab342b4840e
3fa4080333fb0bf
3. nodemv3 100 b09c32086b319a2343dbeff1d5501f5ed0c864c5b9cdde2
9c3bfff884969f5b
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 2
Enter the number of coins to transfer:
> 2
Type the user id:
> ab6acd8f1851a28f0ad16528258ada77aaf04ab342b4840e3fa440833fb
0bf
Sending the transfer to nodemv2
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 1

```



```

mv2 [Corriendo]
osboxes@osboxes:~/go/bin$ ./blockchain-serverless
Running in default port 10000
Initial node
Waiting for connections...
Starting the node setting...
Set successfully!
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> nodemv2
The account has been created!
Your account is:
Name: nodemv2
Coins: 100
Public Key: ab6acd8f1851a28f0ad16528258ada77aaf04ab342b4840e3f
9cf4a9f6426784fb5ad15f44b2770fb9d234f290f9a9564e
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 4
Current bank state:
1. nodemv1 100 87a0f0d416683dda527a84d0183eb3056a9eee442b554499
b005f932ada25d0
2. nodemv2 100 ab6acd8f1851a28f0ad16528258ada77aaf04ab342b4840e
3fa4080333fb0bf
3. nodemv3 100 b09c32086b319a2343dbeff1d5501f5ed0c864c5b9cdde2
9c3bfff884969f5b
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 1

```



```

mv3 [Corriendo]
osboxes@osboxes:~/go/bin$ ./blockchain-serverless
Running in default port 10000
Initial node
Waiting for connections...
Starting the node setting...
Set successfully!
Options:
1. Log in
2. Create account
3. Close connection
> 2
Type your new name:
> nodemv3
The account has been created!
Your account is:
Name: nodemv3
Coins: 100
Public Key: b09c32086b319a2343dbeff1f1d5501f5ed0c864c5b9cdde2
9cf4a9f6426784fb5ad15f44b2770fb9d234f290f9a9564e
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 4
Current bank state:
1. nodemv1 98 87a0f0d416683dda527a84d0183eb3056a9eee442b554499
b005f932ada25d0
2. nodemv2 102 ab6acd8f1851a28f0ad16528258ada77aaf04ab342b4840e
3fa4080333fb0bf
3. nodemv3 100 b09c32086b319a2343dbeff1d5501f5ed0c864c5b9cdde2
9c3bfff884969f5b
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 1

```

Y por último se realiza la última transferencia desde mv3 a mv2 y se detecta en mv1:

```

mv1 [Corriendo]
osboxes@osboxes:~/go/bin 6:05 AM
No Environment
Coins: 100
Public Key: b09c3208b319a2343dbbeffd5501f5ed0c864c5b9cd8e2
Private Key: 60ceac14d7c93023d0229fd88d3dfb7d5501f5ac9423f
Options:
1. View Block
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 4
Current bank state:
1. nodemv1 100 87af0d416683dda527a84d0183eb3056a9eee442b554499b0
55f932ada256d
2. nodemv2 100 ab6acd8f1851a28f0ad16528258ada77aa04ab342b4840e
3fa408333fb0bf
3. nodemv3 100 b09c32086b319a2343dbeffid5501f5ed0c864c5b9cdde2
9c30fffc84969f5b
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 4
Enter the number of coins to transfer:
> 2
Type the user id:
> ab6acd8f1851a28f0ad16528258ada77aa04ab342b4840e3fa408333fb0bf
8fd
Sending the transfer to nodemv2
Options:
1. View Blockchain state
2. View your account state
3. New transaction
4. View bank state
5. Close connection
> 4
Current bank state:
1. nodemv1 98 87af0d416683dda527a4d0183eb3056a9eee442b554499b0
0855f932ada256d
2. nodemv2 107 ab6acd8f1851a28f0ad16528258ada77aa04ab342b4840e
3fa408333fb0bf
3. nodemv3 95 b09c32086b319a2343dbeffid5501f5ed0c864c5b9cd8e29
c3bfcfc84969f5b
Options:
1. View Blockchain state
2. View your account state
3. New transaction

```

The terminal windows show the interaction between three nodes (mv1, mv2, mv3) running a Go-based blockchain application. The nodes are connected via Ethernet and are performing various operations like viewing account states, creating new transactions, and sending coin transfers between them.

Se obtienen las salidas desde Postman de los archivos de blockchain, banco y log:

POSTMAN API CALL: TFM GET AWS BLC

Method: GET

URL: https://3.us-east-2.amazonaws.com/bclserverlessbucket/blc.json

Params: Authorization, Headers, Body, Pre-request Script, Tests

Body:

```

1. {
2.   "Index": 0,
3.   "TxHash": "87af0d416683dda527a84d0183eb3056a9eee442b554499b0",
4.   "Timestamp": "2019-06-16 05:59:25.456059719 -0400 EDT m=+0.820256339",
5.   "Transaction": {
6.     "SourceCell": "sourceGenesis",
7.     "TargetCell": "targetGenesis",
8.     "Amount": 100
9.   },
10.   "Hash": "6e3409cfcb37a989c544ed8f7880c278901d2fb33738768511a98617ef081d",
11.   "PrevHash": null
12. },
13. {
14.   "Index": 1,
15.   "TxHash": "2019-06-16 05:59:52.99139818 -0400 EDT m=-27.55590523",
16.   "Timestamp": "2019-06-16 05:59:52.99139818 -0400 EDT m=-27.55590523",
17.   "SourceCell": "87af0d416683dda527a4d0183eb3056a9eee442b554499b05f5932da256d",
18.   "TargetCell": "nodemv1",
19.   "Amount": 100
20. },
21. {
22.   "Index": 2,
23.   "TxHash": "469dc4551cfef7d79641c593f719248b038e3e035562c1ef776c251341",
24.   "Timestamp": "2019-06-16 05:59:52.99139818 -0400 EDT m=+42.05544990055f5932da256d",
25.   "SourceCell": "87af0d416683dda527a4d0183eb3056a9eee442b554499b05f5932da256d",
26.   "TargetCell": "nodemv2",
27.   "Amount": 95
28. }

```

POSTMAN API CALL: TFM GET AWS BLC

Method: GET

URL: https://3.us-east-2.amazonaws.com/bclserverlessbucket/bank.json

Params: Authorization, Headers, Body, Pre-request Script, Tests

Body:

```

1. {
2.   "BankID": "87af0d416683dda527a84d0183eb3056a9eee442b554499b05f5932da256d",
3.   "Owner": "nodemv1",
4.   "Address": "0322bc8f47184298c8bf738bc635380a42711c9334f88653831076",
5.   "Amount": 98
6. },
7. {
8.   "BankID": "ab6acd8f1851a28f0ad16528258ada77aa04ab342b4840e3fa408333fb0bf",
9.   "Owner": "nodemv2",
10.   "Address": "9c30fffc84969f5b",
11.   "Amount": 100
12. },
13. {
14.   "BankID": "b09c32086b319a2343dbeffid5501f5ed0c864c5b9cd8e29c3bfc84969f5b",
15.   "Owner": "nodev3",
16.   "Address": "08ccce14d7c93023d0229fd88d3dfb7d5501f5ac9423f",
17.   "Amount": 95
18. }

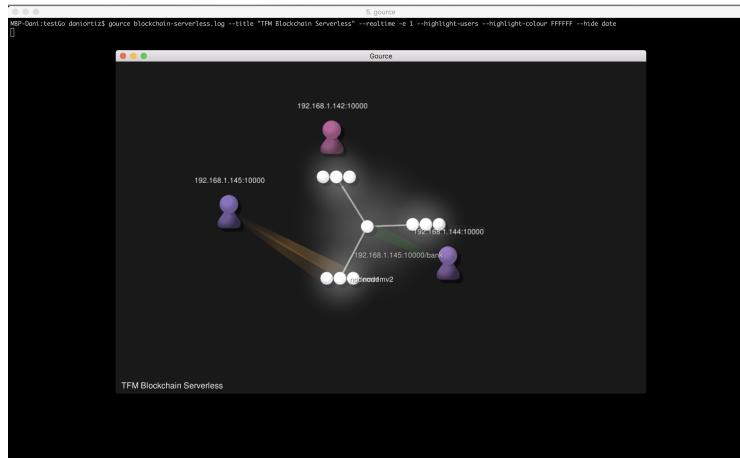
```

```

1 2019061605937192.168.1.145:10000|A|blockchain|#FFFFFF
2 2019061605949192.168.1.145:10000|A|blockchain|#FFFFFF
3 2019061605951192.168.1.145:10000|A|blockchain|#FFFFFF
4 2019061605953192.168.1.145:10000|A|192.168.1.145:10000/bonk/reden1|#FFFFFF
5 2019061605955192.168.1.145:10000|A|blockchain|#FFFFFF
6 2019061605955192.168.1.144:10000|M|blockchain|#FFFFFF
7 2019061605955192.168.1.145:10000|A|192.168.1.145:10000/bonk/reden1|#FFFFFF
8 2019061605957192.168.1.144:10000|M|blockchain|#FFFFFF
9 2019061605957192.168.1.144:10000|A|blockchain|#FFFFFF
10 2019061605959192.168.1.145:10000|M|blockchain|#FFFFFF
11 2019061606009192.168.1.145:10000|M|blockchain|#FFFFFF
12 2019061606009192.168.1.145:10000|M|192.168.1.145:10000/bonk/reden2|#FFFFFF
13 2019061606009192.168.1.145:10000|M|192.168.1.145:10000/bonk/reden1|#FFFFFF
14 2019061606009192.168.1.145:10000|A|blockchain|#FFFFFF
15 2019061606009192.168.1.145:10000|M|blockchain|#FFFFFF
16 2019061606009192.168.1.145:10000|A|blockchain|#FFFFFF
17 20190616060092192.168.1.144:10000|A|blockchain|#FFFFFF
18 20190616060092192.168.1.144:10000|M|blockchain|#FFFFFF
19 201906160600922192.168.1.144:10000|A|blockchain|#FFFFFF

```

El archivo de log se descarga y se procede a ejecutar con Gource para poder visualizar el comportamiento:



Como se puede ver se ven los nodos que han participado, cada uno con su copia de los datos y como interactúan con ellos (las transferencias y las comprobaciones de la blockchain) por lo que se puede dar el ejemplo como satisfactorio.

Capítulo 6

Conclusiones

Se puede concluir que los objetivos que se definieron al comienzo de este proyecto se han cumplido. Se ha cumplido el objetivo principal de combinar la tecnología blockchain con tecnologías serverless manteniendo las virtudes de ambas.

Además, se ha ido más allá y se ha demostrado que se pueden externalizar funcionalidades muy diversas en plataformas de computación sin servidor haciendo uso de servicios complementarios. Se ha conseguido mantener el coste dentro de la capa gratuita de AWS (con excepciones explicadas en la sección de límites) y aprovechar sus servicios de distintas formas. Se ha desarrollado una red blockchain desde cero comprendiendo, una vez estudiadas las existentes soluciones, y aplicando la mayoría de sus puntos, así como desarrollando una aplicación sobre ella sirviendo de ejemplo ilustrativo. También se ha conseguido mostrar de distintas maneras los resultados de la tecnología blockchain (apoyándose en los servicios de AWS nuevamente) tanto como servicio para una aplicación externa como de forma visual.

Por lo tanto, el objetivo principal de fusionar ambas tecnologías en una prueba de concepto se ha completado.

Respecto a los razonamientos posteriores se destaca la relevancia de este TFM en dos puntos fundamentales.

El primero es que gracias a él se puede entender a fondo la tecnología blockchain, en qué consiste realmente y en qué casos puede llegar a ser útil o no. Más allá de la tecnología

elegida para el desarrollo se puede ver que de forma individual sus conceptos no resultan novedosos pero unidos sí que presentan beneficios importantes.

El segundo es que gracias a un gran uso de los servicios en la nube y más concretamente de la computación sin servidor (como las funciones Lambda) se puede concluir que el futuro de los microservicios debe de ir en esa dirección. Sus beneficios son evidentes, ayuda aún más a la tendencia de modular los servicios y su despliegue y mantenimiento resultan sencillo (una vez acostumbrados a su coste por invocación y a sus peculiaridades con el contexto).

Capítulo 7

Trabajo Futuro

Dentro de este proyecto existen varias posibilidades de ampliación. El primer camino consiste en extender el alcance del mismo explicado en el apartado de Alcance y Límites. Otro camino consiste en añadir nueva funcionalidad y características a la solución.

El primer camino se centra en resolver tres cuestiones que quedaron fuera del ámbito de este proyecto: añadir algoritmo de minado, añadir soporte para redes ipv4 globales que funcionen a través de internet y no solo en entornos locales y conseguir implementar la funcionalidad de log de modo que reduzca el número de consultas a AWS.

Respecto a la primera cuestión de implementar el algoritmo de minado es la ampliación más natural. Añadiría un componente de fiabilidad y seguridad a la solución y protegería a la solución frente a usuarios malintencionados. Su desarrollo se debería llevar a cabo en la parte del cliente siguiendo el flujo de la red blockchain, con la posibilidad de realizar las operaciones de minado (que pueden ser costosas) mediante funciones Lambda. Por otra parte, en el soporte a redes ipv4 de internet habría que dar más importancia a la configuración de la red y a conseguir permitir la conexión entre máquinas remotas de este tipo de tráfico. Además, habría que conseguir que desde el cliente (y la librería LibP2P) se obtenga la dirección ip global y no la ip privada. Por último, habría que reformular la manera de obtener el log de la solución de modo que se reduzca notablemente las invocaciones a funciones Lambda y sea más fácil mantenerse en la capa gratuita de AWS.

El segundo bloque de ampliaciones se centra en añadir nuevas características a la solución. La más importante sería conseguir añadir una interfaz con la que poder interactuar con el cliente (ya sea aplicación web o móvil). Mejoraría la usabilidad del producto y no requeriría de conocimientos técnicos para su ejecución. Otra rama de ampliación podría ser obtener los archivos de salida y poder mostrar de forma visual el estado bancario o el estado de la red, igual que la anterior ya sea mediante una aplicación web o móvil o incluso la misma. Independientemente de las ampliaciones anteriores, se podría considerar la inclusión de contratos inteligentes emulando a los que permite Ethereum. La solución es capaz de soportar esta característica sin mucha modificación y añadiría bastante valor añadida a la misma.

Capítulo 8

Introduction

This TFM deals with two new technologies that, a priori, should not have anything in common. On the one hand, we have the blockchain technology. At a time when corruption and transparency are two issues that are present every day, studying a technology that treats them fully becomes very stimulating. On the other hand, the author's career has led him to work building web applications, architectures and microservices. Encouraged by the director of this project, the more he discovered about cloud platforms and possibilities such as lambda functions, the more he believes that the future is to go in that direction. It was necessary to verify if both technologies could be combined and benefit.

The aim of this project is to verify if one can really abstract the functionality of blockchain technology using serverless computing, making it lighter, while maintaining all the intrinsic benefits of its own technology. The realization of the solution has involved depth study of both technologies and multitude of tests and test cases to find a satisfactory solution. In order to demonstrate how both technologies work, the project has been based on an example of banking application with transactions and accounts simulating other existing blockchain platforms such as Bitcoin or Ethereum. The example should always serve as a vehicle to demonstrate in a simpler way how both technologies can be combined.

The proposed solution should be understood as a starting point to advance in this direction and as a proof of concept of how these two technologies can work together, later

it explain the limits and the reason to abstract certain functionalities.

As can be seen in the bibliography, there are many references to both technologies separately, but when both terms are joined together, it has been more difficult to find references.

In the third chapter you can see the state of the art of technologies. A study of the fundamental concepts of blockchain is carried out explaining its functioning and its main platforms. Also it is detailed that means the serverless technology, its different forms and examples of each one of its variants are provided.

Once all the points have been understood, the solution is explained in the fourth chapter. It begins by showing the global architecture (which here takes on great importance), the details of its parts and their connections. They are analyzed individually detailing the technologies used and the reasons that have led to choose them. After it, the limits and scope of the solution, as well as the expected results after testing it.

The fifth chapter shows the test cases carried out and whether they have been satisfactory or not.

And it ends with the sixth and seventh chapters checking whether the proposed objectives have been met, the problems encountered, the conclusions obtained and the future work to be carried out.

Capítulo 9

Conclusions

Overall, it can be concluded that the objectives defined at the beginning of this project have been met. The main objective of combining blockchain technology with serverless technologies while maintaining the virtues of both has been achieved.

In addition, it has gone further and it has been demonstrated that very diverse functionalities can be externalised to serverless computing platforms by making use of complementary services. It has been possible to keep the cost within the free AWS tier (with exceptions explained in the limits section) and take advantage of its services in different ways. A blockchain network has been developed from scratch, once the existing solutions have been studied, and applying most of their points, as well as developing an application on it to serve as an illustrative example. It has also been possible to show in different ways the results of the blockchain technology (relying on the services of AWS again) as a service for an external application and visually.

Therefore, the main objective of combined the two technologies into a proof of concept has been completed.

With respect to the next reasoning, the relevance of this TFM is highlighted in two fundamental points.

The first is that thanks to it the blockchain technology can be understood in depth, what it really consists of and in which cases it can be useful or not. Beyond the technology chosen for the development it can be seen that individually their concepts are not novel

but together they present important benefits.

The second is that, thanks to the use of cloud services and, more specifically, serverless computing (such as Lambda functions), it can be concluded that the future of microservices should move in that direction. Its benefits are obvious, it helps even more to the tendency of modulating the services and its deployment and maintenance are simple (once they are accustomed to its cost per invocation and to its peculiarities with the context).

Bibliografía

- [1] libp2p implementation in go. contribute to libp2p/go-libp2p development by creating an account on GitHub. original-date: 2015-09-30T23:24:32Z.
- [2] Matthias Brantner, Daniela Florescu, David Graf, Donald Kossmann, and Tim Kraska. Building a database on s3. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 251–264. ACM, 2008.
- [3] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. white paper, 2014.
- [4] Christian Cachin. Architecture of the hyperledger blockchain fabric. In Workshop on Distributed Cryptocurrencies and Consensus Ledgers, volume 310, 2016.
- [5] Mike Cantelon, Marc Harter, TJ Holowaychuk, and Nathan Rajlich. Node.js in Action. Manning Greenwich, 2014.
- [6] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, Vignesh Kalyanaraman, et al. Blockchain technology: Beyond bitcoin. Applied Innovation, 2(6-10):71, 2016.
- [7] Alan AA Donovan and Brian W Kernighan. The Go programming language. Addison-Wesley Professional, 2015.
- [8] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. Communications of the ACM, 61(7):95–102, 2018.
- [9] Simson Garfinkel. An evaluation of amazon’s grid computing services: Ec2, s3, and sqs. 2007.

- [10] Anthony D JoSEP, RAnDy KAtz, AnDy KonWinSKi, LEE Gunho, DAViD PAterSon, and ARiEL RABKin. A view of cloud computing. Communications of the ACM, 53(4), 2010.
- [11] SPT Krishnan and Jose L Ugia Gonzalez. Google compute engine. In Building your next big thing with Google cloud platform, pages 53–81. Springer, 2015.
- [12] Alexis Lê-Quôc, Mike Fiedler, and Carlo Cabanilla. The top 5 aws ec2 performance problems. Whitepaper. Datadog Inc, 2013.
- [13] Alexander Lomov. Openshift and cloud foundry paas: High-level overview of features and architectures. white paper, Altoros, 2014.
- [14] Maciej Malawski, Kamil Figiela, Adam Gajek, and Adam Zima. Benchmarking heterogeneous cloud functions. In European Conference on Parallel Processing, pages 415–426. Springer, 2017.
- [15] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
- [16] Guillaume Pierre and Corina Stratan. Conpaas: a platform for hosting elastic cloud applications. IEEE Internet Computing, 16(5):88–92, 2012.
- [17] Marc Pilkington. 11 blockchain technology: principles and applications. Research handbook on digital transformations, 225, 2016.
- [18] Frank Schmager, Nicholas Cameron, and James Noble. Gohotdraw: Evaluating the go programming language with design patterns. In Evaluation and Usability of Programming Languages and Tools, page 10. ACM, 2010.
- [19] Josef Spillner. Snafu: Function-as-a-service (faas) runtime design and implementation. arXiv preprint arXiv:1703.07562, 2017.
- [20] Balaji Varanasi and Sudha Belida. Spring REST. Apress, 2015.

- [21] Jinesh Varia. Best practices in architecting cloud applications in the aws cloud. In Cloud Computing: Principles and Paradigms, volume 18, pages 459–490. Wiley Online Library, 2011.
- [22] Mario Villamizar, Oscar Garces, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, et al. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures. In 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pages 179–182. IEEE, 2016.
- [23] Mario Villamizar, Oscar Garcés, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, et al. Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures. Service Oriented Computing and Applications, 11(2):233–247, 2017.
- [24] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, 151:1–32, 2014.
- [25] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In 2015 IEEE Security and Privacy Workshops, pages 180–184. IEEE, 2015.

Apéndice A

Prueba inicial con Ethereum

Prueba inicial llevada a cabo como parte del proceso de aprendizaje de la tecnología blockchain.

A.1. Blockchain mediante Geth en entorno virtual

A.1.1. Descripción

Escenario inicial de toma de contacto con tecnología blockchain. Consiste en usar el cliente de Ethereum desarrollado en Golang llamado Geth para crear una red blockchain privada en un entorno virtual simulando un posible entorno real de blockchain. Constará de un entorno con 3 máquinas virtuales en las que se comunicarán unas con otras llegándose a configurar los nodos correctamente y a comenzar a minar ether.

A.1.2. Objetivos

- Comprender funcionamiento de blockchain.
- Conseguir configurar la tecnología para crear la red.
- Conseguir configurar y conectar 3 mineros a la red.
- Comprender el modo de funcionamiento de Ethereum.

A.1.3. Requisitos

1. Configuración de tres máquinas virtuales:

- SO usado para mv1: Ubuntu 16.04
- SO usado para mv2: Lubuntu 18.04
- SO usado para mv3: Linux Lite 3.8
- Herramienta para la gestión de paquetes: apt-get
- Conexión entre todas las mvs a la misma red.
 - Actualmente mediante ipv4.
 - Opción red interna.
 - Mv1: 192.168.0.157
 - Mv2: 192.168.0.156
 - Mv3: 192.168.0.159

2. Instalar Geth y dependencias en todos ellos.

- `sudo add-apt-repository -y ppa:ethereum/ethereum`
- `sudo apt-get update`
- `sudo apt-get install ethereum`
- `sudo apt-get install software-properties-common`
- `sudo add-apt-repository -y ppa:ethereum/ethereum`
- `sudo apt-get update`
- `sudo apt-get install ethereum`

A.1.4. Creación del génesis

Se usa la herramienta **puppeth**¹ para la creación del fichero genesis.json (definición del primer bloque de blockchain). Basta con teclear (ejecutar) desde cualquier consola y seguir los siguientes pasos:

```
osboxes@osboxes:~/blockchain/case_geth$ puppeth
```

1. Se identifica a la red con *test_geth_blockchain*.
2. Se elige *proof-of-work* como algoritmo de consenso.
3. Se obvia la pregunta de cuentas prefinanciadas.
4. Network Id: 39090
5. Se exporta el fichero génesis desde puppeth con el nombre de nuestra red. Debe ser copiado al resto de máquinas que se quiera que formen parte de la blockchain.

Si se abre el archivo se puede encontrar al comienzo la configuración elegida:

```
{
  "config": {
    "chainId": 39090,
    "homesteadBlock": 1,
    "eip150Block": 2,
    "eip150Hash": "0x000000...",
    "eip155Block": 3,
    "eip158Block": 3,
    "byzantiumBlock": 4,
    "ethash": {}
  }...
```

Listing A.1: *genesis.json*

A.1.5. Creación de los nodos

El siguiente paso es la creación de los nodos y los bloques iniciales de la cadena.

Ejecutamos en mv1 el comando geth junto con el directorio elegido para los mineros y el fichero génesis del paso anterior:

```
geth --datadir minero init test_geth_blockchain.json
```

¹<https://github.com/puppeth>

Se obtiene la siguiente salida:

```
osboxes@osboxes:~/blockchain/case_geth$ geth --datadir minero init test_geth_blockchain.json
WARN [11-05|17:39:53.453] Sanitizing cache to Go's GC limits provided=1024 updated=328
INFO [11-05|17:39:53.456] Maximum peer count ETH=25 LES=0 total=25
INFO [11-05|17:39:53.456] Allocated cache and file handles
database=/home/osboxes/blockchain/case_geth/minero/geth/chaindata cache=16 handles=16
INFO [11-05|17:39:53.468] Writing custom genesis block
INFO [11-05|17:39:53.481] Persisted trie from memory database nodes=354 size=51.71kB
time=5.325697ms gcnodes=0 gcsiz=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [11-05|17:39:53.481] Successfully wrote genesis state database=chaindata
hash=13f462...bdebb0
INFO [11-05|17:39:53.481] Allocated cache and file handles
database=/home/osboxes/blockchain/case_geth/minero/geth/lightchaindata cache=16 handles=16
INFO [11-05|17:39:53.494] Writing custom genesis block
INFO [11-05|17:39:53.502] Persisted trie from memory database nodes=354 size=51.71kB
time=1.09917ms gcnodes=0 gcsiz=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [11-05|17:39:53.505] Successfully wrote genesis state
database=lightchaindata hash=13f462...bdebb0clear
```

Listing A.2: Salida de consola

Como se puede observar al final de la salida, se indica que se ha creado el bloque inicial correctamente. Se debe repetir este proceso en mv2 y mv3. Aunque no sea necesario para este escenario, se debe crear un monedero para cada nodo para almacenar el ether obtenido. Por tanto, en cada nodo se ejecuta:

```
osboxes@osboxes:~/blockchain/case_geth$ geth --datadir . account new
```

Se introducen las contraseñas requeridas y se guardan para consultar las cuentas más adelante. Las contraseñas son monederomvX(done X es 1,2 o 3 respectivamente). Para comprobar que se han creado las cuentas correctamente:

```
geth --datadir . account list
```

Hay que tener en cuenta en que las redes de Ethereum no son autodescubriles. Por tanto, se debe indicar en cada nodo el resto de nodos que forman la red. Esto se consigue por medio del fichero *static-nodes.json* que contendrá la información de todos los nodos que forman parte de la red. Para arrancar y obtener la información del nodo:

```
geth --identity "miner1" --networkid 39090 --datadir . --nodiscover --rpc
--rpcport "8041" --port "30301" --unlock 0 --rpccapi
"txpool,admin,db,eth,net,web3,miner,personal" --rpccaddr 0.0.0.0 --rpccorsdomain
"*" --cache=4096 --syncmode "fast" --rpccvhosts "*"
```

Listing A.3: Salida de consola

Es posible que se pida la contraseña (la introducida unos pasos anteriores) para desbloquear la cuenta, se proporciona en caso de que se quiera, pero no es necesario desbloquear la cuenta para este paso.

En la salida hay que buscar el apartado del enode y cambiamos 127.0.0.1 por la ip de la máquina:

```
self="enode://0c1d30ac6f4b8db74b56385078cb40dad35da4cdbd7abc408  
6b8e8eeac613a7a1440e804d4343e7c90d15656897b04b77a237b63a121a42  
337abb6acf1729595@192.168.0.157:30301?discport=0"
```

Listing A.4: Salida de consola

Hay que copiar la salida y pegarla en el fichero *static-nodes.json*. Se repite el proceso con mv2 y mv3 (cambiando los parámetros necesarios) para obtener el fichero completo.

Deberá estar en el mismo directorio en el que se ha generado nuestro fichero génesis en cada mv.

A.1.6. Conexión de los nodos

En este punto los nodos ya deberían estar conectados gracias al fichero del paso anterior *static-nodes.json*. Para comprobarlo hay que ver si desde cada uno de ellos se ven los demás. Para ello primero hay que conectarse a la consola de Geth. Por ejemplo, desde mv2:

```
geth attach http://192.168.0.156:8041
```

Proporciona la siguiente salida:

```
instance: Geth/miner2/v1.8.17-stable-8bbe7207/linux-amd64/go1.10.1  
coinbase: 0x0d46a7f5f7d77895f10b4fb5516512c5ee31ac53  
at block: 0 (Wed, 31 Dec 1969 19:00:00 EST)  
datadir: /home/osboxes/blockchain/case_geth  
modules: admin:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0  
web3:1.0  
>
```

Listing A.5: Salida de consola

Se puede ver que proporciona los datos del nodo en el que nos encontramos y nos da el control de la consola. Para comprobar a quién está conectado:

```
> admin.peers
```

El resultado son el resto de los nodos:

```
[{
  caps: ["eth/63"],
  enode:
"enode://0c1d30ac6f4b8db74b56385078cb40dad35da4cdbd7abc4086b8e8eeac613a7a1440e804d4343e7
c90d15656897b04b77a237b63a121a42337abb6acf1729595@192.168.0.157:30301?discport=0",
  id: "0d838e11da0b25030ab0032405d9b7eb0e923a6c3d5c1aab5af76f92775c2c8b",
  name: "Geth/miner1/v1.8.17-stable-8bbe7207/linux-amd64/g01.10",
  network: {
    inbound: false,
    localAddress: "192.168.0.156:54552",
    remoteAddress: "192.168.0.157:30301",
    static: true,
    trusted: false
  },
  protocols: {
    eth: {
      difficulty: 17179869184,
      head: "0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
      version: 63
    }
  }
},
{
  caps: ["eth/63"],
  enode:
"enode://dc1f9d4262d5ea2dc64c30c7f247edca208bb458e007d403d058c62e3ed93595ab09e8abaa9f1e1
c4260377dc6709912c34c1ac743e71154aa3d76e6d4ca764f@192.168.0.159:44756",
  id: "efb99c41fb156c6eeded32ae93949807477afc93f1a22d4551337e40e172a23e6",
  name: "Geth/miner3/v1.8.17-stable-8bbe7207/linux-amd64/g01.10",
  network: {
    inbound: true,
    localAddress: "192.168.0.156:30301",
    remoteAddress: "192.168.0.159:44756",
    static: false,
    trusted: false
  },
  protocols: {
    eth: {
      difficulty: 17179869184,
      head: "0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
      version: 63
    }
  }
}]
```

Listing A.6: Salida de consola

Se puede observar como se han reconocido los otros nodos conectados a la red. Si se procede igual en el resto de nodos se puede comprobar como también se reconocen entre sí. En este punto se da por conseguida la conexión entre nodos de una red privada Ethereum.

A.1.7. Minado de los nodos

Una vez se tienen conectados los nodos se puede proceder a arrancar el minado.

El primer paso es volver a conectar el nodo a la consola de Geth como hemos visto antes.

Una vez ahí:

```
> miner.start(1)
```

El "1" indica el número de hijos que se ejecutarán en el proceso de minado. Se ejecuta la misma instrucción en los nodos que se quiera que sean mineros. Para parar el minero:

```
> miner.stop()
```

Para comprobar la eficacia del minado:

```
> miner.getHashrate()
```

Puede tardar un pequeño intervalo de tiempo desde que se los nodos comienzan a minar hasta que se vea un valor distinto a 0. Para consultar el ether obtenido:

```
> eth.getBalance(eth.coinbase).toNumber();
```

En caso de que se tenga más de una cuenta o no se sepa a que cuenta hace referencia actualmente "coinbase" se puede crear una función Javascript desde la propia consola para obtener el total de ether en todas las cuentas: El resultado son el resto de los nodos:

```
function checkAllBalances() {
    var totalBal = 0;
    for (var acctNum in eth.accounts) {
        var acct = eth.accounts[acctNum];
        var acctBal = web3.fromWei(eth.getBalance(acct), "ether");
        totalBal += parseFloat(acctBal);
        console.log("eth.accounts[" + acctNum + "]: \t" + acct + " \tbalance: "
+ acctBal + " ether");
    }
    console.log("Total balance: " + totalBal + " ether");
};
```

Listing A.7: función checkAllBalances

Se ejecuta la función simplemente llamándola:

```
> checkAllBalances();
eth.accounts[0]: 0xda7921587b0620900526bd3428de1af53e70de56
balance: 0 ether Total balance: 0 ether
```

El resultado, aunque se ha comprobado que los mineros estaban trabajando es aún de 0 ether. Hay que tener en cuenta que en el minado influyen factores como la dificultad de la red (que se incluye en el génesis) o el número de hijos que hemos indicado al lanzarlo o el número de mineros que están al mismo tiempo trabajando. No hay que olvidar que Ethereum usa la técnica de PoW (Proof of Work). Por otra parte, desde la consola donde de los nodos se puede ir viendo las salidas que se van produciendo: actualización de los mineros, generaciones de operaciones etc.