INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO



**Diseño de un prototipo basado en criptomoneda para la detonación de economías locales**

|  |
| --- |
| **TESIS**  **QUE PARA OBTENER EL TÍTULO DE**  **INGENIERO EN COMPUTACIÓN**  P R E S E N T A  JOSÉ FRANCISCO ALTAMIRANO ZEVALLOS |

ASESOR: **Dr. José Alberto Domingo Incera Dieguez**

**CIUDAD DE MÉXICO 2020**

**TABLA DE CONTENIDO**

Contenido

[1. INTRODUCCIÓN 5](#_Toc108528225)

[1.1 Contexto 5](#_Toc108528226)

[1.2 Identificación del problema 6](#_Toc108528227)

[1.3 Objetivo 7](#_Toc108528228)

[1.4 Alcance 7](#_Toc108528229)

[1.5 Metodología 8](#_Toc108528230)

[1.6 Organización del documento 8](#_Toc108528231)

[2. ANÁLISIS 10](#_Toc108528232)

[2.1 Requerimientos 10](#_Toc108528233)

[2.1.1. Funcionales 10](#_Toc108528234)

[2.1.2 No funcionales 10](#_Toc108528235)

[2.2 Restricciones 11](#_Toc108528236)

[2.3 Trabajos relacionados 12](#_Toc108528237)

[3. DISEÑO 13](#_Toc108528238)

[3.1 Arquitectura 13](#_Toc108528239)

[3.1.1 Red 20](#_Toc108528240)

[3.1.2 Base de datos 21](#_Toc108528241)

[3.1.2.1 Base de datos de Blockchain: 21](#_Toc108528242)

[3.1.2.2 Base de datos de estado: 22](#_Toc108528243)

[3.1.3 Proceso de transacción 24](#_Toc108528244)

[3.2 Entorno de trabajo 26](#_Toc108528245)

[3.2.1 Dominios 27](#_Toc108528246)

[3.2.2 Roles 27](#_Toc108528247)

[3.2.3 Activos 30](#_Toc108528248)

[3.3 Soluciones alternativas 30](#_Toc108528249)

[3.3.1 Ethereum 31](#_Toc108528250)

[3.3.2 Hyperledger 31](#_Toc108528251)

[3.4 Estándares y licencias utilizados 32](#_Toc108528252)

[4. IMPLEMENTACIÓN 34](#_Toc108528253)

[4.1 Descripción completa de la implementación 34](#_Toc108528254)

[4.1.1 Instalación y configuración de paquetería y herramientas para la construcción inicial. 34](#_Toc108528255)

[4.1.1.1 Instalación Python. 34](#_Toc108528256)

[4.1.1.2 Instalación git. 35](#_Toc108528257)

[4.1.1.3 Instalación Docker. 35](#_Toc108528258)

[4.1.1.4 Configuración PostgreSQL en Docker. 36](#_Toc108528259)

[4.1.1.5 Configuración C++ Build Tools 36](#_Toc108528260)

[4.1.1.6 Configuración Hyperledger Iroha. 37](#_Toc108528261)

[4.1.2 Instalación y configuración de paquetería y herramientas para la construcción de una red con múltiples nodos. 40](#_Toc108528262)

[4.1.2.1 Preparación previa. 40](#_Toc108528263)

[4.1.2.2 Replicación del prototipo en los demás dispositivos. 40](#_Toc108528264)

[4.1.2.3 Configuración del router. 41](#_Toc108528265)

[4.1.2.4 Configuración del firewall. 41](#_Toc108528266)

[4.1.2.5 Creación de las llaves y modificación del genesis block. 42](#_Toc108528267)

[4.1.3 Instalación y configuración de paquetería y herramientas para la adición de un nuevo nodo a la red de múltiples nodos existente. 43](#_Toc108528268)

[4.1.4 Pausa, inicio y reinicio del ambiente configurado en Docker. 44](#_Toc108528269)

[4.1.5 Configuración del entorno de trabajo. 45](#_Toc108528270)

[4.1.6 Interfaz 45](#_Toc108528271)

[4.2 Especificaciones de software y hardware 45](#_Toc108528272)

[4.2.1 Software 45](#_Toc108528273)

[4.2.1.1 Git 45](#_Toc108528274)

[4.2.1.2 Docker 45](#_Toc108528275)

[4.2.1.3 Hyperledger Iroha 46](#_Toc108528276)

[4.2.1.4 PostgreSQL 46](#_Toc108528277)

[4.2.2 Hardware 46](#_Toc108528278)

[5. PRUEBAS Y RESULTADOS 47](#_Toc108528279)

[5.1 Descripción de las pruebas 47](#_Toc108528280)

[5.2 Resultados 47](#_Toc108528281)

[5.3 Análisis de resultados 47](#_Toc108528282)

[6. CONCLUSIONES 48](#_Toc108528283)

[7. ANEXOS 49](#_Toc108528284)

[7.1 Anexo A: Algoritmos de consenso 49](#_Toc108528285)

[7.2 Anexo B: Versión previa al 12/11/21 52](#_Toc108528286)

[7.3 Anexo C: Genesis block nodo independiente. 54](#_Toc108528287)

[7.3 Anexo D: Genesis block múltiples nodos. 57](#_Toc108528288)

[7.4 Anexo E: Config.docker 64](#_Toc108528289)

[7.5 Anexo F: Ambiente de Python 65](#_Toc108528290)

[REFERENCIAS 75](#_Toc108528291)

# 1. INTRODUCCIÓN

A lo largo de este capítulo, se definirá el contexto en el que se desarrollará el proyecto, el cual se analizará para poder identificar el problema a resolver. Para su resolución, se propondrá un objetivo que estará ligado al problema identificado previamente y que representará la meta final del proyecto, y se definirán la metodología y alcance que tendrá el desarrollo. Finalmente, se presentará la organización que tendrá el documento, junto con una descripción sobre cada segmento.

## 1.1 Contexto

México es el tercer país receptor de remesas a nivel global, siendo que recibe 5.4% de las remesas totales mundiales [1]. Esto significa que las remesas son una actividad comercial vital para México, ya que ocupan el segundo lugar como fuente de ingresos a nivel nacional, superadas únicamente por la industria automotriz [2].

Esto ha ocasionado que la economía mexicana desarrolle una fuerte dependencia de las remesas al integrar el 2.9% del PIB nacional en el 2019 [1]; lo que es el mayor grado de dependencia experimentado desde el 2003 y el mayor históricamente si se excluye el valor del 2003. A nivel estatal, esta dependencia llega a ser mucho más pronunciada, ya que alcanza porcentajes hasta del 11.8%, como en el caso de Michoacán.

Lo anterior demuestra la importancia de las remesas como actividad económica a nivel nacional; sin embargo, las remesas también han tenido un fuerte impacto en el estilo de vida llevado por los individuos emisores y receptores de las remesas.  
Esto se dice ya que 68.5% de los emisores de remesas las envían a personas que son completamente dependientes de estas [3]. Además, el monto recibido por las remesas se utiliza en un 80% en alimentación, vestimenta y salud, siendo los siguientes usos más populares la vivienda, educación y deudas [1].

Actualmente el envío de remesas es realizado en un 83% por casas comerciales, 10.4% por bancos y el porcentaje restante se encuentra dividido entre diversos métodos [1]. Estos envíos son realizados en un 45.4% con una frecuencia mensual, 20.6% quincenal, 9.7% semanal y el porcentaje restante con otras modalidades [3]. Finalmente, el monto promedio del envío es de 325 dólares y se suele cobrar una tarifa de procesamiento promedio de 6.69 dólares [1]. Del lado del receptor, estas son cobradas un 25% el mismo día, 35% al siguiente, 30% dentro de una semana y el 10% restante, en otro plazo [3].

El envío de las remesas está experimentando un gradual cambio hacia la tecnología de las criptomonedas debido a que: ofrecen mayor velocidad, mejor tipo de cambio, costos de transferencias más bajos, evitan muchos problemas de volatilidad y dan liquidez a los bancos al evitarles tener que poseer reservas de las divisas. Actualmente, 6% de las remesas que se envían desde los Estados Unidos de América (EUA) a México utilizan criptomoneda [4], y se espera que este porcentaje vaya en aumento con el tiempo.

## 1.2 Identificación del problema

Por lo previamente dicho, sabemos entonces que la mayoría de la gente que recibe remesas en México es completamente dependiente de estas; lo que provoca que las comunidades en las que las remesas son una fuente importante de ingresos económicos se vuelvan, a su vez, dependientes de las remesas.

El que una comunidad sea dependiente de las remesas es un problema, ya que estas tienen un escaso impacto en la capacidad de crecimiento y desarrollo económico [5]. Esto se debe mayormente a que las remesas son utilizadas en artículos de primera necesidad, lo que permite mejorar el nivel de vida de los receptores, pero no ayuda a resolver los problemas estructurales que perpetúan la pobreza y mantienen a las familias en estado de dependencia.

Entonces, las comunidades dependientes de las remesas son vulnerables, ya que no cuentan con una economía local lo suficientemente fuerte para desarrollarse y dejar de serlo, ni son capaces de mantener el mismo nivel de vida ante una situación problemática con las remesas o ante cualquier otro problema de índole económica.

Existen múltiples programas de apoyo a las comunidades vulnerables en México, los cuales consisten en transferencias económicas, subsidios, cupones, etc. Sin embargo, ninguno de estos busca resolver los problemas estructurales que perpetúan el estado de vulnerabilidad en el que se encuentran las comunidades, sino que implementan soluciones temporales.

Entre los programas que buscan resolver los problemas estructurales se encuentra el uso de las Community Inclusion Currencies (CIC). El CIC es una moneda complementaria a la moneda nacional que tiene como objetivo promover la economía local mediante el establecimiento de un mercado exclusivo a la comunidad. Para lograrlo, el CIC solo es aceptado dentro de la comunidad y no tiene ningún valor ni respaldo fuera de esta; lo que ocasiona que dicha moneda no pueda salir de la comunidad y, por lo tanto, este circulando constantemente dentro de esta al intercambiarla por bienes y/o servicios [6].

Por un lado, la ventaja que tiene el CIC sobre medidas como paquetes de ayuda o transferencias monetarias, es que el CIC estimula la economía local y el dinero fiduciario (dinero Fiat), no. Esto se debe a que el Fiat es intercambiable por cualquier tipo de bien o servicio, el cual no necesariamente es de la comunidad; lo que ocasiona que el dinero salga rápidamente de la comunidad, por lo que solo presenta una solución a corto plazo.

Por el otro, la ventaja que tiene el CIC por encima de los subsidios y cupones, es que estos solo incentivan la compra de ciertos productos, pero no detonan una economía local, lo que también los convierte en una solución a corto plazo.

El uso de CIC se ha implementado exitosamente en varios países, entre ellos en México; pero se ha enfrentado con dificultades para su distribución y reimpresión, lo que ha limitado muy seriamente su efectividad en México.   
Además, es necesario realizar un estudio sobre el estado y necesidades de la comunidad en la que se implementará, por lo que se ha implementado en muy pocas comunidades.

## 1.3 Objetivo

Diseñar un programa de CIC utilizando un prototipo de un Sistema de Ledger Distribuido (DLT, por sus siglas en inglés) que permita reducir la vulnerabilidad de las comunidades mediante el aumento de la productividad de las remesas.

## 1.4 Alcance

El análisis sobre el método de transferencia de las remesas se realizó con un enfoque en las características políticas económicas y sociales de EUA y México, por lo que su alcance geográfico se limita al espacio de los países mencionados.

El análisis sobre la detonación de economías locales se realizó con un enfoque en las características políticas económicas y sociales de las localidades mexicanas, por lo que su alcance geográfico se limita a México.

## 1.5 Metodología

La metodología por seguir para el desarrollo de este proyecto es la metodología de cascada. Se selecciona esta metodología debido a que nos permite desarrollar un proyecto de manera secuencial y constante utilizando un conjunto de pasos claros y definidos para el desarrollo del proyecto. Además, al final de cada paso se realiza una revisión para determinar si el proyecto está listo para avanzar a la siguiente fase o si es necesario ajustarlo antes de continuar.

## 1.6 Organización del documento

El documento contará con los capítulos a continuación mencionados en orden de lectura: Introducción, Análisis del problema, Diseño de la solución, Implementación, Pruebas y Resultados, Conclusiones, y Referencias. Cada uno de estos capítulos representa un paso de la metodología de cascada.

A continuación, se hará una pequeña descripción de que contiene cada capítulo.

En la introducción se realizará una investigación basada en diversos documentos oficiales sobre la migración, remesas, economías locales, casos de estudio sobre las CIC, trabajos de investigación sobre XRP y guías oficiales para la implementación y uso de un nodo de XRP. Esto se verá reflejado en una explicación a grandes rasgos donde se desarrolla este trabajo; la identificación del problema, actores y su contexto; definición del objetivo con base en el problema identificado; definición del alcance del proyecto; y selección de la metodología y estructura del proyecto.

Después, en el análisis del problema se especificarán, sobre el proyecto, los requerimientos funcionales y no funcionales, las restricciones y los trabajos relacionados que ofrezcan soluciones a problemas similares.

Posteriormente, en el diseño de la solución se especificará la arquitectura del proyecto mediante una descripción completa, las soluciones alternativas al problema pero que se desvíen de la aquí propuesta, y los estándares utilizados por el prototipo.

A continuación, el capítulo de implementación hará una descripción completa del hardware, software y todos los pasos que se llevaron a cabo para su correcta implementación.

Seguidamente, en las pruebas y resultados se describirán las pruebas realizadas, se explicará porque se escogieron estas pruebas, se expondrán los resultados y finalmente se explicarán los resultados obtenidos y se definirá si estos son favorables o no al prototipo.

Finalmente, en la conclusión se especificará el funcionamiento de la versión final del prototipo junto con las especificaciones del mantenimiento que se le debe dar que asegure su correcto funcionamiento y compatibilidad con las nuevas versiones.

# 2. ANÁLISIS

En este capítulo se especifican todos los requerimientos con los que debe cumplir el prototipo para alcanzar el objetivo establecido en el capítulo anterior. De la misma manera, se desarrollarán las restricciones que afectarán al diseño de este prototipo. Finalmente, se hará una pequeña presentación sobre los trabajos relacionados a la problemática presentada.

## 2.1 Requerimientos

A continuación, se dividirán los requerimientos en dos partes, por un lado, en aquellos que especifican qué es lo que debe hacer el prototipo, o requerimientos funcionales; y por el otro, cómo es que debe hacer estas cosas el prototipo, o requerimientos no funcionales.

### 2.1.1. Funcionales

La solución debe:

* Permitir el ingreso de nuevos usuarios mediante un formulario de registro.
* Permitir que los usuarios realicen transacciones de CIC entre sí.
* Evaluar las transacciones para determinar si se aprueban o se rechazan.
* Registrar en una base de datos todas las transacciones que se aprueben.
* Permitir consultas a la base de datos.
* Dar la capacidad a todos los usuarios de tener una billetera virtual capaz de almacenar cualquier cantidad de CIC y la capacidad de consultarla
* Retirar un pequeño porcentaje de CIC de todas las cuentas a final de mes para promover el gasto y reducir el costo del programa.

### 2.1.2 No funcionales

La solución debe:

* Implementar un sistema autorizado. Es decir, que cada usuario que quiera unirse a la red debe recibir aprobación del administrador para hacerlo.
* Precisar que el registro de los usuarios debe realizarse utilizando CURP, correo electrónico, contraseña, código postal, estado, alcaldía o municipio, y asentamiento o colonia.
* Identificar a cada usuario mediante un ID único.
* Asegurar la anonimidad de cada usuario frente a otros usuario del sistema.
* Realizar las transacciones de manera rápida, segura y anónima.
* Especificar que las transacciones deben tener un ID, fuente, destino, fecha y valor. La fuente y destino deben ser usuarios registrados.
* Evaluar las transacciones mediante un protocolo de consenso para que los nodos decidan si se aprueban o rechazan.
* Dar una copia de las bases de datos a todos los nodos de validación participantes.
* Prohibir que las bases de datos sean modificables por cualquier usuario, estas solo deben ser alterables por los nodos de validación.
* Tener un porcentaje de disponibilidad del 99.98% como mínimo.
* Tener una interfaz fácil de comprender y disponible en español.
* Implementar una interfaz lo más ligera posible, para que sea posible acceder e interactuar con ella utilizando la red de internet móvil normalmente disponible en las comunidades.

## 2.2 Restricciones

* El prototipo debe ser implementado en menos de seis meses, para poder completar correctamente este trabajo de tesis.
* El costo total para la creación del prototipo debe ser el menor posible, ya que no se encuentra asignado ningún presupuesto para el desarrollo de este proyecto.
* En cuanto a hardware, se encuentra disponible una computadora con 16Gb de RAM, un procesador I7-8750h y una tarjeta gráfica GTX 1050Ti para el desarrollo de la aplicación. Y para el nodo, se cuenta con un servidor con 32Gb de RAM, un procesador Xeon E3-1240 v3, y con velocidad de conexión de 100MBits.
* El prototipo debe ser implementado en el servidor previamente mencionado, ya que este fue otorgado por Ripple al ITAM como parte de la iniciativa: “Ripple’s University Blockchain Research Initiative”.   
  Donde uno de los requisitos para formar parte de la iniciativa es desarrollar proyectos relacionados a criptomonedas, Blockchain y pagos digitales, utilizando los materiales otorgados por esta.
* En cuanto a software, solamente se poseen licencias que sean de libre uso para el desarrollo del programa.

## 2.3 Trabajos relacionados

Como alternativas al uso de un CIC basado en tecnología DLT para la detonación de una economía local, existen múltiples métodos tradicionales que buscan apoyar a las comunidades vulnerables. Entre estos se encuentran: subsidios sobre productos seleccionados, cupones sobre productos seleccionados, construcción de infraestructura, exención de impuestos para empresas que estén localizadas en el área, promoción turística de la zona, generación de empleos mediante obras públicas y otorgamiento de créditos para creación de empresas o compra de tecnologías.

Dentro de las implementaciones de CIC actualmente en circulación, existen múltiples opciones que varían ampliamente en su objetivo, presentación y obtención.

La Fureai Kippu, es una CIC virtual que busca apoyar a las comunidades de gente mayor en Japón, siendo que se obtiene al realizar una hora de servicios sociales que sean benéficos para los adultos mayores. Esta CIC se puede canjear por servicios del mismo tipo [7].

El dinar Kelantanes es una CIC física que busca ser una alternativa al dinero Fiat. Surge debido a la volatilidad del Ringgit, la moneda oficial de Malasia, y busca solucionarlo al hacer que la CIC se acuñe con metales como el oro o la plata [8].

El Sarafu es una CIC virtual que busca ser una alternativa al dinero Fiat en tiempos de escasez. Surge debido a que la pobreza extrema en Kenya ocasiona que la gente se quede sin dinero, pero sigan teniendo productos que se pueden intercambiar. El objetivo del Sarafu es crear una economía local en la que se puedan seguir intercambiando los bienes disponibles mediante CIC, y no se dependa de la disponibilidad del Shilling keniano, la moneda oficial de Kenia [9].

La Chiemgauer es una CIC física alemana emitida en Bavaria que tiene su valor atado al euro. Su objetivo es ayudar a la economía local, al hacer que la moneda solamente sea aceptada dentro de Bavaria y, por tanto, deba de ser gastada en la comunidad [10].

De todos los trabajos mencionados previamente existen casos de estudio que evalúan el éxito que tuvieron en conseguir sus respectivos objetivos; esto mediante el análisis de que elementos tuvieron éxito y cuales fallaron. Esta experiencia se puede aprovechar en el diseño de este prototipo en múltiples aspectos, pero sobre todo en la creación de CIC virtual, su aplicación para promover la economía local y la manera en la que una CIC define su valor con respecto a la moneda oficial del país.

# 3. DISEÑO

El desarrollo de este capítulo está dividido en cuatro secciones.

En la sección de arquitectura se expondrán las características con las que debe de cumplir el diseño, se presentará la plataforma seleccionada que cumpla con dichas características, se explicará el funcionamiento del prototipo a nivel general y se profundizará en el funcionamiento del prototipo en tres subsecciones tituladas red, base de datos y proceso de transacción.

En la sección de entorno de trabajo, se presentará la solución propuesta con base en lo definido en la sección de arquitectura.

En la sección de soluciones alternativas, se explorarán opciones distintas para la base de datos y algoritmos propuestos.

Por último, se especificarán los estándares y licencias utilizadas en el diseño del prototipo.

## 3.1 Arquitectura

La arquitectura que se utilizará para el diseño del prototipo debe de contar con las siguientes características, las cuales están determinadas por los requerimientos funcionales, requerimientos no funcionales y restricciones presentadas en el capítulo anterior.

* Red permisionada cerrada. Para llevar un control sobre quienes pueden registrarse como usuarios o unirse como nodo.
* Tecnología de Ledger Distribuido. Es necesario cumplir con los requisitos de seguridad, convergencia y disponibilidad que exige el manejo de cualquier tipo de moneda virtual.
* Smart contracts. Para responder a la necesidad de realizar intercambios entre dos usuarios de manera automatizada.
* Algoritmo de consenso. Este debe ofrecer una manera de llegar al consenso que sea rápida, no ocupe muchos recursos computacionales, sea tolerante a fallos, sea tolerante a fallas bizantinas y sea escalable.
* Código abierto. Por las restricciones monetarias.

Al considerar todos los puntos expuestos previamente, se decidió utilizar Hyperledger Iroha como plataforma sobre la cual basar el desarrollo del prototipo, ya que cumple y excede todas las características mencionadas previamente.

La presentación de la arquitectura se dividirá en tres secciones debido a que el prototipo comprende una red para la conexión de los equipos, una base de datos para el almacenamiento de la información y los algoritmos para implementar la funcionalidad del prototipo. A continuación, se presenta una breve explicación paso a paso e ilustrada sobre el funcionamiento usual del prototipo.

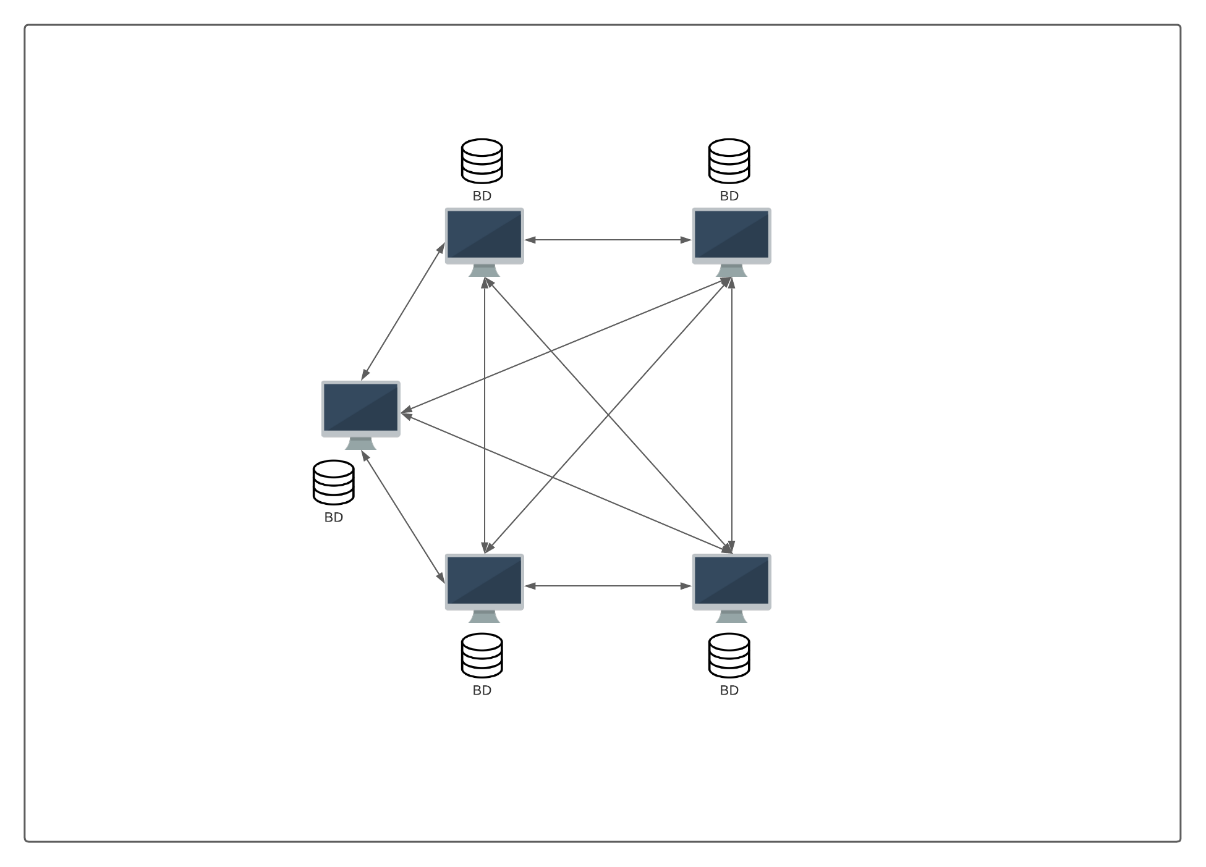


Figura 1. La red se encuentra en estado estable.

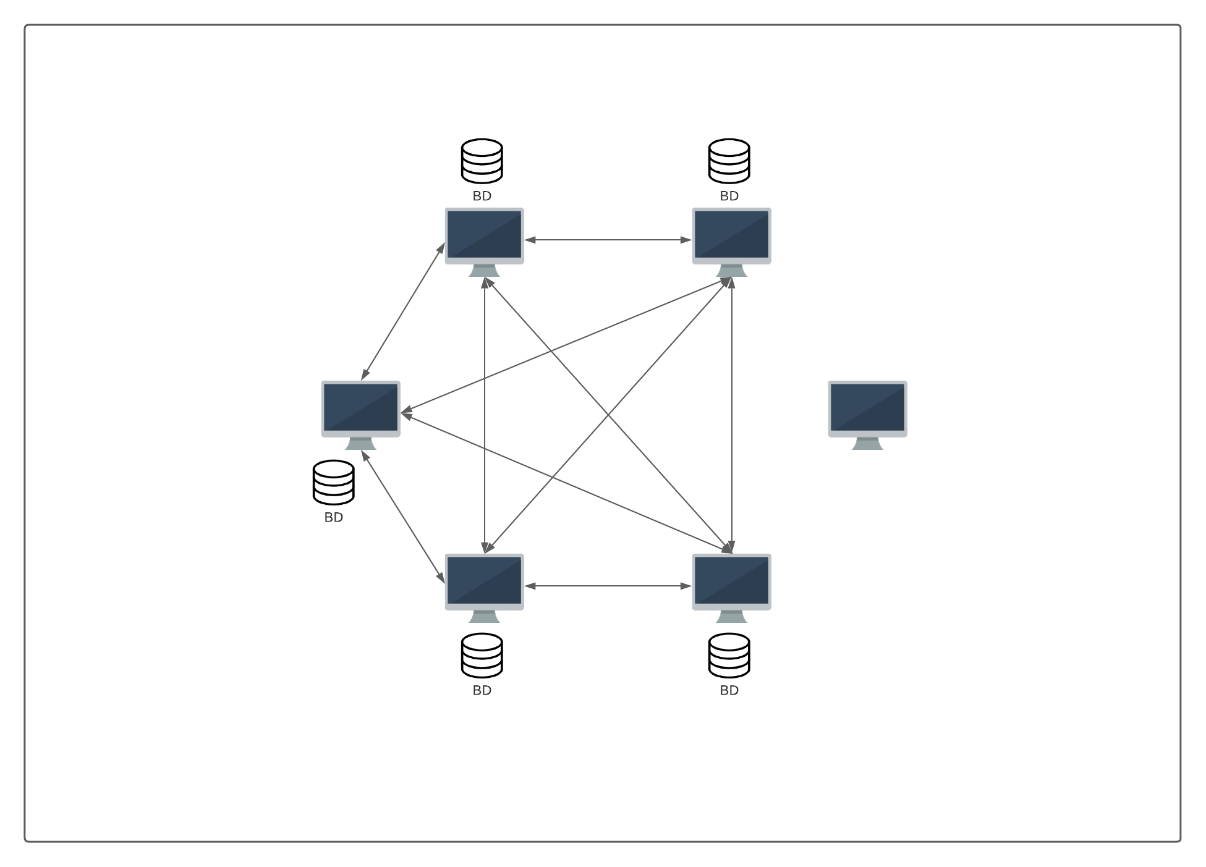


Figura 2. Un nuevo nodo desea unirse a la red establecida.

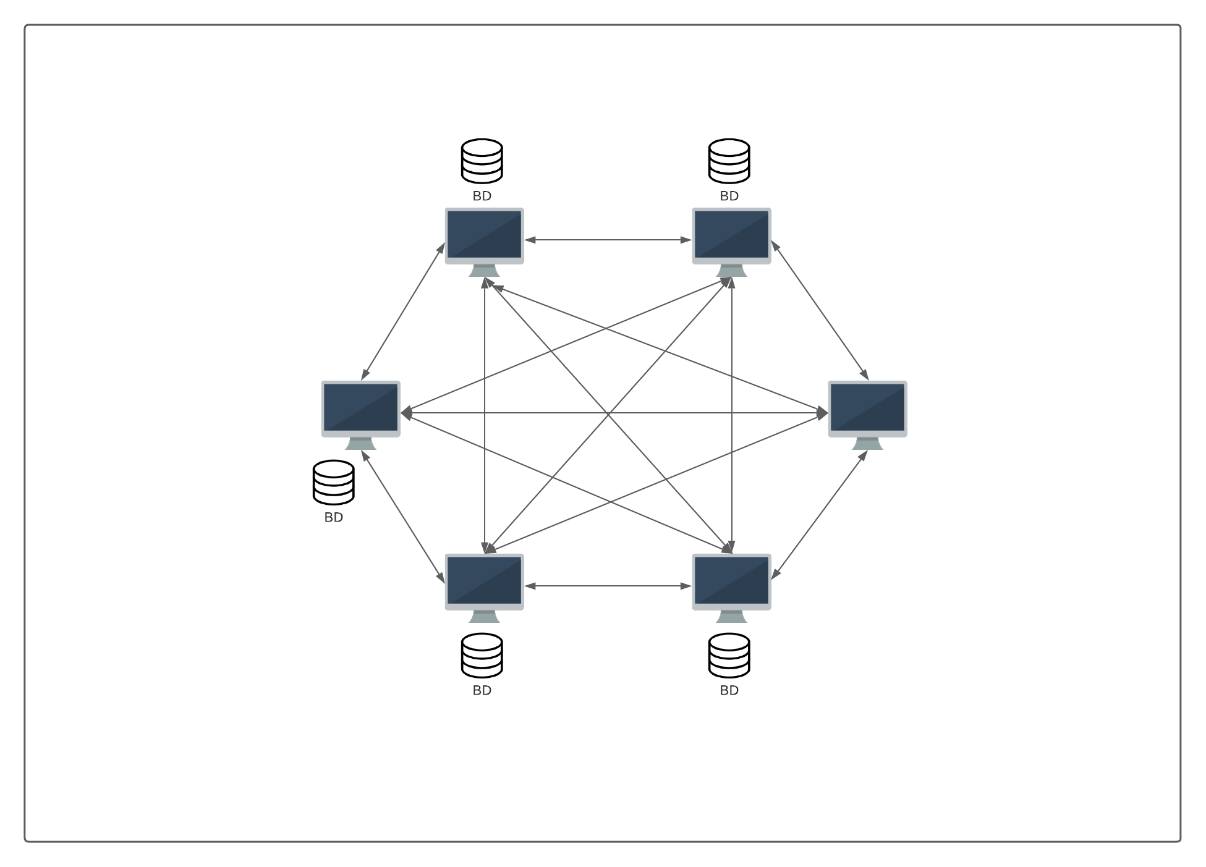


Figura 3. Uno de los nodos que ya forman parte de la red utiliza el comando addPeer, con lo que el nuevo nodo establece una conexión con todos los nodos de la red.

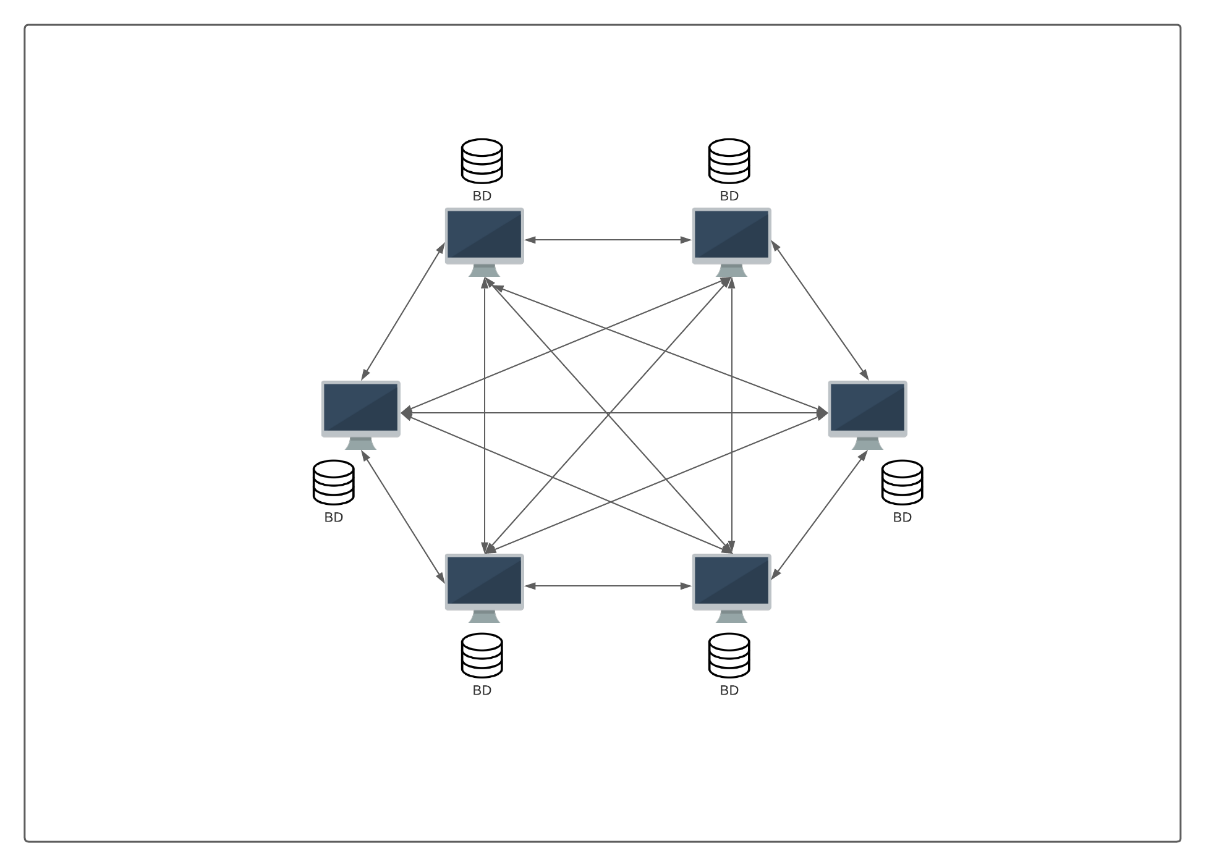


Figura 4. El nuevo nodo utiliza su sincronizador para obtener el estado actual de la base de datos.

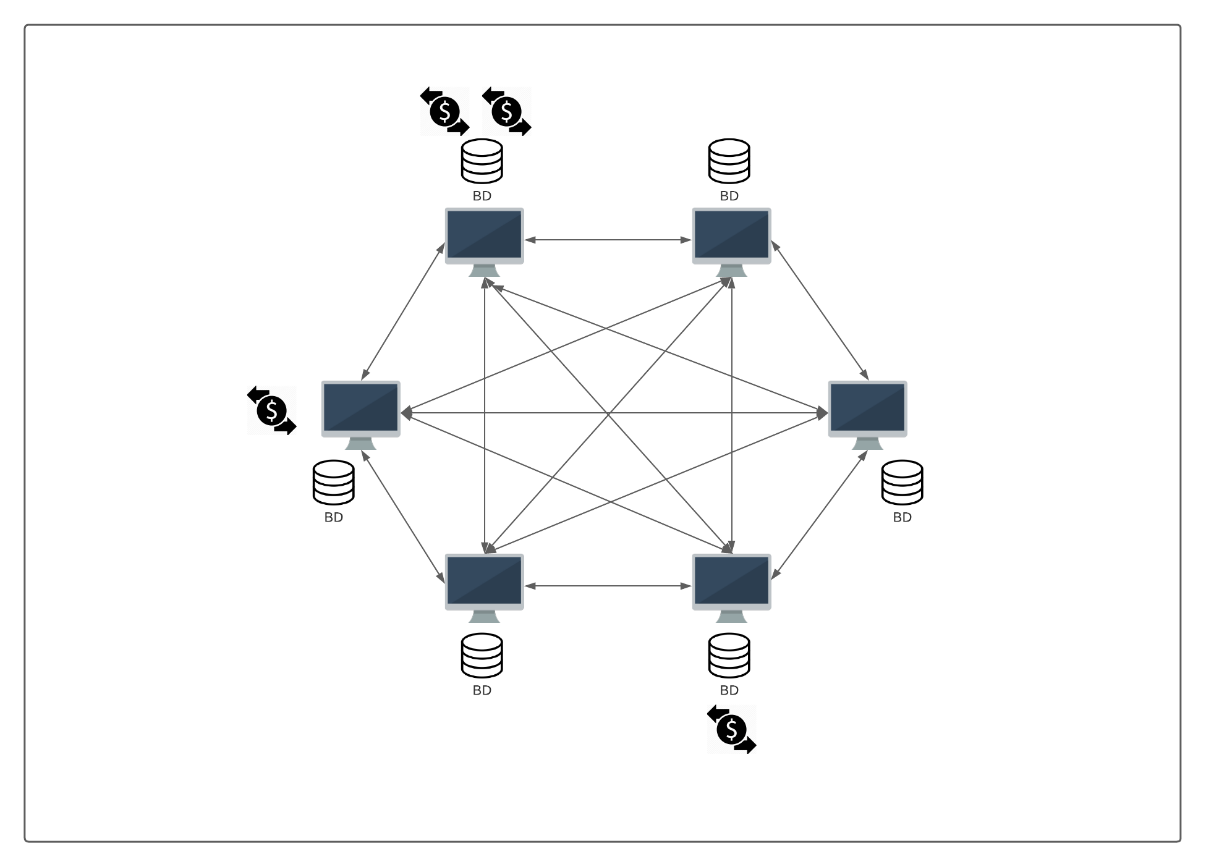


Figura 5. Tres nodos proponen transacciones validadas sintácticamente pero no validadas en cuanto a estado.

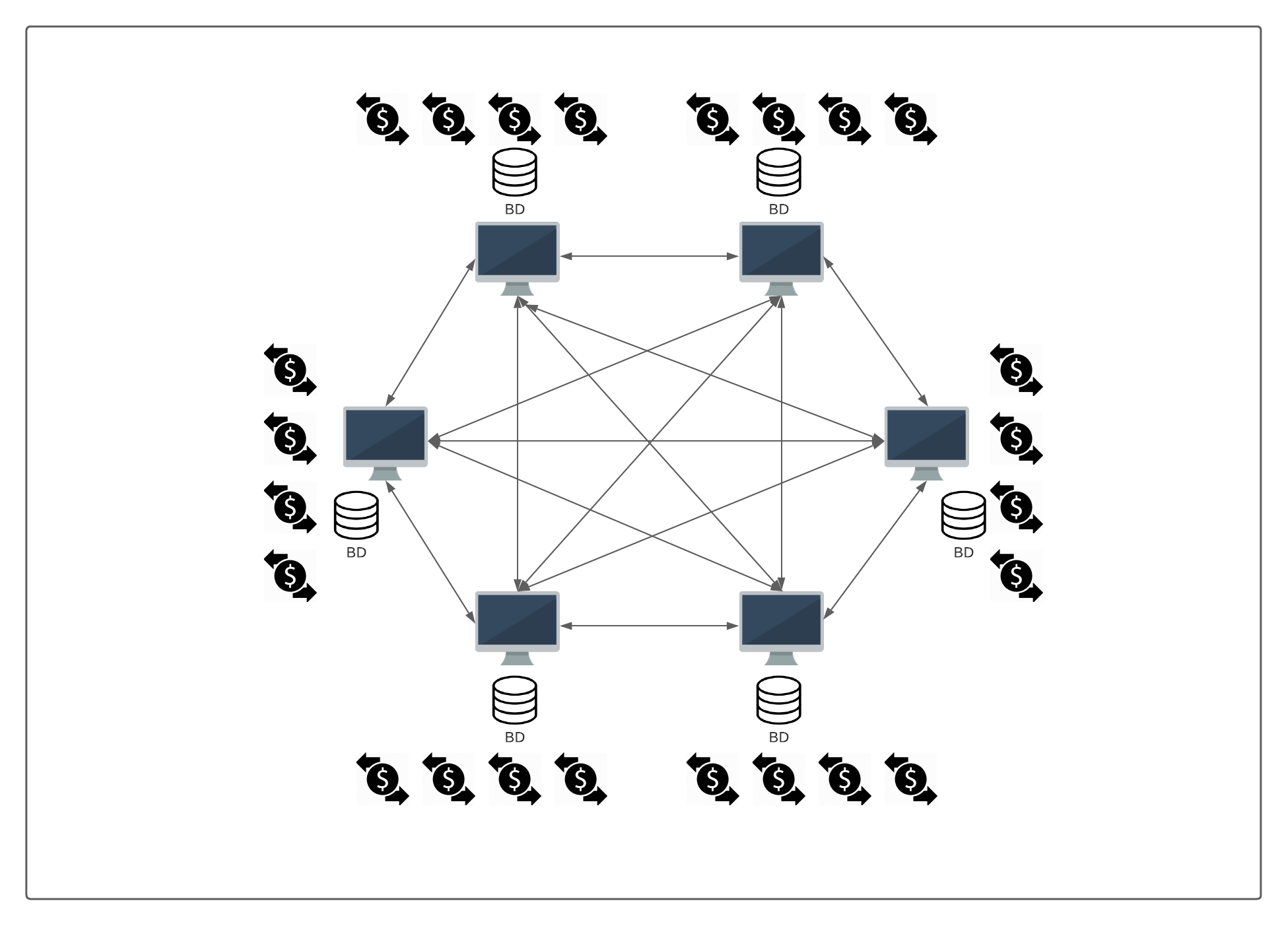


Figura 6. Las transacciones son enviadas a todos los nodos con los que se encuentran conectados.

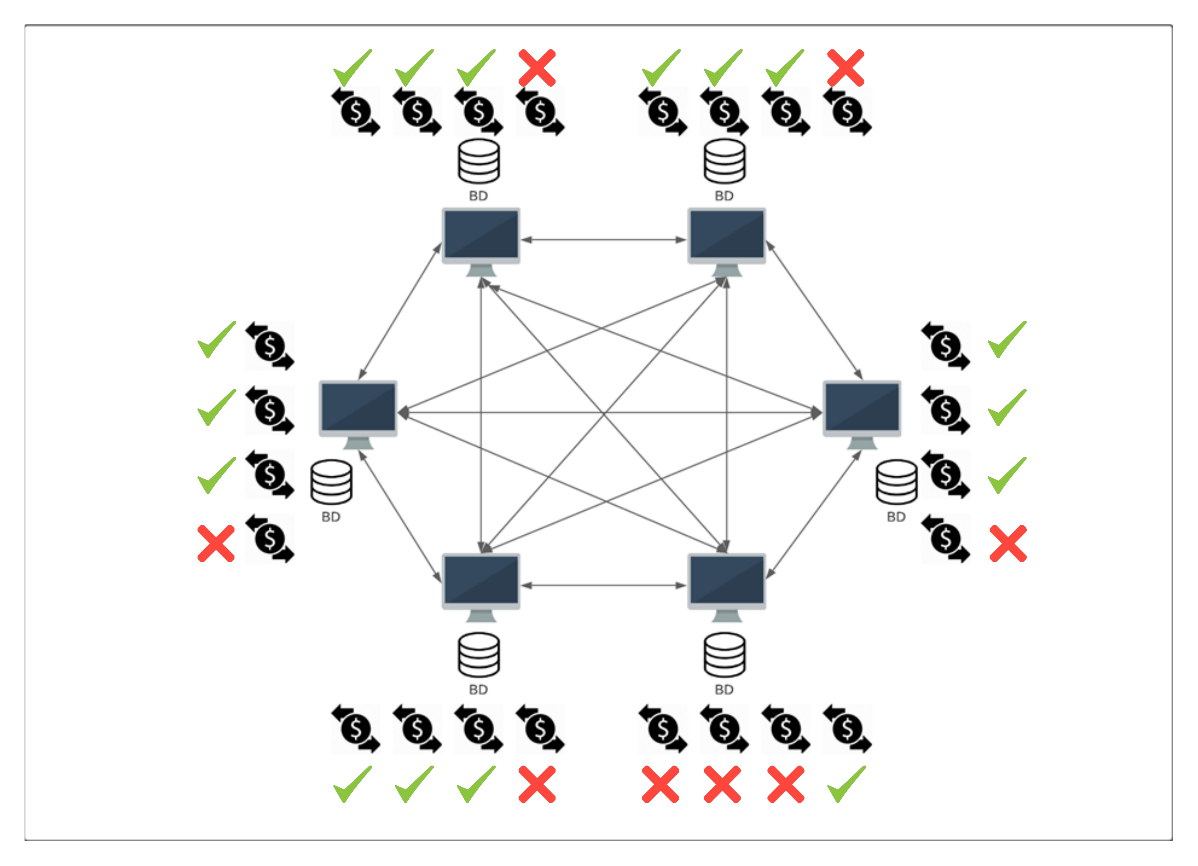


Figura 7. Cada nodo vota individualmente sobre cada transacción, convirtiéndolas en transacciones validadas sintácticamente y en cuanto a estado.

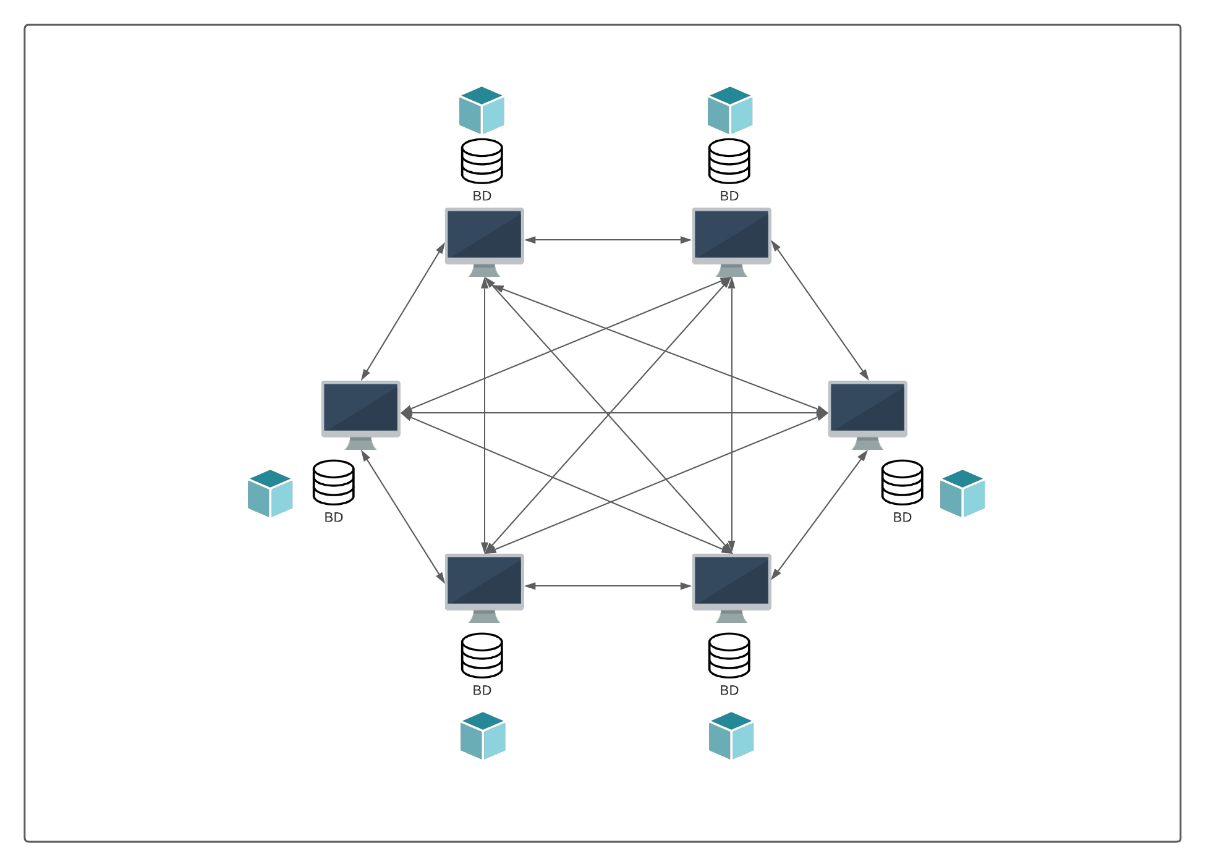


Figura 8. Cada nodo genera un bloque de transacciones con las transacciones validadas y además genera un hash con base en el bloque y los nodos que conforman en la red.

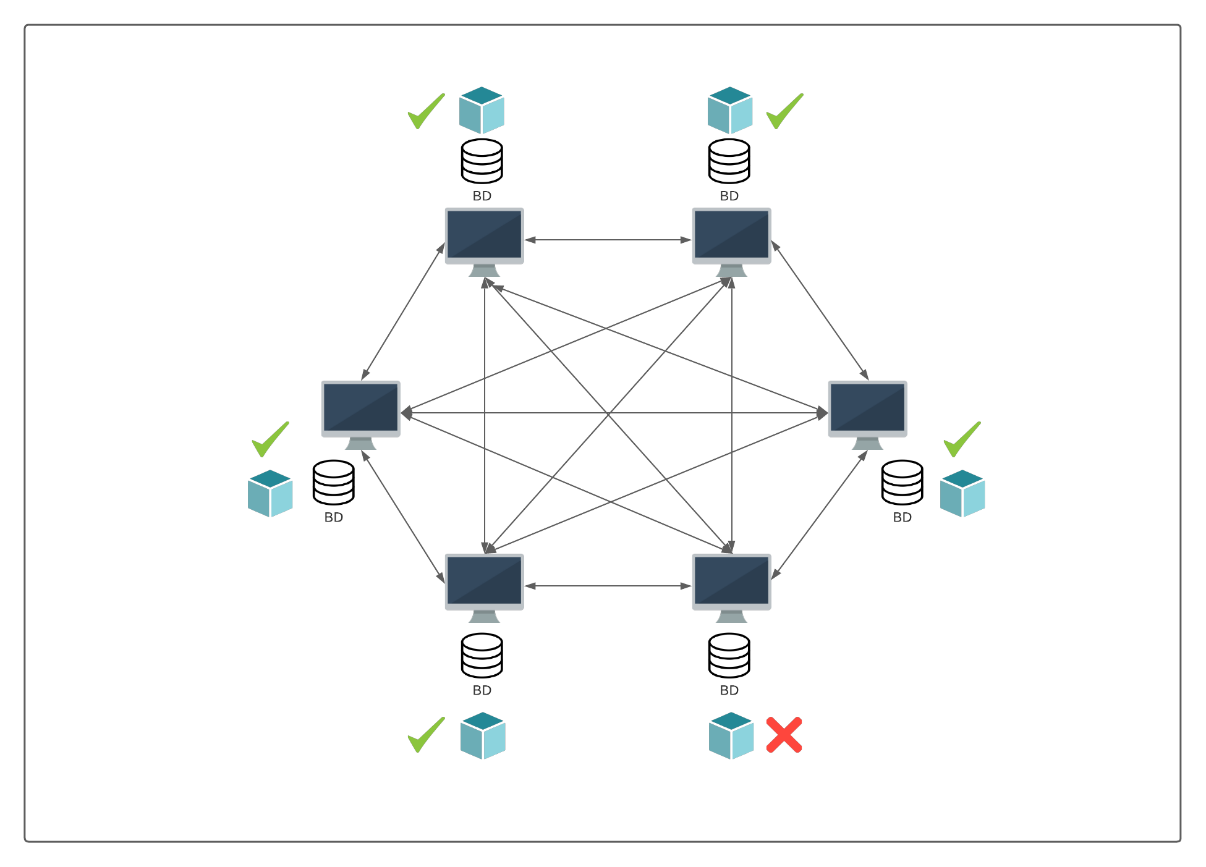


Figura 9. Cada nodo vota sobre el bloque.

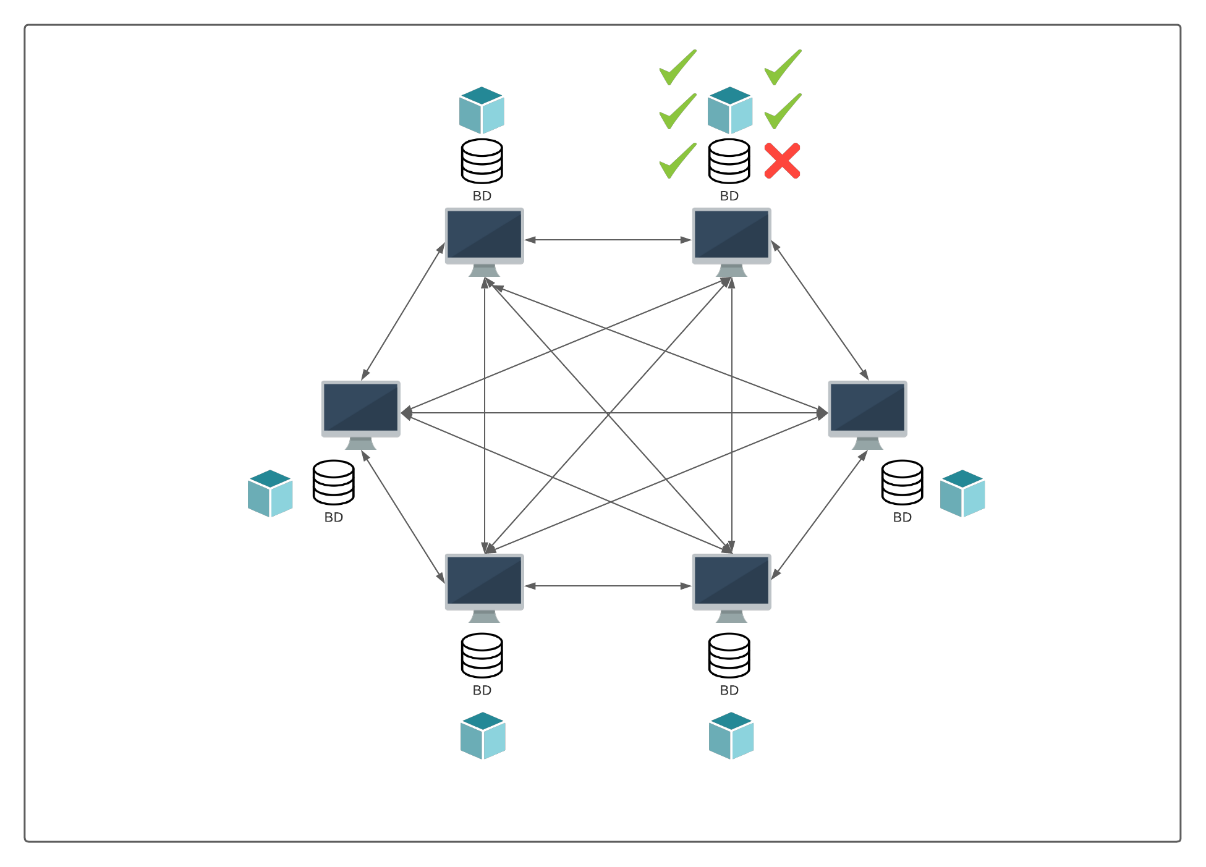


Figura 10. Cada nodo pasa el hash del bloque por una función que le indica a que nodo tiene que enviarle el resultado de su votación, y le envía el resultado.

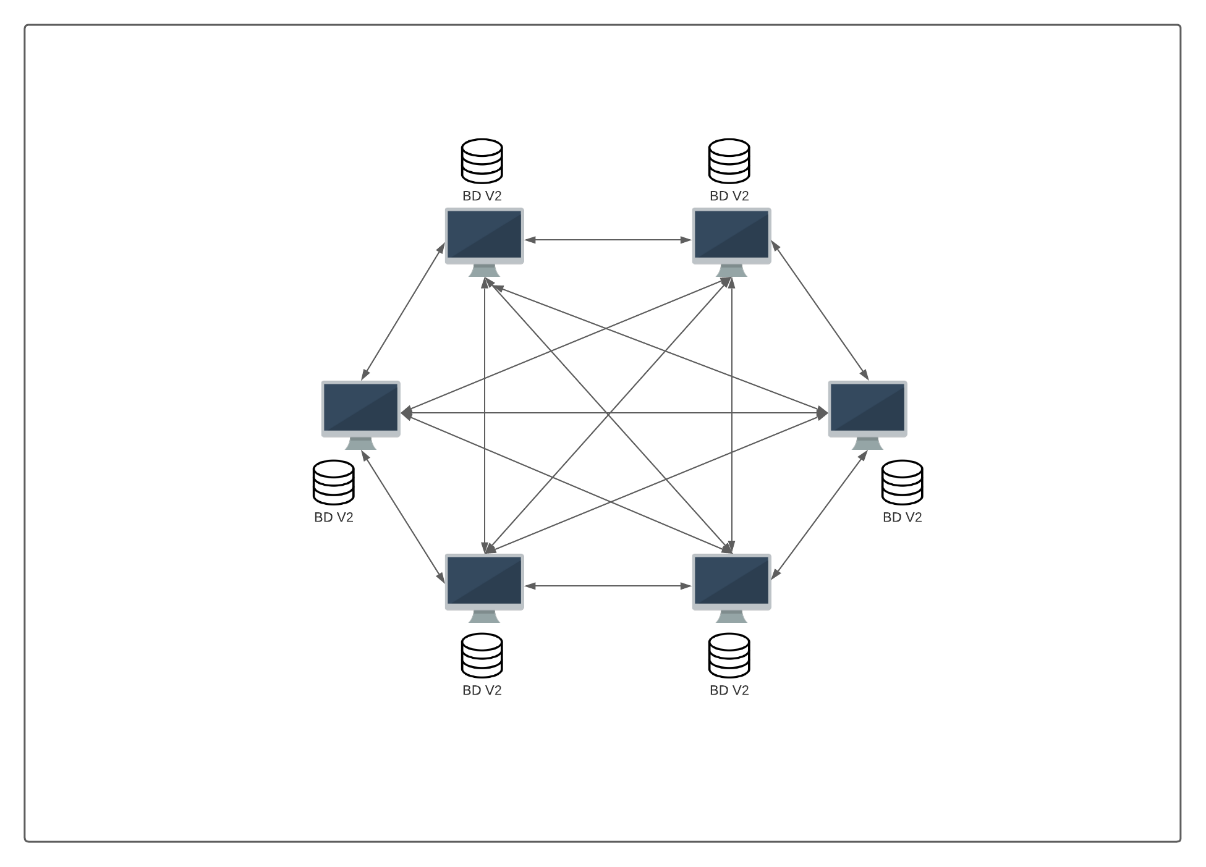


Figura 11. El nodo encargado de la votación cuenta los votos; si recibe 2/3+1 de los votos totales de la red a favor, considera que el bloque es válido, lo registra y manda un mensaje a todos los nodos para que apliquen los cambios. Si recibe 1/3 de los votos totales de la red en contra, considera que el bloque es inválido y manda un mensaje a todos los nodos para que lo desechen.

### 3.1.1 Red

Para el uso de un blockchain, es necesaria la implementación de una red entre múltiples nodos. Por ello, la red que se implementará será de tipo peer to peer, en la que todos los nodos poseen las mismas capacidades.

Además, todos los nodos de la red se conocen entre sí y se encuentran completamente conectados.

El diagrama de esta red se verá de la siguiente forma en el caso en el que existan 6 nodos en la red (Figura 12):

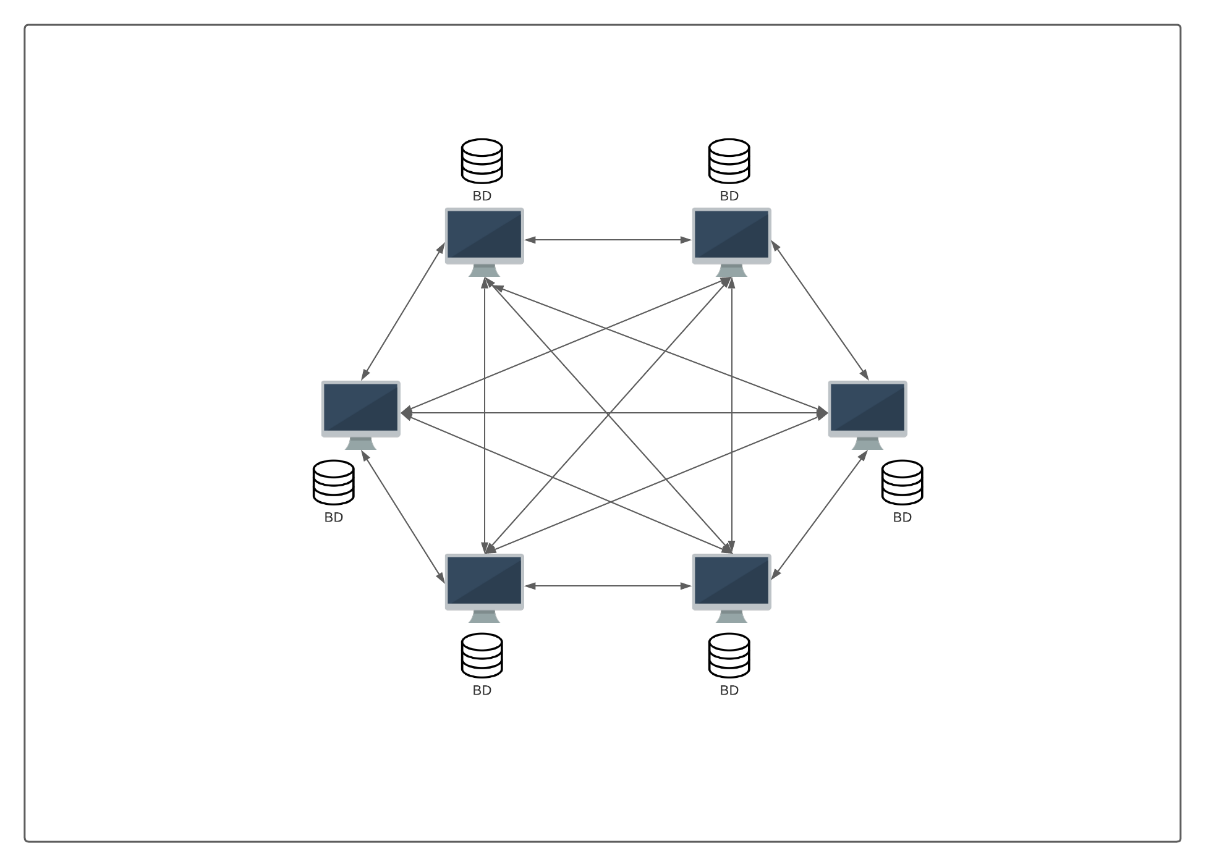


Figura 12. Red Peer to Peer (P2P).

Para que un nodo se conecte por primera vez a la red debe seguir los siguientes pasos:

1. Instalar la paquetería de Hyperledger Iroha y PostgresSQL.
2. Generar un genesis block que sea exactamente igual al de la red con la que desee conectarse.
3. Establecer su dirección y llave de nodo.
4. Esperar a que alguno de los nodos pertenecientes a la red ejecute el comando addPeer para que lo agregue a la red. Una vez ejecutado este comando, todos los nodos pertenecientes a la red solicitarán establecer una conexión con este nodo; lo que debe aceptar.
5. Actualizar su configuración, agregando a su atributo “initial\_nodes”, todos los nodos que se encuentran en la red en ese momento.
6. Utilizar su synchronizer para solicitar a los nodos que le entreguen el estado actual de la base de datos.
7. Actualizar su base de datos con la versión que le fue entregada.

### 3.1.2 Base de datos

Para el desarrollo de este prototipo se contará con dos bases de datos separadas, la base de datos de blockchain y la base de datos de estado. Ambas seguirán lo especificado a continuación.

Cada nodo que conforme a la red debe tener su propia base de datos, y la base de datos debe ser igual para todos los nodos.

La base de datos solamente debe actualizarse a través del sistema implementado por el prototipo, nunca de manera manual.

A continuación, se describirá el diseño de cada base de datos de manera individual.

#### 3.1.2.1 Base de datos de Blockchain:

Esta base de datos comprenderá toda la información relacionada con el blockchain. Es decir que contendrá todos los bloques que conformen al blockchain.

Cada bloque contendrá la siguiente información:

* Timestamp: Es la fecha en la que se creó el bloque; debe especificarse hasta el grado de los milisegundos.
* Altura: Indica la cantidad de bloques que tiene el blockchain hasta el momento.
* Transacciones validadas: Contiene el conjunto de hashes de las transacciones que fueron aprobadas; cada hash debe de estar compuesto por 64 caracteres.
* Transacciones rechazadas: Contiene el conjunto de hashes de las transacciones que fueron rechazadas; cada hash debe de estar compuesto por 64 caracteres.
* Número de transacciones: Indica la cantidad de transacciones que se encuentran dentro del bloque, tanto aprobadas como rechazadas.
* Hash del bloque previo: Este debe ser calculado con base en la información que contiene el bloque anterior. cada hash debe de estar compuesto por 64 caracteres.

Al inicio, la base de datos de blockchain únicamente contará con el bloque génesis del blockchain; y como caso especial, será el único bloque en el que se especifique manualmente el hash del bloque previo. Este hash siempre tendrá el valor de “0000000000000000000000000000000000000000000000000000000000000000”.

Esta base de datos no será modificable de ninguna manera.

Para realizar una consulta, será necesario conectarse a un nodo y enviarle una consulta. El nodo deberá de convertir la consulta en transacción y si la transacción es validada, entonces se responderá a la consulta.

#### 3.1.2.2 Base de datos de estado:

Esta base de datos comprenderá toda la información relacionada con las cuentas, dominios, activos, roles y permisos.

Esta se compondrá de la siguiente manera:

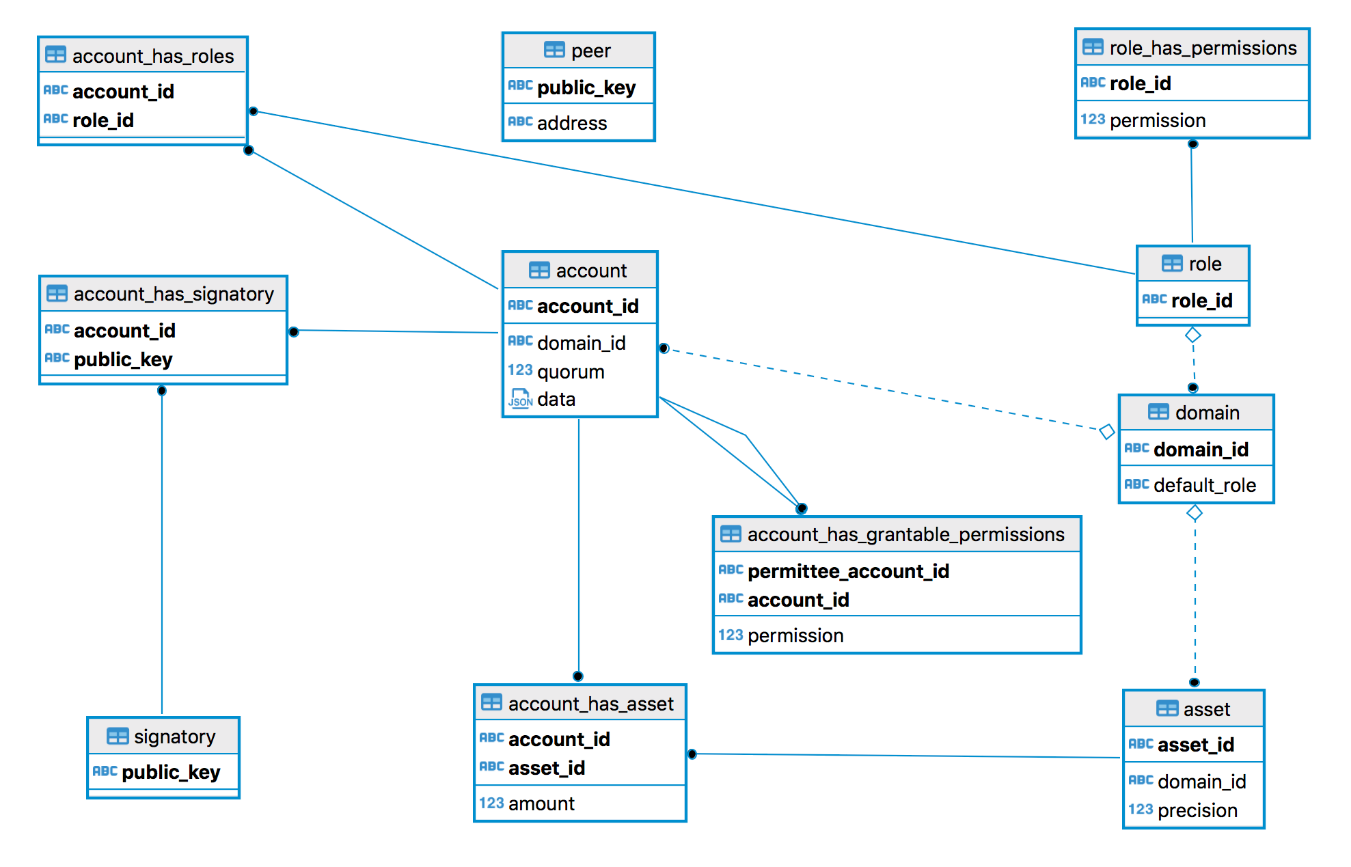


Figura 13. Diagrama Entidad-Relación [11].

A continuación, se dará una breve explicación de cada tabla junto con la información que contendrá cada una:

account: cuentas registradas junto con la información de cada una.

* account\_id es el identificador único de cada cuenta.
* domain\_id es el identificador único del espacio en el que el usuario se encuentra registrado.
* data es un conjunto de detalles sobre la cuenta, estos detalles serán: nombres, apellidos, CURP, correo electrónico, código postal, estado, alcaldía o municipio y asentamiento o colonia.

account\_has\_signatory: cuentas junto con sus llaves públicas.

* account\_id es el identificador único de la cuenta.
* public\_key es la llave pública de la cuenta.

signatory: llaves públicas registradas.

* public\_key es una llave pública.

account\_has\_grantable\_permissions: permisos otorgados por una cuenta a otra.

* permitteee\_account\_id es el identificador único de la cuenta que recibe un permiso.
* account\_id es el identificador único de la cuenta que otorga un permiso.
* permission son los permisos que account\_id le otorgo a permitteee\_account\_id para que pudiera modificar las características de su cuenta.

account\_has\_asset: cuentas junto con la cantidad de activos de cada tipo que cada una posee.

* account\_id es el identificador único de la cuenta.
* asset\_id es el identificador único del activo.
* amount es la cantidad del activo que actualmente posee la cuenta.

asset: activos registrados junto con sus especificaciones.

* asset\_id es el identificador único de un activo.
* domain\_id es el identificador único del espacio en el que el activo se encuentra registrado.
* precision es la cantidad de números decimales con los que puede contar el activo.

domain: dominios registrados junto con su rol estándar.

* domain\_id es el identificador único de cada espacio.
* default\_role es el identificador único del rol asignado por defecto al dominio.

role: roles registrados.

* role\_id es el identificador único de un rol.

role\_has\_permissions: roles junto con los permisos que poseen.

* role\_id es el identificador único del rol.
* permission son los permisos que el rol tiene asignado.

account\_has\_roles: cuentas junto con los roles que poseen

* account\_id es el identificador único de la cuenta.
* role\_id es el identificador único del rol.

peer: nodos registrados junto con su dirección de acceso.

* public\_key es la llave pública del nodo.
* address es la dirección a través de la cual se puede conectar con el nodo.

Para realizar una modificación será necesario conectarse a un nodo y enviarle un comando. El nodo deberá de convertir la consulta en transacción y si la transacción es validada, entonces se aplicará la modificación.

Para realizar una consulta será necesario conectarse a un nodo y enviarle una consulta. El nodo deberá de convertir la consulta en transacción y si la transacción es validada, entonces se responderá a la consulta.

### 3.1.3 Proceso de transacción

A continuación, se explicará el proceso de transacción mediante un diagrama y se explicará paso a paso lo que sucede en cada etapa y también se dará una breve explicación sobre los elementos involucrados en cada etapa.

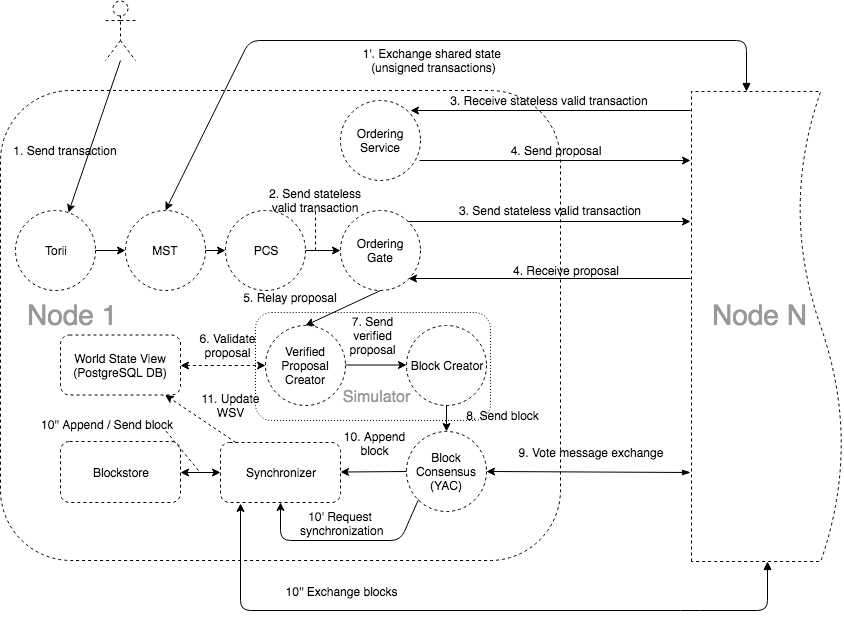


Figura 14. Proceso de transacción [12].

1. El proceso de transacción inicia cuando un usuario genera una transacción y la manda al nodo para que sea procesada.   
   El nodo recibe esta transacción mediante la puerta de Torii, cuyo objetivo es recibir las transacciones y analizarlas de manera sintáctica para enviarlas al PCS (Peer Communication Service) en caso de que se consideren válidas, o descartarlas en caso de que no.
   1. En caso de que alguna de las transacciones que haya sido enviada al nodo necesite ser firmada por más de un usuario, esta es enviada por el Torii hacia el procesador de MST (Multisignature Transactions Processor).   
      El MST envía la transacción al resto de los nodos para que pueda recibir la firma de los usuarios faltantes. Al recibir las firmas faltantes, el MST envía la transacción al PCS.
2. El PCS recibe la transacción, la prepara para ser enviada al resto de los nodos y la envía al Ordering Gate.
3. El ordering gate recibe la transacción del PCS y la envía hacia el ordering service de todos los demás nodos. Al mismo tiempo, el ordering service de este nodo esta recibiendo aquello enviado por el ordering gate de los demás nodos.
4. El ordering service organiza las transacciones que le fueron enviadas y las propias en un solo paquete llamado propuesta y envía esta propuesta al ordering gate de los demás nodos.   
   El ordering service sigue dos criterios para generar una propuesta, el primero es un número máximo de transacciones, donde si se alcanza el número máximo se genera la propuesta con dichas transacciones y se guarda el resto de las transacciones para otra propuesta. El segundo es un timeout, donde si se alcanza el tiempo de timeout y no se ha alcanzado el número máximo de transacciones, se genera la propuesta con las transacciones existentes.
5. El ordering gate envía la propuesta al creador de propuestas verificadas.
6. El creador de propuestas verificadas compara cada transacción contra el World State View, que es la base de datos de estado, y determina si cada transacción es válida o no.   
   La propuesta puede tener transacciones no válidas y otras que si lo sean.
7. El creador de propuestas verificadas manda la propuesta al creador de bloques, el cual elimina las transacciones que son inválidas. Con las transacciones restantes crea un bloque que sigue todas las especificaciones del blockchain y, por último, con la información del bloque genera un hash y lo asigna al bloque.
8. El creador de bloques envía el bloque al algoritmo de consenso YAC (Yet Another Consensus Algorithm).
9. Cada nodo vota si rechaza o acepta el bloque que se está tratando de agregar.   
   Después, el algoritmo de consenso utilizará el hash del bloque como parámetro para su función de ordenamiento, siendo que lo que regresa esta función es el identificador del nodo al que se le tienen que enviar los resultados de la votación. Debido a que todos los nodos tienen el mismo hash, todos los nodos calculan el mismo identificador de nodo al que se le tienen que enviar los resultados.   
   Posteriormente, el nodo que está encargado de recibir los votos esperará a que se cumpla una de dos condiciones. Si recibe 2/3+1 de los votos totales existentes en la red y son votos a favor, entonces manda un mensaje a todos los nodos para que agreguen el bloque al blockchain. Si recibe 1/3 de los votos totales existentes de la red y son en contra, entonces manda un mensaje a todos los nodos para que descarten el bloque y finalicen este proceso.
10. El algoritmo de consenso le manda el bloque al synchronizer, el cual debe mandarlo a su vez al blockstore, o base de datos de blockchain, para que agregue el bloque a la cadena.  
    En caso de que un nodo sufra de una conexión lenta, y apenas haya mandado su voto al nodo que recibe todos los votos, este recibe inmediatamente la instrucción de solicitar una sincronización con el synchronizer. Al solicitar la sincronización, el synchronizer se comunica con el synchronizer de los demás nodos para solicitar la versión más actual del blockchain y si recibe algún bloque nuevo, sigue el proceso de mandarlo al blockstore para que se agregue al blockchain.
11. Finalmente, el synchronizer actualiza la base de datos de estado con la información de las transacciones que se aplicaron.

## 3.2 Entorno de trabajo

El entorno de trabajo de Hyperledger Iroha utiliza un sistema de control de acceso basado en roles, por ello, está conformado por los siguientes cinco elementos:

* Permiso. Determina si se puede ejecutar, o no, una acción en específico.
* Rol. Es un conjunto de permisos y es identificable por su nombre de rol.
* Dominio. Es un espacio virtual en el que se agrupan usuarios y activos. En este espacio se ejecutarán todas las acciones como intercambiar activos, agregarlos, etc. Es identificable por su nombre de dominio, y cuenta con un rol por default que se le agrega a todos los usuarios la primera vez que se registran en el dominio.
* Usuario. Es la unidad funcional del entorno de trabajo. El conjunto de acciones que puede realizar esta determinado por el rol, o roles, que posea; y siempre debe tener asignados un dominio y al menos un rol.
* Activo. Es aquello que se intercambia, recibe y entrega por los usuarios; tiene asignado un dominio.

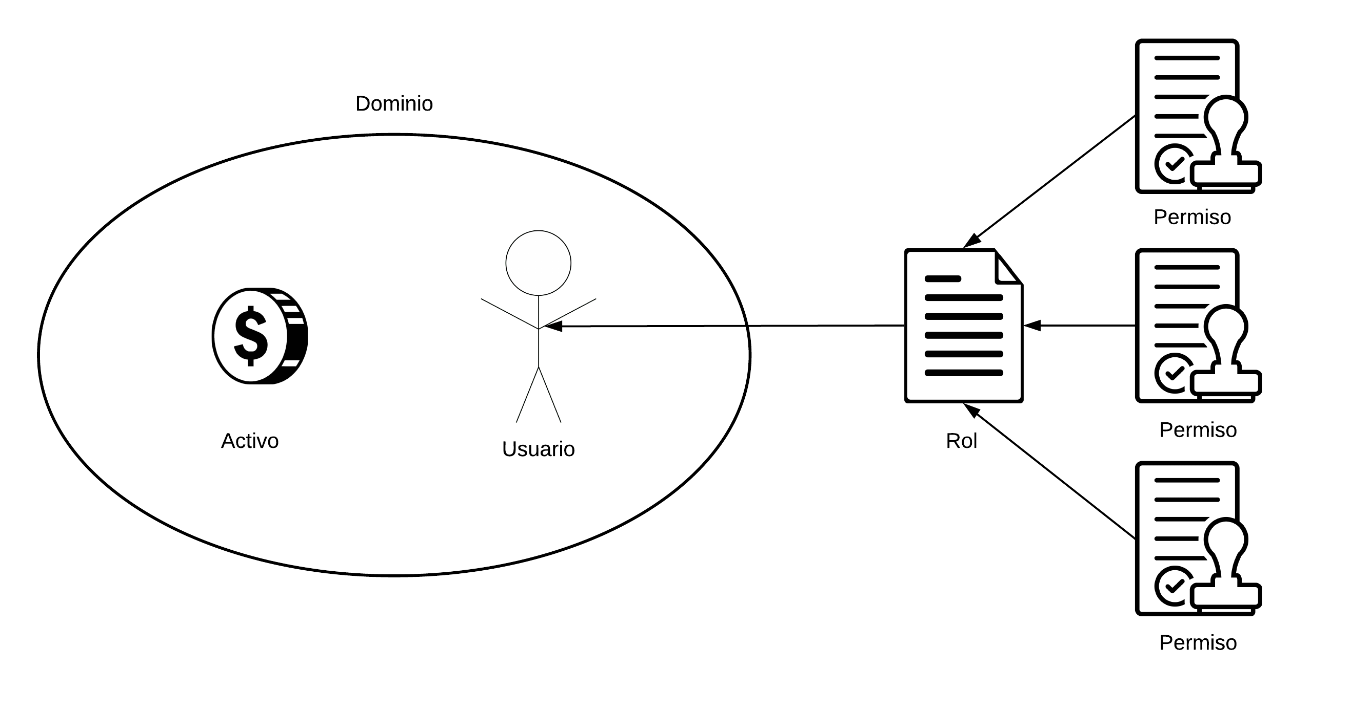


Figura 15. Entorno de trabajo.

Para el diseño del entorno de trabajo utilizado por este prototipo, se crearán y configurarán los roles, dominios y activos. No se configurarán los permisos ya que estos ya están definidos por Hyperledger Iroha, ni los usuarios, ya que esto se realizará en la implementación.

A continuación, se mencionará la configuración que se utilizará para los roles, dominios y activos.

3.2.1 Dominios

Se creará un solo dominio que representará a la comunidad en la que se implementará el prototipo. Sin embargo, a futuro se pueden agregar más dominios para abarcar más comunidades en caso de que el prototipo sea un éxito.

### 3.2.2 Roles

En este diseño, se incluirán los permisos necesarios en caso de que se decida implementar más de un dominio en el futuro. Se decidió esto debido a que estos permisos no afectan en nada a la actual implementación en caso de que no se utilicen; pero es problemático agregarlos una vez ya realizada la implementación.

A continuación, se indicarán los roles que se van a crear y se dará una breve explicación del porque se les asigno cada permiso.

Se crearán tres roles:

Usuario:

1. can\_receive: Para que el usuario reciba dinero de otras cuentas.
2. can\_transfer: Para que el usuario envíe dinero a otras cuentas.
3. can\_grant\_can\_set\_my\_account\_detail: Para que el administrador configure los detalles de la cuenta.
4. can\_grant\_can\_set\_my\_quorum: Para que el administrador de dinero modifique el quorum necesario para las MST (Multi Signature transaction).
5. can\_grant\_can\_transfer\_my\_assets: Para implementar la reducción porcentual de dinero al pasar un mes.
6. can\_add\_signatory. Para que los usuarios agreguen llaves públicas extras a su cuenta.
7. can\_remove\_signatory. Para que los usuarios eliminen llaves públicas de su cuenta.
8. can\_set\_quorum. Para realizar MST con otros usuarios.
9. can\_get\_my\_acc\_detail. Para que el usuario consulte sus datos.
10. can\_get\_my\_account. Para que el usuario consulte su estatus general en el dominio.
11. can\_get\_my\_acc\_ast. Para que el usuario consulte la cantidad de activos que posee.
12. can\_get\_my\_acc\_ast\_txs. Para que el usuario consulte todas las transacciones que ha realizado respecto a un activo en específico.
13. can\_get\_my\_acc\_txs. Para que el usuario consulte todas las transacciones que ha realizado.
14. can\_read\_assets. Para que el usuario consulte los datos de un activo.
15. can\_get\_my\_signatories. Para que el usuario consulte sus llaves públicas.
16. can\_get\_my\_txs. Para que el usuario consulte la información correspondiente a una de sus transacciones en específico.

Administrador de dinero:

1. can\_receive. Para que el administrador de dinero reciba dinero e inicie el proceso de intercambiarlo por una moneda FIAT.
2. can\_transfer. Para que el administrador de dinero cree dinero y lo transfiera a un usuario cuando este cumpla con los requisitos para recibir el CIC.
3. can\_set\_quorum: Para realizar MST con los usuarios que vayan a recibir dinero o quieran intercambiarlo por una moneda FIAT.
4. can\_get\_my\_account. Para que el administrador de dinero consulte su estatus general en el dominio.
5. can\_get\_my\_acc\_ast. Para que el administrador de dinero consulte la cantidad de activos que posee.
6. can\_read\_assets. Para que el administrador de dinero consulte los datos de un activo.
7. can\_add\_domain\_asset\_qty. Para que el administrador de dinero cree activos y agregarlos a su cuenta, limitando esto a los activos que existen en el dominio al que pertenece el administrador de dinero.
8. can\_subtract\_domain\_asset\_qty. Para que el administrador de dinero elimine activos de su cuenta, limitando esto a los activos que existen en el dominio al que pertenece el administrador de dinero.
9. can\_get\_domain\_acc\_detail. Para identificar con qué tipo de usuario se están realizando las transacciones. limitando esto a las cuentas que existen en el dominio al que pertenece el administrador de dinero.
10. can\_get\_domain\_accounts. Para identificar las cuentas existentes en el dominio al que pertenece el administrador de dinero.
11. can\_get\_domain\_signatories. Para obtener las firmas de las cuentas existentes en el dominio al que pertenece el administrador de dinero.

Administrador:

1. can\_create\_account. Para que el administrador cree una cuenta nueva para cada usuario que comience a utilizar el prototipo por primera vez.
2. can\_set\_detail. Para que el administrador configure los detalles de las cuentas existentes.
3. can\_add\_peer. Para que el administrador agregue a la red a un nodo que se quiera unir por primera vez.
4. can\_remove\_peer. Para que el administrador elimine de la red a un nodo.
5. can\_append\_role. Para que el administrador añada roles a un usuario ya existente.
6. can\_detach\_role. Para que el administrador elimine roles de un usuario ya existente.
7. can\_read\_assets. Para que el administrador consulte los datos de un activo.
8. can\_get\_blocks. Para que el administrador consulte los bloques existentes en la base de datos de blockchain.
9. can\_get\_roles. Para que el administrador consulte todos los roles existentes en el entorno de trabajo.
10. can\_get\_peers. Para que el administrador consulte la información de todos los nodos conectados en la red.
11. can\_create\_role. Para que el administrador cree nuevos roles y asignarles permisos.
12. can\_create\_asset. Para que el administrador cree nuevos activos. No se le asignó al administrador de dinero debido a que la funcionalidad de creación y destrucción de elementos debe estar limitada al Administrador.
13. can\_create\_domain. Para que el administrador cree nuevos activos.
14. can\_get\_domain\_signatories. Para obtener las llaves públicas de las cuentas existentes en el dominio al que pertenece el administrador.
15. can\_get\_domain\_acc\_detail. Para obtener los detalles de las cuentas existentes en el dominio al que pertenece el administrador.
16. can\_get\_domain\_accounts. Para obtener el listado de cuentas existentes en el dominio al que pertenece el administrador.
17. can\_get\_domain\_acc\_ast. Para obtener el listado de activos existentes en el dominio al que pertenece el administrador.
18. can\_get\_domain\_acc\_ast\_txs. Para obtener el conjunto de transacciones realizadas con un activo en específico en el dominio al que pertenece el administrador.
19. can\_get\_domain\_acc\_txs. Para obtener el conjunto de transacciones realizadas en el dominio al que pertenece el administrador.

Además de estos diecinueve permisos necesarios para el correcto funcionamiento del rol, se agregarán al rol de Administrador los permisos pertenecientes a los roles de Administrador de dinero y Usuario.

Esto se debe al diseño de Hyperledger Iroha, ya que ciertos comandos, como por ejemplo asignar un rol, requieren que el usuario que está ejecutando el comando tenga al menos los mismos permisos que el rol que está tratando de asignar.

### 3.2.3 Activos

Se creará un solo activo para cada dominio existente y este activo será el CIC. Debido a que en este momento solamente se cuenta con un dominio, solamente se creará un activo; sin embargo, si a futuro se agregan más dominios, se crearán más activos para que cada dominio tenga su propio activo.

## 3.3 Soluciones alternativas

Se decidió utilizar Hyperledger Iroha como plataforma para desarrollar este prototipo; sin embargo, existen otras plataformas distintas a Hyperledger Iroha que cumplen con las características requeridas para este proyecto en menor o mayor medida, pero que ofrecen distintas funcionalidades.

A continuación, se hará una breve presentación sobre estas soluciones alternativas. Y debido a que en estas secciones se hace referencia a múltiples algoritmos de consenso, se agregó al final de este documento el Anexo A, en el cual viene una breve explicación sobre los algoritmos de consenso más usados y cada uno de los mencionados en este trabajo.

### 3.3.1 Ethereum

Es una plataforma open source que tiene como objetivo permitir el envío de criptomoneda entre todas las personas que cuenten con una cartera que acepte dicha criptomoneda. [13]

Permite realizar forks de su código fuente para generar una solución basada en Ethereum pero que se adapte a las necesidades de cada proyecto. Esto resulta en que principalmente soporte ETH, que es la criptomoneda nativa, pero que también sea compatible con otras criptomonedas, principalmente aquellas que son criptomonedas nativas de forks de Ethereum.

La ventaja principal de Ethereum es que ofrece la implementación de contratos inteligentes; es decir, transacciones que se pueden ejecutar de manera automática al cumplirse una serie de condiciones especificadas previamente.

Las razones por la que no se escogió utilizar Ethereum son que su algoritmo de consenso utiliza múltiples recursos al utilizar una combinación de proof of work y proof of stake; además, el tiempo requerido para finalizar una transacción ronda los 15 segundos; y finalmente, sería necesario haber desarrollado un fork de Ethereum para poder establecer una red permisionada privada [14].

### 3.3.2 Hyperledger

Cuenta con otras cinco plataformas dedicadas a la creación de ledgers distribuidos además de Iroha. Estas son Besu, Burrow, Indy, Sawtooth y Fabric; sin embargo, Hyperledger Indy está orientada al control de identidades y no cubre los requisitos para el prototipo, por lo que no se mencionará. Las características que ofrece cada una a nivel general son semejantes a las ofrecidas por Iroha, por lo que a continuación solamente se mencionaran las características que diferencian a cada una de las demás y la razón por la que no se escogió [15].

Hyperledger Besu es un cliente de Ethereum, por lo que puede utilizar todas las funcionalidades de Ethereum y se puede conectar directamente a la red utilizada por Ethereum, o crear una red que funcione de manera local. Otra ventaja que tiene además de la compatibilidad con Ethereum es que es un sistema modular, por lo que se le pueden agregar plugins oficiales o independientes fácilmente, complementando con esto el conjunto de funcionalidades que tiene.

Sus desventajas son que los algoritmos de consenso que usa son únicamente Proof of Work, Proof of Stake o Proof of Authority; y que usa muchos recursos de CPU debido a la integración de Ethereum con las características de Hyperledger, y además, la integración de los plugins [14], [16].

Hyperledger Burrow tiene la ventaja de dar soporte al EVM (Ethereum Virtual Machine) y, por lo tanto, de soportar los Smart Contracts desarrollados en Ethereum.

Su desventaja es que está orientada a las redes públicas y específicamente a las redes que utilizan Proof of Stake como algoritmo de consenso [17].

Hyperledger Sawtooth es una plataforma cuyo procesador de transacciones se encuentra integrado con Ethereum; a este procesador se le llama Seth debido a esto. Este procesador integrado le permite a Sawtooth utilizar todos los recursos de solidity ofrecidos por Ethereum. Además, puede implementarse en redes tanto públicas como privadas y ofrece múltiples algoritmos de consenso que se pueden utilizar, siendo estos PBFT, Proof of Elapsed Time, Raft, Proof of Work y Proof of Stake.

Su desventaja es que su red de nodos permite que haya forks dentro de las versiones de las bases de datos en la red, por lo que la finalidad de las transacciones se puede ver retrasada en lo que se resuelven los forks y se regresa la red a una versión estable [18].

Hyperledger Fabric es la plataforma principal de Hyperledger y la mayor ventaja que tiene sobre las demás plataformas es que puede abrir canales privados dentro de la red; este canal privado incluye a un subconjunto de los usuarios de la red, siendo que este subconjunto puede realizar transacciones entre si con la ventaja de que nadie que no forme parte del canal pueda enterarse de estas transacciones.

Sus desventajas son que es una red orientada al intercambio de activos e información entre empresas, por lo que la funcionalidad para administrar y utilizar una criptomoneda se ve reducida; y además, el algoritmo de consenso que utiliza no es tolerante a fallas bizantinas [14], [19].

## 3.4 Estándares y licencias utilizados

Para la implementación del prototipo, se utilizarán múltiples herramientas, protocolos y demás elementos ya existentes que siguen un conjunto de reglas y especificaciones. A continuación, se presentarán los estándares que siguen los elementos que se utilizarán en el prototipo.

1. IEEE 802.3n. Es un estándar creado por el Institute of Electrical and Electronic Engineers (IEEE). Define la manera en la que debe realizar el control de accesos a medios vía ethernet para las capas física y de enlace de datos [20].
2. IEEE 802.11n. Es un estándar creado por el IEEE. Define la manera en la que debe realizar el control de accesos a medios de manera inalámbrica para las capas física y de enlace de datos [21].
3. ISO/IEC 14882:2020. Es un estándar creado de manera conjunta por el International Organization for Standarization (ISO) y la International Electrotechnical Commision (IEC). Define los requisitos para implementaciones de C++ y, por lo tanto, como es que se define C++ [22].
4. ISO/IEC 9075-1. Es un estándar creado de manera conjunta por el ISO y la IEC. Define los requisitos que se deben cumplir para decir que se está utilizando un Framework de SQL [23].
5. ISO/IEC 9075-2. Es un estándar creado de manera conjunta por el ISO y la IEC. Define los requisitos que se deben cumplir para decir que el software cumple con los fundamentos de SQL [23].
6. ISO/IEC 9075-11. Es un estándar creado de manera conjunta por el ISO y la IEC. Define los requisitos que se deben cumplir para decir que se están creando esquemas de definición y se esta registrando la información según lo especificado por SQL [23].
7. ISO/IEC 9075-14. Es un estándar creado de manera conjunta por el ISO y la IEC. Define los requisitos que se deben cumplir para decir que se están utilizando las especificaciones de XML definidas por SQL [23].
8. GNU General Public License. Es una licencia de derechos de autor creada por Free Software Foundation. Permite el uso, estudio, modificación e intercambio del programa evitando que cualquier persona se apropie del código que este protegido bajo dicha licencia y en caso de distribuir una versión modificada, incluir el código fuente de dicha distribución [24].
9. MIT License. Es una licencia de derechos de autor creada por el Massachusetts Institute of Technology (MIT). Permite el uso, copia, modificación, fusión, publicación, distribución, sub - licenciamiento y venta del software [25].
10. Python Software Foundation License. Es una licencia de software libre creada por el Python Software Foundation. Permite el uso, copia, modificación y distribución del software mientras se conserve el aviso de copyright del Python Software Foundation License [26].
11. Apache License. Es una licencia de software libre creada por The Apache Software Foundation. Permite el uso, distribución, modificación, y venta del software mientras se conserve el aviso de copyright de The Apache Software Foundation y se descargue de responsabilidad a esta [27].
12. Certified Kubernetes Conformance Program. Es una certificación creada por The Linux Foundation. Se otorga solamente al software que demuestra cumplir con los estándares de calidad de producto o servicio y los estándares de mantenimiento establecidos por The Linux Foundation [28].

# 4. IMPLEMENTACIÓN

Durante el desarrollo de este capítulo, se realizará una descripción paso a paso sobre el proceso de implementación que se llevó a cabo para la creación del prototipo. Además, al final se especificarán todos los elementos de software y hardware que fueron utilizados durante el desarrollo.

## 4.1 Descripción completa de la implementación

Para el desarrollo de esta sección, se comenzará indicando la serie de pasos necesarios para la instalación y configuración de la paquetería y herramientas requeridas para la implementación de la funcionalidad base.

Posteriormente se describirá el entorno de trabajo y se especificarán los pasos que se tomaron para generarlo.

Finalmente se presentará la interfaz desarrollada para la interacción con el prototipo y los pasos necesarios para su implementación.

Debido a que el prototipo está desarrollado sobre el sistema operativo de Windows 10 de 64 bits, los pasos a continuación presentados son los que se deben seguir para la correcta implementación de la solución en Windows 10 de 64 bits.

### 4.1.1 Instalación y configuración de paquetería y herramientas para la construcción inicial.

La serie de pasos que se presentaran a continuación son los que se deben seguir para la construcción del nodo de Hyperledger Iroha. En caso de que se desee construir el nodo con la distribución existente antes del 12 de Noviembre del 2021, consultar el anexo B.

#### 4.1.1.1 Instalación Python.

1. Ingresar a <https://www.python.org/downloads/>.
2. Descargar la distribución de Python 3 para Windows.
3. Una vez descargado, ejecutar el Python-3.x.x-amd64.
4. Dentro de las opciones de configuración, seleccionar la opción “agregar Python 3.x to PATH”. El resto de la configuración puede realizarse según el criterio del usuario.
5. Comenzar el proceso de instalación.
6. Al terminar el proceso de instalación, la computadora debe reiniciarse para que se complete la instalación de Python correctamente.

#### Instalación git.

1. Ingresar a <https://git-scm.com/downloads>.
2. Descargar la distribución de git para Windows.
3. Una vez descargado, ejecutar el archivo de instalación.
4. Configurar la instalación según el criterio del usuario.
5. Al terminar el proceso de instalación, reiniciar la computadora para que se complete la instalación de git correctamente.

#### 4.1.1.3 Instalación Docker.

1. Ingresar a <https://www.docker.com/products/docker-desktop>.
2. Descargar la distribución de Docker para Windows.
3. Una vez descargado, ejecutar el Docker Desktop Installer.exe.
4. Dentro de las opciones de configuración, seleccionar la opción “instalar el WSL2” (Windows Subsystem for Linux). El resto de la configuración puede realizarse según el criterio del usuario.
5. Comenzar el proceso de instalación.
6. Al terminar el proceso de instalación, reiniciar la computadora.
7. Al volver a encender la computadora aparecerá un mensaje que dice “WSL 2 installation is incomplete”; por lo que será necesario ingresar a <https://aka.ms/wsl2kernel>, buscar el paso número cuatro y descargar el WSL2 Linux kernel update package for x64 machines.   
   **Es importante no cerrar la ventana del aviso.**
8. Una vez descargado, ejecutar el wsl\_update\_x64.msi.
9. Comenzar el proceso de instalación.
10. Al completar la instalación, regresar a la ventana de aviso que apareció en el paso cinco y hacer clic en reiniciar para que se complete la instalación del WSL 2 correctamente.
11. Reiniciar la computadora para que se complete la instalación del Docker Desktop correctamente.
12. Finalmente, al volver a encender la computadora, aparecerá un mensaje solicitando permisos elevados para Docker; aceptar esta solicitud.
13. Ejecutar Docker Desktop.
14. Hacer clic en los ajustes.
15. En la pestaña de general, activar la funcionalidad "Expose daemon".

#### 4.1.1.4 Configuración PostgreSQL en Docker.

1. Ejecutar el símbolo del sistema como administrador.
2. Ingresar el comando: docker network create iroha-network   
   El objetivo de este comando es crear una red a la cual agregar contenedores, para que estos contenedores sean capaces de compartir información entre sí.
3. Ingresar el comando: docker run --name some-postgres -e POSTGRES\_USER=postgres -e POSTGRES\_PASSWORD=mysecretpassword -p 5432:5432 --network=iroha-network -d postgres:9.5  
   El objetivo de este comando es agregar la base de datos de PostgreSQL en un contenedor.

--name se utiliza para asignar el nombre del contenedor.

-e se utiliza para asignar variables del entorno, por lo que en este caso determinaremos el usuario y contraseña necesarios para acceder a la BD de Postgres

-p determina el puerto con el que se comunicará el contenedor.

--network es para añadirlo a la red que definimos con el comando previo.

-d es un parámetro que indica que el contenedor se está iniciando en modo detached; esto significa que al cerrarse el proceso padre, en este caso postgres:9.5, se cierra el contenedor también [29].

1. Esperar a que se complete la descarga de la imagen de postgres:9.5.
2. Al finalizar la descarga, Windows solicitará un permiso de firewall para Docker, es necesario aceptar esta solicitud.
3. Ingresar el comando docker volume create blockstore.   
   El objetivo de este comando es crear un volumen, para que aunque el contenedor se vea modificado, la información almacenada no se pierda.

#### 4.1.1.5 Configuración C++ Build Tools

1. Abrir un navegador web e ingresar la dirección web: <https://visualstudio.microsoft.com/es/visual-cpp-build-tools/>
2. Presionar el botón de “Descargar Build Tools”
3. Una vez descargado, ejecutar el archivo de instalación.
4. Seleccionar el paquete de C++ build tools.
5. Confirmar la descarga e instalación de los paquetes que se seleccionaron automáticamente.
6. Al finalizar el proceso de descarga e instalación, la computadora debe reiniciarse para terminar la instalación correctamente.

#### 4.1.1.6 Configuración Hyperledger Iroha.

1. Ejecutar el símbolo del sistema como administrador.
2. Ingresar el comando C:\Users\nombreUsuario\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip
3. Ingresar el comando pip3 install iroha
4. Ingresar el comando docker run -it --name iroha -p 50051:50051 -p 10001:10001 -v blockstore:/tmp/block\_store --network=iroha-network --entrypoint=/bin/bash hyperledger/iroha-builder:latest  
   El objetivo de este comando es agregar Hyperledger Iroha a un contenedor.  
   -it se utiliza para indicar que el contenedor debe ser interactivo, por lo que Docker asigna un tty (teletype) al contenedor.   
   -v es para indicarle al contenedor que tiene que utilizar el volumen que creamos en el paso número 4.1.1.4.   
   --entrypoint sirve para iniciar el contenedor, en el directorio deseado [29].  
   Al ingresar este comando, el símbolo del sistema automáticamente accederá al contenedor iroha.
5. Ingresar los siguientes comandos:
   1. git clone https://github.com/hyperledger/iroha.git
   2. cd iroha
   3. ./vcpkg/build\_iroha\_deps.sh $PWD/vcpkg-build
   4. cmake -B build -DCMAKE\_TOOLCHAIN\_FILE=$PWD/vcpkg-build/scripts/buildsystems/vcpkg.cmake . -DCMAKE\_BUILD\_TYPE=RELEASE -GNinja -DUSE\_BURROW=OFF -DUSE\_URSA=OFF -DTESTING=ON -DCOVERAGE=ON -DBENCHMARKING=ON -DPACKAGE\_DEB=ON -DPACKAGE\_RPM=ON -DPACKAGE\_TGZ=ON -DPACKAGE\_ZIP=ON  
      Este comando genera un makefile, el cual especifica una serie de instrucciones que se deben de ejecutar. Estas instrucciones tienen como objetivo que la instalación resultante sea compatible con el sistema operativo que se está utilizando y solamente se utilicen las funcionalidades especificadas mediante los parámetros.  
      -B especifica el directorio que se utilizará como la raíz del directorio de construcción [30].  
      -DCMAKE\_TOOLCHAIN\_FILE especifica cuál es el primer archivo que se debe cargar para definir valores para los compiladores [31].  
      -DUSE\_BURROW determina la utilización de la integración con Hyperledger Burrow.  
      -DUSE\_URSA determina la utilización de la integración con Hyperledger Ursa.  
      -DTESTING determina la construcción de las pruebas.  
      -DCOVERAGE determina la activación de la cobertura de código usando lcov.  
      -DBENCHMARKING determina la construcción de las pruebas utilizando la biblioteca de benchmarks de Google.  
      -DPACKAGE\_DEB determina la activación del empaquetamiento deb.  
      -DPACKAGE\_RPM determina la activación del empaquetamiento rpm.  
      -DPACKAGE\_TGZ determina la activación del empaquetamiento tar.gz.  
      -DPACKAGE\_ZIP determina la activación del empaquetamiento zip [32].  
      En caso de que este comando no funcione, será necesario volver a introducir el anterior y luego reintroducir este comando. Esto se debe a que el comando anterior descarga e instala un gran número de elementos, y si se llega a dar un error en la red, no descargará e instalará todo lo necesario.
   5. cd build
   6. cmake --build . --target all  
      Este comando construye el programa deseado con base en el makefile generado en el paso anterior.
   7. ctest . --output-on-failure  
      Este comando ejecuta todas las pruebas que se hayan especificado en el makefile para probar que la construcción se realizó correctamente.  
      Hay tres casos en los que no se pasan todas las pruebas:
      1. Si se aprobó 71% de las pruebas, se debe a que no se desplegó correctamente el contenedor de PostgreSQL, es necesario corregirlo y posteriormente reintroducir este comando.
      2. Si se aprobó el 99% de las pruebas y la que falló fue la prueba 170 “integration\_pipeline\_test”, se debe a que hubo un error en la integración con Hyperledger Ursa y Hyperledger Burrow. Este error se debe a problemas de estabilidad con la paquetería, pero ya que estos elementos no se utilizarán en el desarrollo de este prototipo, este error no afecta a la construcción y se puede ignorar.
      3. Si se aprobó un porcentaje distinto a 71%, 99% o 100%, será necesario regresar al paso número 3 de este apartado (4.1.1.6). Esto se debe a que este comando descarga e instala un gran número de elementos, y si se llega a dar un error en la red, no descargará e instalará todo lo necesario.
   8. cd ..
6. Ejecutar otro símbolo del sistema.
7. Ingresar el comando docker exec -u root -it iroha /bin/bash  
   El objetivo de este comando es acceder al contenedor de Iroha con permisos de root.
8. Ingresar los siguientes comandos:
   1. cd ..
   2. cd ..
   3. cd tmp
   4. cd block\_store
   5. chown -R iroha-ci /tmp/block\_store  
      El objetivo de este comando es cambiar el propietario del directorio de manera recursiva al usuario utilizado normalmente por Iroha.
   6. apt update
   7. apt install iputils-ping
   8. apt install telnet
9. Cerrar este símbolo del sistema.
10. Ingresar en el símbolo del sistema que se encuentra abierto actualmente el comando cd example
11. Ingresar el comando vim genesis.block
12. Substituir todo el contenido del archivo por el contenido del Anexo C
13. Ingresar el comando cd ..
14. Ingresar el comando ./build/bin/irohad --config example/config.docker --genesis\_block example/genesis.block --keypair\_name example/node0 --overwrite\_ledger  
    El objetivo de este comando es ejecutar irohad (daemon de Iroha) utilizando los siguientes parámetros:  
    --config indica el archivo de configuración que usaremos, en este caso usaremos el config.docker ya que estamos utilizando Iroha dentro de Docker  
    --genesis\_block indica el bloque génesis que usaremos  
    --keypair\_name indica el nombre de los archivos .pub y .priv que funcionarán como llaves

Al terminar de realizar estos pasos, se tendrá una implementación del prototipo completamente funcional en un solo nodo. Por lo que la siguiente sección indicará como modificar esta implementación para que el prototipo funcione en más de un nodo.

### 4.1.2 Instalación y configuración de paquetería y herramientas para la construcción de una red con múltiples nodos.

Para toda red que se vaya a construir, es necesario que se encuentren activos al mismo tiempo tres o más nodos en todo momento. De otra manera, aunque se validen las transacciones estas no podrán ser registradas en la cadena de bloques debido a que no se cumple el requisito del mínimo de votos, que son tres.

La configuración a continuación mostrada se realizó utilizando un solo router; sin embargo, en caso de que se posea un router por cada nodo que se desee agregar a la red, no será necesario seguir las instrucciones especificadas en el apartado “4.1.2.3 Configuración del router”.

#### 4.1.2.1 Preparación previa.

Todos los dispositivos que vayan a albergar algún nodo, exceptuando aquel en el que se configuró el nodo original, tienen que seguir los pasos especificados en esta sección.

1. Replicar los pasos especificados en la sección 4.1.1.1
2. Replicar los pasos especificados en la sección 4.1.1.2
3. Replicar los pasos especificados en la sección 4.1.1.3
4. Ejecutar el símbolo del sistema como administrador.
5. Ingresar el comando: docker network create iroha-network
6. Ingresar el comando docker volume create blockstore.
7. Replicar los pasos especificados en la sección 4.1.1.5
8. Ingresar el comando C:\Users\nombreUsuario\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip
9. Ingresar el comando pip3 install iroha

#### 4.1.2.2 Replicación del prototipo en los demás dispositivos.

1. Ejecutar un símbolo del sistema.
2. Introducir el comando docker commit iroha imagenIroha
3. Navegar dentro del símbolo del sistema hasta un directorio donde se tenga un espacio de almacenamiento de al menos 17 gb.
4. Introducir el comando docker save imagenIroha > imagenIroha.tar
5. Esperar a que se termine el proceso de guardado.
6. Cerrar el símbolo del sistema.
7. Copiar el archivo imagenIroha.tar a los demás dispositivos que albergarán un nodo.
8. Ejecutar un símbolo del sistema en los dispositivos a los que se copió el archivo.
9. Introducir el comando docker load < imagenIroha.tar
10. Esperar a que termine el proceso de cargado.
11. Introducir el comando docker run -it --name iroha -p 50051:50051 -p 10001:10001 -v blockstore:/tmp/block\_store --network=iroha-network --entrypoint=/bin/bash -td imagenIroha
12. Ingresar el comando: docker run --name some-postgres -e POSTGRES\_USER=postgres -e POSTGRES\_PASSWORD=mysecretpassword -p 5432:5432 --network=iroha-network -d postgres:9.5  
    Es necesario crear un nuevo contenedor de PostgreSQL posterior a la creación del contenedor de Iroha ya que, de otra manera, el contenedor replicado de Iroha pensará que el contenedor de PostgreSQL es el mismo que el del nodo original y al realizar la conexión fallará, ya que no es el mismo. Por lo que la solución es no dejar que el contenedor replicado de Iroha encuentre el contenedor de PostgreSQL y este se cree después.
13. Cerrar el símbolo del sistema.

#### 4.1.2.3 Configuración del router.

1. Ejecutar un símbolo del sistema.
2. Introducir el comando ipconfig.
3. Copiar la dirección ip de la puerta de enlace predeterminada.
4. Cerrar el símbolo del sistema.
5. Abrir un navegador.
6. Introducir en la barra de búsqueda el texto copiado.
7. Iniciar sesión en el router.
8. Crear reglas de reenvío para los puertos 50051 y 10001 para las direcciones ip de los equipos que vayan a albergar un nodo.
9. Cerrar el navegador.

#### 4.1.2.4 Configuración del firewall.

1. Abrir el firewall de Windows.
2. Acceder a las opciones avanzadas.
3. Activar las dos funcionalidades llamadas "Compartir archivos e impresoras (solicitud eco: ICMPv4 de entrada)".
4. Crear reglas de entrada y salida para los puertos 50051 y 10001
5. Cerrar el firewall de Windows.
6. Ejecutar una terminal de comandos en todos los equipos que vayan a albergar un nodo
7. Generar pings desde cada nodo a todos los demás nodos para confirmar la conectividad entre estos.
8. Iniciar el contenedor Iroha en docker en cada uno de los nodos.
9. Generar llamadas de telnet utilizando el puerto 50051 de cada nodo a todos lo demás para confirmar que se puede acceder a Iroha desde cada nodo.
10. Cerrar la terminal de comandos.

#### 4.1.2.5 Creación de las llaves y modificación del genesis block.

1. Crear un archivo de Python que contenga el siguiente código:

from iroha import IrohaCrypto

llavePrivada = IrohaCrypto.private\_key()

llavePublica = IrohaCrypto.derive\_public\_key(privada)

print(llavePrivada)

print(llavePublica)

1. Ejecutar el código en cada nodo y anotar las llaves generadas.
2. Asignarle a cada nodo un número.
3. En cada nodo crear un archivo de texto llamado nodeN.priv, reemplazando N con el número asignado a este nodo.
4. Copiar la llave privada generada por el código en este archivo.
5. En cada nodo crear un archivo de texto llamado nodeN.pub, reemplazando N con el número asignado a este nodo.
6. Copiar la llave pública generada por el código en este archivo.
7. Revisar que ambos archivos ocupen exactamente 64 bytes.
8. En cada nodo ejecutar un símbolo del sistema.
9. Navegar hasta el directorio en el que se encuentren almacenados los archivos nodeN.priv y nodeN.pub.
10. Introducir el siguiente comando  
     docker cp nodeN.priv iroha:/opt/iroha/iroha/example
11. Introducir el siguiente comando  
     docker cp nodeN.pub iroha:/opt/iroha/iroha/example  
    La razón por la que copiamos el archivo desde el dispositivo al contenedor en vez de crear una copia dentro del contenedor de los archivos ya existentes llamados node0.pub y node0.priv, es debido a que al editarlos utilizando Vim, siempre se agrega el carácter de newline al final del texto por lo que el tamaño del archivo pasa a ser de 65 kb y por lo tanto, no es aceptado como llave.
12. Cerrar el símbolo del sistema.
13. En cada nodo, abrir la terminal del contenedor.
14. Introducir los siguientes comandos:
15. cd iroha
16. cd example
17. vim genesis.block
18. Reemplazar el contenido con el del Anexo D.
19. Reemplazar las direcciones ip y llaves públicas presentes por las direcciones ip y llaves públicas de los nodos que se encuentran en uso.
20. Guardar el archivo.
21. Introducir el comando vim config.docker.
22. Agregar como parámetro lo especificado en el Anexo E.
23. Reemplazar las direcciones ip y llaves públicas presentes por las direcciones ip y llaves públicas de los nodos que se encuentran en uso.
24. Guardar el archivo.
25. Introducir el comando cd ..
26. Introducir el comando ./build/bin/irohad --config example/config.docker --genesis\_block example/genesis.block --keypair\_name example/nodeN --overwrite\_ledger  
    Reemplazando N con el número asignado a este nodo.

### 4.1.3 Instalación y configuración de paquetería y herramientas para la adición de un nuevo nodo a la red de múltiples nodos existente.

Para la adición de un nuevo nodo a un red que ya cuenta con múltiples nodos y se encuentra en funcionamiento no es necesario apagar todos los nodos, repetir en cada uno de estos los pasos para modificar el genesis block y el config.docker con la información correspondiente al nuevo nodo y volverlos a encender.

En vez de esto, para su adición a una red que ya se encuentra establecida, únicamente es necesario que este nuevo nodo tenga el mismo genesis block y config.docker que los nodos originales, y que además, este nuevo nodo sea agregado a la red por alguno de los que ya pertenecen a esta.

Por ello, en el nodo que se quiera agregar a la red se tienen que realizar los siguientes pasos, ajustándolos para que el cambio únicamente se haga en el nuevo nodo:

1. Replicar los pasos especificados en la sección 4.1.2.1
2. Copiar el archivo imagenIroha.tar al dispositivo que albergará el nodo.
3. Ejecutar un símbolo del sistema.
4. Introducir el comando docker load < imagenIroha.tar
5. Esperar a que termine el proceso de cargado.
6. Introducir el comando docker run -it --name iroha -p 50051:50051 -p 10001:10001 -v blockstore:/tmp/block\_store --network=iroha-network --entrypoint=/bin/bash -td imagenIroha
7. Ingresar el comando: docker run --name some-postgres -e POSTGRES\_USER=postgres -e POSTGRES\_PASSWORD=mysecretpassword -p 5432:5432 --network=iroha-network -d postgres:9.5
8. Cerrar el símbolo del sistema
9. Replicar los pasos especificados en la sección 4.1.2.3
10. Replicar los pasos especificados en la sección 4.1.2.4
11. Replicar los pasos especificados en la sección 4.1.2.5
12. Ejecutar el comando “Addpeer”, utilizando la información del nuevo nodo como parámetros, dentro de alguno de los nodos que ya se encuentran dentro de la red.

### 4.1.4 Pausa, inicio y reinicio del ambiente configurado en Docker.

En caso de que se desee detener la ejecución de Docker, lo que se debe hacer es:

1. En la terminal del contendor de Hyperledger Iroha, detener su ejecución. Esto se puede hacer con Ctrl+C, o ^C.
2. Detener el contenedor de Hyperledger Iroha.
3. Detener el contenedor de PostgreSQL.
4. Salir de Docker Desktop.

En caso de que se desee iniciar la ejecución de Docker, lo que se debe hacer es:

1. Abrir Docker Desktop.
2. Iniciar el contenedor de PostgreSQL.
3. Iniciar el contenedor de Hyperledger Iroha.
4. Ejecutar el CLI del contenedor de Hyperledger Iroha.
5. Ingresar el comando irohad --config config.docker --genesis\_block genesis.block --keypair\_name nodeN  
   Reemplazando N con el número asignado a este nodo.

En caso de que se desee reiniciar el blockchain utilizado por el contenedor de Hyperledger Iroha, lo que se debe hacer es:

1. Si se está ejecutando Hyperledger Iroha, detener su ejecución.   
   Es importante no detener la ejecución del contenedor de Hyperledger Iroha.
2. En el CLI del contenedor de Hyperledger Iroha ingresar el comando irohad --config config.docker --genesis\_block genesis.block --keypair\_name nodeN --overwrite\_ledger  
   Reemplazando N con el número asignado a este nodo.

### 4.1.5 Configuración del entorno de trabajo.

La configuración del entorno de trabajo es realizada en el genesis block de la blockchain y en el ambiente de Python.

En el genesis block se crearán y configurarán los nodos iniciales, los roles, el dominio, el activo y los usuarios que contarán con los roles de administración y administración de dinero.

El código de este genesis block se puede consultar en los Anexos C y D.

En el ambiente de Python, se crearán todas las transacciones y consultas necesarias para implementar la funcionalidad requerida por cada dominio y sus correspondientes roles.

El código de este ambiente se pude consultar en el anexo F.

### 4.1.6 Interfaz

A

## 4.2 Especificaciones de software y hardware

A continuación, se mencionarán todos los elementos de software y hardware que fueron utilizados para la implementación de la solución, junto con una breve descripción sobre estos y sobre la función que realizan en el prototipo.

### 4.2.1 Software

A

#### 4.2.1.1 Git

A

#### 4.2.1.2 Docker

A

#### 4.2.1.3 Hyperledger Iroha

A

#### 4.2.1.4 PostgreSQL

A

### 4.2.2 Hardware

A

# 5. PRUEBAS Y RESULTADOS

A

## 5.1 Descripción de las pruebas

A

## 5.2 Resultados

A

## 5.3 Análisis de resultados

A

# 6. CONCLUSIONES

A

# 7. ANEXOS

## 7.1 Anexo A: Algoritmos de consenso

1. Proof of Work.

Como funciona es que para agregar un nuevo bloque al Blockchain, es necesario calcular el hash que correspondería a este nuevo bloque.

Para calcular este hash, la red genera un acertijo que es matemáticamente difícil de resolver, por lo que es necesario dedicar muchos recursos computacionales a su resolución. Al resolver este problema, el nodo propone el hash al resto de los nodos, el cual es fácilmente verificable, y se agrega al Blockchain si el 51% de la red vota a favor. Si se agrega el bloque, el nodo que propuso el bloque recibe una gratificación monetaria [33].

Está basado en el principio de que es sumamente difícil calcular el nuevo hash y fácil de comprobarlo, por lo que la adición de un bloque malicioso es imposible, e intentarlo costaría una gran cantidad de recursos. Mientras que hacerlo correctamente otorga una gratificación monetaria mayor al costo de los recursos.

1. Proof of Weight.

Como funciona es que a cada usuario se le asigna un peso que está basado en la cantidad de dinero que el usuario tiene en su cuenta. Al momento de realizar una votación sobre una transacción, el voto de los usuarios con mayor peso valdrá más que el de los usuarios con un menor peso [34].

Está basado en el principio de que los usuarios que tienen una gran cantidad de recursos invertidos en el sistema son los que menos quieren que el sistema se vea comprometido, por lo que solamente validaran las transacciones honestas.

1. Proof of Stake.

Como funciona es que, para poder validar transacciones, cada usuario debe comprar una cierta cantidad de la criptomoneda y congelar esa cantidad como prueba de su interés. Posteriormente, entre mayor sea la cantidad de criptomoneda que tiene congelada, mayor será la posibilidad de que le toque evaluar una transacción, la cual posteriormente será validada por el resto de la red. En caso de que se apruebe, el costo de transacción que se le cobró al que inició a la transacción se le otorga el nodo [35].

Está basado en que entre mayor sea la cantidad de recursos que un usuario tiene congelados, mayor interés tiene en que el sistema no se vea comprometido a riesgo de perder toda su inversión, por lo que solamente validará transacciones honestas.

1. Delegated Proof of Stake.

Es una variación de Proof of Stake, donde aquellos que tienen interés congelado votan para crear un conjunto de nodos que serán los que se encargarán de realizar la validación. Además, este conjunto de nodos puede trabajar de manera conjunta en agregar bloques, por lo que tiene una mayor velocidad que Proof of Stake [36].

Está basado en lo mismo que Proof of stake, siendo que se cree que los usuarios con dinero congelado siempre votarán por los nodos confiables.

1. Proof of Elapsed Time.

Como funciona es que hay un conjunto de nodos que continuamente generan valores de tiempo al azar y envían estos valores a los nodos conectados a la red, y los nodos deben de decir cual es el valor que les toco; posteriormente, se compara el valor de todos los nodos, y aquel que tenga el menor valor es el que procesará la transacción [37].

Está basado en que, en una red confiable, este algoritmo asegura que todos tengan una oportunidad justa de participar y se gasten pocos recursos.

1. Proof of Burn.

Es una variación de Proof of Stake, donde los usuarios en lugar de congelar cierta cantidad de dinero deben de enviarlo a una dirección de donde no se puede recuperar [38].

Está basado en el principio de que aquellos que quieren una mayor oportunidad de conseguir validar la próxima transacción, deben de pagar más para tener una mayor probabilidad de ser escogidos, y que buscarán solo validar transacciones honestas debido a que, si el sistema deja de ser confiable y se deja de usar, no pueden recuperar nada de su inversión.

1. Proof of Activity.

Es una mezcla de Proof of Work y Proof of Stake, siendo que primero se ejecuta Proof of work de manera normal, y cuando el nodo anuncia el hash que obtuvo, este es validado por un grupo de nodos escogido al azar acorde a la cantidad de dinero que tienen congelado [39].

Está basado en juntar los principios de ambos algoritmos para incrementar su seguridad.

1. Proof of Authority.

Es una variación de Proof of Stake, donde en vez de congelar dinero, se registra el desempeño de cada nodo y se le asigna una calificación de autoridad, siendo que entre mayor sea esa calificación, mayor probabilidad tendrá de ser escogido para procesar la transacción [40].

Está basado en que, para tener una buena autoridad, es necesario haber trabajado una gran cantidad de tiempo, por lo que los usuarios buscarán validar transacciones honestas para tener más probabilidad de ganar dinero, y no desperdiciar el tiempo invertido.

1. Proof of Importance.

Como funciona es que entre mayor sea la capacidad del nodo para realizar transacciones, mayor probabilidad hay de que efectúe la transacción [41].

Está basado en que un usuario que dedique esta cantidad de recursos a trabajar para un sistema, le interesa que el sistema continúe funcionando para seguir ganando dinero.

1. Proof of Capacity o Proofs of Space.

Es una variación de Proof of Importance, siendo que, en lugar de capacidad para realizar transacciones, es almacenamiento disponible. Esta basado en el mismo principio, pero busca dar una alternativa más barata [42].

1. Practical Byzantine Fault Tolerance.

Como funciona es que cada transacción es entregada a todos los miembros que integran una red y de manera individual deben decidir si la transacción es válida o no. Una vez que hayan decidido cada nodo publica su resultado en una ronda de votación, donde según el porcentaje establecido por la red de votos a favor se decide si la transacción es aprobada [43].

Está basado en que para realizar una transacción maliciosa es necesario tener bajo control a al menos el mismo porcentaje de nodos necesarios para aprobar una transacción. Cosa que es poco factible que suceda, lo que asegura que el sistema sea seguro.

1. SIEVE.

Este algoritmo es una expansión del Practical Byzantine Fault Tolerance. El aporte de SIEVE es la capacidad de detectar operaciones no deterministas, es decir, operaciones que tienen un diferente resultado a pesar de tener el mismo input y proceso. Lo que hace con estas operaciones es que, en caso de encontrar un pequeño número de irregularidades en el resultado ofrecido por el algoritmo, elimina estos valores y trata de alcanzar un consenso con los restantes; si el número de irregularidades es mayor, elimina la transacción completa [44].

1. Unique Node List.

Es una variación del Pactical Byzantine Fault Tolerance, siendo que su diferencia está en que cada nodo genera una lista con un subconjunto de nodos. Al procesar una transacción, esta será evaluada y votada únicamente por la lista de nodos que tenga el nodo que propuso la transacción [45]. Se modificó de esta manera para ofrecer una latencia mejor que la del algoritmo original.

1. Raft.

Como funciona es que, dentro del conjunto de nodos, se escoge a un líder, el cual solamente es capaz de registrar cosas en la base de datos, no modificar ni eliminar. Al agregar algo dentro de la base, el líder debe mandar una copia al resto de los nodos para que estén actualizados. El puesto de líder va cambiado de nodo a nodo utilizando parámetros de antigüedad y de latencia con respecto al resto de los nodos [46].

Está basado en que, al tener una red confiable, se puede repartir la carga de trabajo entre los nodos participantes al seleccionar al nodo más capaz para el trabajo y cambiarlo cuando deje de serlo; lo que ofrece mayor velocidad de procesamiento y mayor disponibilidad de la red.

## 7.2 Anexo B: Versión previa al 12/11/21

Instalación Notepad++

1. Ingresar a <https://notepad-plus-plus.org/downloads/>.
2. Descargar la versión más reciente de Notepad ++ para Windows.
3. Una vez descargado, ejecutar el archivo de instalación.
4. Configurar la instalación según el criterio del usuario.

Configuración Hyperledger Iroha.

1. Ejecutar el símbolo del sistema como administrador.
2. Ingresar el comando C:\Users\nombreUsuario\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip
3. Ingresar el comando pip3 install iroha
4. Navegar hasta el directorio C:/Users/nombreUsuario
5. Ingresar el comando git clone -b main https://github.com/hyperledger/iroha --depth=1.
6. Abrir un explorador de Windows.
7. Navegar hasta la dirección C:/Users/nombreUsuario/iroha/scripts.
8. Abrir con Notepad++ el archivo run-iroha-dev.sh.
9. En la pestaña de “Buscar” seleccionar la opción de “Buscar y reemplazar”
10. Colocar en el área de buscar “\r\n” y en el área de reemplazar “\n”.   
    Es necesario realizar este paso debido a que el salto de línea de Windows está especificado como “\r\n” y en Ubuntu como “\n”, por lo que de no realizar esta corrección, el archivo no podrá ser leído correctamente por Ubuntu.
11. Hacer clic en la opción “Reemplazar todo”.
12. Guardar el archivo.
13. Cerrar el archivo.
14. En el símbolo del sistema que tenemos abierto, ingresar el comando bash iroha/scripts/run-iroha-dev.sh.  
    Este comando creará un contenedor en Docker en el que estará contenido Hyperledger Iroha.
15. Ingresar el comando exit
    1. En caso de que se haya apagado el contenedor en Docker debido al comando de exit, volver a encenderlo.
16. Ingresar el comando docker exec -u root -it iroha1000\_node\_1 /bin/bash.  
    Esto se hace para poder acceder al contenedor desde el usuario root y por lo tanto, poseer todos los permisos necesarios para la configuración.
17. Ingresar los siguientes comandos:
    1. apt-get update
    2. apt-get -y --no-install-recommends install build-essential ninja-build git ca-certificates tar curl unzip cmake
    3. apt-get install pkg-config
    4. git clone https://github.com/jupp0r/prometheus-cpp
    5. cd prometheus-cpp
    6. mkdir \_build
    7. git submodule init
    8. git submodule update
    9. cd \_build
    10. cmake .. -DBUILD\_SHARED\_LIBS=ON -DENABLE\_PUSH=OFF -DENABLE\_COMPRESSION=OFF
    11. cmake --build . --parallel 4
    12. ctest -V
    13. cmake --install .  
        Este comando instala la construcción hecha previamente para que todos los demás programas dentro del contenedor puedan tener acceso a sus funcionalidades.
    14. cd ..
    15. cd ..
    16. git clone https://github.com/civetweb/civetweb
    17. cd civetweb
    18. mkdir \_build
    19. cd \_build
    20. cmake ..
    21. cmake --build .
    22. ctest -V  
        En este paso, la prueba 37 “test-publicserver-server-requests” va a fallar, esto es normal debido a que aún no se realiza la conexión con el servidor. Esto no afecta en nada al programa por lo que se puede ignorar.
    23. cmake --install .
    24. cd ..
    25. cd ..
    26. git clone https://github.com/facebook/rocksdb
    27. apt-get install libgflags-dev
    28. apt-get install libsnappy-dev
    29. apt-get install zlib1g-dev
    30. apt-get install liblz4-dev
    31. apt-get install libzstd-dev
    32. cd rocksdb
    33. make shared\_lib
    34. Esperar a que se termine la construcción
    35. Crear un folder en el escritorio
    36. Entrar al directorio "C:\Users\nombreUsuario\iroha\rocksdb" y copiar los archivos llamados "librocksdb.so", "librocksdb.so.6", "librocksdb.so.6.26", "librocksdb.so.6.26.0" al folder en el escritorio.
    37. Regresar al símbolo del sistema y ejecutar el comando make install
    38. Entrar al directorio "C:\Users\nombreUsuario\iroha\rocksdb" y copiar el archivo llamado ""librocksdb.a" al folder en el escritorio.
    39. Regresar al símbolo del sistema y ejecutar el comando mkdir \_build
    40. cd \_build
    41. Copiar los archivos en el folder del escritorio al directorio "C:\Users\nombreUsuario\iroha\rocksdb\\_build"
    42. Eliminar el folder del escritorio
    43. Regresar al símbolo del sistema y ejecutar el comando cmake ..
    44. cmake --install .
    45. cd ..
    46. cd ..
    47. git clone https://github.com/nlohmann/json
    48. cd json
    49. mkdir build
    50. cd build
    51. cmake .. -DJSON\_BuildTests=On
    52. cmake --build .
    53. ctest --output-on-failure
    54. cmake --install .
    55. cd ..
    56. cd ..
    57. cmake -DCMAKE\_TOOLCHAIN\_FILE=/opt/dependencies/scripts/buildsystems/vcpkg.cmake -G "Ninja" ..
    58. cmake --build . --target all -- -j n  
        En este paso, es importante reemplazar la n por el número de núcleos que tiene el sistema.

## 7.3 Anexo C: Genesis block nodo independiente.

{

"block\_v1":{

"payload":{

"transactions":[

{

"payload":{

"reducedPayload":{

"commands":[

{

"addPeer":{

"peer":{

"address":"127.0.0.1:10001",

"peerKey":"bddd58404d1315e0eb27902c5d7c8eb0602c16238f005773df406bc191308929"

}

}

},

{

"createRole":{

"roleName":"admin",

"permissions":[

"can\_create\_account",

"can\_set\_detail",

"can\_create\_asset",

"can\_receive",

"can\_transfer",

"can\_add\_domain\_asset\_qty",

"can\_subtract\_domain\_asset\_qty",

"can\_create\_domain",

"can\_grant\_can\_set\_my\_account\_detail",

"can\_grant\_can\_set\_my\_quorum",

"can\_grant\_can\_transfer\_my\_assets",

"can\_add\_peer",

"can\_remove\_peer",

"can\_append\_role",

"can\_create\_role",

"can\_detach\_role",

"can\_add\_signatory",

"can\_remove\_signatory",

"can\_set\_quorum",

"can\_get\_domain\_acc\_detail",

"can\_get\_domain\_accounts",

"can\_get\_my\_acc\_detail",

"can\_get\_my\_account",

"can\_get\_domain\_acc\_ast",

"can\_get\_my\_acc\_ast",

"can\_get\_domain\_acc\_ast\_txs",

"can\_get\_my\_acc\_ast\_txs",

"can\_get\_all\_acc\_txs",

"can\_get\_domain\_acc\_txs",

"can\_get\_my\_acc\_txs",

"can\_read\_assets",

"can\_get\_blocks",

"can\_get\_roles",

"can\_get\_domain\_signatories",

"can\_get\_my\_signatories",

"can\_get\_my\_txs",

"can\_get\_peers"

]

}

},

{

"createRole":{

"roleName":"user",

"permissions":[

"can\_receive",

"can\_transfer",

"can\_grant\_can\_set\_my\_account\_detail",

"can\_grant\_can\_set\_my\_quorum",

"can\_grant\_can\_transfer\_my\_assets",

"can\_add\_signatory",

"can\_remove\_signatory",

"can\_set\_quorum",

"can\_get\_my\_acc\_detail",

"can\_get\_my\_account",

"can\_get\_my\_acc\_ast",

"can\_get\_my\_acc\_ast\_txs",

"can\_get\_my\_acc\_txs",

"can\_read\_assets",

"can\_get\_my\_signatories",

"can\_get\_my\_txs"

]

}

},

{

"createRole":{

"roleName":"money\_creator",

"permissions":[

"can\_receive",

"can\_transfer",

"can\_set\_quorum",

"can\_get\_my\_account",

"can\_get\_my\_acc\_ast",

"can\_read\_assets",

"can\_add\_domain\_asset\_qty",

"can\_subtract\_domain\_asset\_qty",

"can\_get\_domain\_acc\_detail",

"can\_get\_domain\_accounts",

"can\_get\_domain\_signatories"

]

}

},

{

"createDomain":{

"domainId":"domain",

"defaultRole":"user"

}

},

{

"createAsset":{

"assetName":"sentli",

"domainId":"domain",

"precision":2

}

},

{

"createAccount":{

"accountName":"admin",

"domainId":"domain",

"publicKey":"313a07e6384776ed95447710d15e59148473ccfc052a681317a72a69f2a49910"

}

},

{

"appendRole":{

"accountId":"admin@domain",

"roleName":"admin"

}

}

],

"quorum":1

}

}

}

],

"txNumber":1,

"height":"1",

"prevBlockHash":"0000000000000000000000000000000000000000000000000000000000000000"

}

}

}

## 7.3 Anexo D: Genesis block múltiples nodos.

{

"block\_v1":{

"payload":{

"transactions":[

{

"payload":{

"reducedPayload":{

"commands":[

{

"addPeer":{

"peer":{

"address":"201.137.76.165:10001",

"peerKey":"80bac4a64cda9995f8a5a3c55eb018d247fb7d49bc2e45a7f43be1db5bf06e0e"

}

}

},{

"addPeer":{

"peer":{

"address":"201.137.76.165:10002",

"peerKey":"bddd58404d1315e0eb27902c5d7c8eb0602c16238f005773df406bc191308929"

}

}

},{

"addPeer":{

"peer":{

"address":"201.137.76.165:10003",

"peerKey":"f4d9e0c15e97292e7164f9b0d500b687314fc80644394d79ba33f5d2e9202ee9"

}

}

},

{

"createRole":{

"roleName":"admin",

"permissions":[

"can\_create\_account",

"can\_set\_detail",

"can\_create\_asset",

"can\_receive",

"can\_transfer",

"can\_add\_domain\_asset\_qty",

"can\_subtract\_domain\_asset\_qty",

"can\_create\_domain",

"can\_grant\_can\_set\_my\_account\_detail",

"can\_grant\_can\_set\_my\_quorum",

"can\_grant\_can\_transfer\_my\_assets",

"can\_add\_peer",

"can\_remove\_peer",

"can\_append\_role",

"can\_create\_role",

"can\_detach\_role",

"can\_add\_signatory",

"can\_remove\_signatory",

"can\_set\_quorum",

"can\_get\_domain\_acc\_detail",

"can\_get\_domain\_accounts",

"can\_get\_my\_acc\_detail",

"can\_get\_my\_account",

"can\_get\_domain\_acc\_ast",

"can\_get\_my\_acc\_ast",

"can\_get\_domain\_acc\_ast\_txs",

"can\_get\_my\_acc\_ast\_txs",

"can\_get\_all\_acc\_txs",

"can\_get\_domain\_acc\_txs",

"can\_get\_my\_acc\_txs",

"can\_read\_assets",

"can\_get\_blocks",

"can\_get\_roles",

"can\_get\_domain\_signatories",

"can\_get\_my\_signatories",

"can\_get\_my\_txs",

"can\_get\_peers"

]

}

},

{

"createRole":{

"roleName":"user",

"permissions":[

"can\_receive",

"can\_transfer",

"can\_grant\_can\_set\_my\_account\_detail",

"can\_grant\_can\_set\_my\_quorum",

"can\_grant\_can\_transfer\_my\_assets",

"can\_add\_signatory",

"can\_remove\_signatory",

"can\_set\_quorum",

"can\_get\_my\_acc\_detail",

"can\_get\_my\_account",

"can\_get\_my\_acc\_ast",

"can\_get\_my\_acc\_ast\_txs",

"can\_get\_my\_acc\_txs",

"can\_read\_assets",

"can\_get\_my\_signatories",

"can\_get\_my\_txs"

]

}

},

{

"createRole":{

"roleName":"money\_creator",

"permissions":[

"can\_receive",

"can\_transfer",

"can\_set\_quorum",

"can\_get\_my\_account",

"can\_get\_my\_acc\_ast",

"can\_read\_assets",

"can\_add\_domain\_asset\_qty",

"can\_subtract\_domain\_asset\_qty",

"can\_get\_domain\_acc\_detail",

"can\_get\_domain\_accounts",

"can\_get\_domain\_signatories"

]

}

},

{

"createDomain":{

"domainId":"domain",

"defaultRole":"user"

}

},

{

"createAsset":{

"assetName":"sentli",

"domainId":"domain",

"precision":2

}

},

{

"createAccount":{

"accountName":"admin",

"domainId":"domain",

"publicKey":"313a07e6384776ed95447710d15e59148473ccfc052a681317a72a69f2a49910"

}

},

{

"appendRole":{

"accountId":"admin@domain",

"roleName":"admin"

}

}

],

"quorum":1

}

}

}

],

"txNumber":1,

"height":"1",

"prevBlockHash":"0000000000000000000000000000000000000000000000000000000000000000"

}

}

}

## 7.4 Anexo E: Config.docker

"initial\_peers": [

{

"address":"201.137.76.165:10001",

"public\_key": "80bac4a64cda9995f8a5a3c55eb018d247fb7d49bc2e45a7f43be1db5bf06e0e"

},

{

"address":"201.137.76.165:10002",

"public\_key": "bddd58404d1315e0eb27902c5d7c8eb0602c16238f005773df406bc191308929"

},

{

"address":"201.137.76.165:10003",

"public\_key": "f4d9e0c15e97292e7164f9b0d500b687314fc80644394d79ba33f5d2e9202ee9"

}

]

## 7.5 Anexo F: Ambiente de Python

# SPDX-License-Identifier: Apache-2.0

#

# Iroha python library consists of 3 parts: Iroha, IrohaCrypto and IrohaGrpc

import os

import binascii

import commons

from iroha import IrohaCrypto

from iroha import Iroha, IrohaGrpc

from iroha.primitive\_pb2 import can\_set\_my\_account\_detail

from iroha import primitive\_pb2

import sys

if sys.version\_info[0] < 3:

raise Exception('Python 3 or a more recent version is required.')

# Connection to Iroha network. It uses the address and port of the node it wishes to send the transaction to.

#IROHA\_HOST\_ADDR = os.getenv('IROHA\_HOST\_ADDR', '127.0.0.1')

#IROHA\_PORT = os.getenv('IROHA\_PORT', '50051')

IROHA\_HOST\_ADDR = os.getenv('IROHA\_HOST\_ADDR', '201.137.76.165')

IROHA\_PORT = os.getenv('IROHA\_PORT', '50051')

net = IrohaGrpc('{}:{}'.format(IROHA\_HOST\_ADDR, IROHA\_PORT))

# Here we retrieve the admin@domain account created on the genesis block, we need it to create the iroha object for this account.

# os.getenv(par1, par2) gets the value of par1, if it doesnt exist, then it assigns the value of par2. This is why we use os.getenv insted of: ADMIN\_ACCOUNT\_ID = 'admin@domain'

ADMIN\_ACCOUNT\_ID = os.getenv('ADMIN\_ACCOUNT\_ID', 'admin@domain')

# This creates a new private key

# print(IrohaCrypto.private\_key())

ADMIN\_PRIVATE\_KEY = os.getenv(

'ADMIN\_PRIVATE\_KEY', 'f101537e319568c765b2cc89698325604991dca57b9716b58016b253506cab70')

# The Iroha object allows us to use the Iroha library commands. Each account must have and use its own Iroha object

iroha = Iroha(ADMIN\_ACCOUNT\_ID)

# Here we created a key with the IrohaCrypto.private\_key() method and we will assign it to user as a default value for testing

# We are not creating any user here, just creating the keys.

# To create a user, only the public key is needed, the private key will only be used to sign transactions.

# print(IrohaCrypto.private\_key())

user\_private\_key = os.getenv(

'user\_private\_key', '622e124e078333c58c644f5d107ac8a5c0002aeee222104411355ab10fc0faa8')

user\_private\_key2 = os.getenv(

'user\_private\_key2', '116eac80e88983cabb0b47bcf2be1c0a25222e6aa30ec43bd5dcc3144eaf4c60')

money\_administrator\_private\_key = os.getenv(

'money\_administrator\_private\_key', '61c8067b64855de16e56504b316d06c64652faf1f83cabc8684887cd2682ccc4')

# The derived public key of a private key is always the same.

user\_public\_key = IrohaCrypto.derive\_public\_key(user\_private\_key)

user\_public\_key2 = IrohaCrypto.derive\_public\_key(user\_private\_key2)

money\_administrator\_public\_key = IrohaCrypto.derive\_public\_key(money\_administrator\_private\_key)

# -------------------------------------------------------------------------------------------

# The following functions are used for tracking.

def trace(func):

"""

A decorator for tracing methods' begin/end execution points

"""

def tracer(\*args, \*\*kwargs):

name = func.\_\_name\_\_

print('\tEntering "{}"'.format(name))

result = func(\*args, \*\*kwargs)

print('\tLeaving "{}"'.format(name))

return result

return tracer

@trace

def send\_transaction\_and\_print\_status(transaction):

"""

Sends the transaction and prints its status.

"""

hex\_hash = binascii.hexlify(IrohaCrypto.hash(transaction))

print('Transaction hash = {}, creator = {}'.format(

hex\_hash, transaction.payload.reduced\_payload.creator\_account\_id))

net.send\_tx(transaction)

for status in net.tx\_status\_stream(transaction):

print(status)

# -------------------------------------------------------------------------------------------

# The following functions are used to implement the functionality.

@trace

def create\_domain\_and\_asset(domainName, defaultRole, assetName, assetPrecision, irohaObject, signingPrivateKey):

"""

Creates a domain with its default role, and an asset for this domain with a specified precision.

"""

commands = [

irohaObject.command('CreateDomain', domain\_id=domainName,

default\_role=defaultRole),

irohaObject.command('CreateAsset', asset\_name=assetName,

domain\_id=domainName, precision=assetPrecision)

]

tx = IrohaCrypto.sign\_transaction(

irohaObject.transaction(commands), signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

@trace

def create\_domain(domainName, defaultRole, irohaObject, signingPrivateKey):

"""

Creates a domain with its default role.

"""

commands = [

irohaObject.command('CreateDomain', domain\_id=domainName,

default\_role=defaultRole)

]

tx = IrohaCrypto.sign\_transaction(

irohaObject.transaction(commands), signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

@trace

def create\_asset(domainName, assetName, assetPrecision, irohaObject, signingPrivateKey):

"""

Creates an asset for a specified domain with a specified precision.

"""

commands = [

irohaObject.command('CreateAsset', asset\_name=assetName,

domain\_id=domainName, precision=assetPrecision)

]

tx = IrohaCrypto.sign\_transaction(

irohaObject.transaction(commands), signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

@trace

def add\_asset(assetId, quantity, irohaObject, signingPrivateKey):

"""

Adds a certain quantity of units of the asset to the signing account

"""

tx = irohaObject.transaction([

irohaObject.command('AddAssetQuantity',

asset\_id=assetId, amount=quantity)

])

IrohaCrypto.sign\_transaction(tx, signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

@trace

def create\_account(name, domainName, publicKey, irohaObject, signingPrivateKey):

"""

Create an account and its Iroha object

"""

tx = irohaObject.transaction([

irohaObject.command('CreateAccount', account\_name=name, domain\_id=domainName,

public\_key=publicKey)

])

IrohaCrypto.sign\_transaction(tx, signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

tempIroha = name + '\_iroha'

tempAccountId = name + '@' + domainName

globals()[tempIroha] = Iroha(tempAccountId)

@trace

def create\_role(roleName, permissionList, irohaObject, signingPrivateKey):

"""

Create a role

"""

tx = irohaObject.transaction([

irohaObject.command('CreateRole', role\_name=roleName,

permissions=permissionList)

])

IrohaCrypto.sign\_transaction(tx, signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

@trace

def transfer\_asset\_from\_account\_one\_to\_account\_two(accountOneId, accountTwoId, assetId, transactionDescription, quantity, irohaObject, signingPrivateKey):

"""

Transfer a specific quantity of assets from account number one to account number two.

"""

tx = irohaObject.transaction([

irohaObject.command('TransferAsset', src\_account\_id=accountOneId, dest\_account\_id=accountTwoId,

asset\_id=assetId, description=transactionDescription, amount=quantity)

])

IrohaCrypto.sign\_transaction(tx, signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

@trace

def userone\_grants\_to\_admin\_set\_account\_detail\_permission(grantingAccountId, grantedAccountId, irohaObject, signingPrivateKey):

"""

Make an account able to set details of the signing account

"""

tx = irohaObject.transaction([

irohaObject.command('GrantPermission', account\_id=grantedAccountId,

permission=can\_set\_my\_account\_detail)

], creator\_account=grantingAccountId)

IrohaCrypto.sign\_transaction(tx, signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

@trace

def set\_detail\_to\_account(accountId, objectKey, objectValue, irohaObject, signingPrivateKey):

"""

Set age to an account by signing account

"""

tx = irohaObject.transaction([

irohaObject.command('SetAccountDetail',

account\_id=accountId, key=objectKey, value=objectValue)

])

IrohaCrypto.sign\_transaction(tx, signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

@trace

def set\_role(accountId, roleName, irohaObject, signingPrivateKey):

"""

Set a role to an account by a signing account

"""

tx = irohaObject.transaction([

irohaObject.command('AppendRole',

account\_id=accountId, role\_name=roleName)

])

IrohaCrypto.sign\_transaction(tx, signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

@trace

def get\_asset\_info(assetId, irohaObject, signingPrivateKey):

"""

Get all the information about an asset

"""

query = irohaObject.query('GetAssetInfo', asset\_id=assetId)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

data = response.asset\_response.asset

print('Asset id = {}, precision = {}'.format(data.asset\_id, data.precision))

@trace

def get\_account\_assets(accountId, irohaObject, signingPrivateKey):

"""

List all the assets of an account

"""

query = irohaObject.query('GetAccountAssets', account\_id=accountId)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

data = response.account\_assets\_response.account\_assets

for asset in data:

print('Asset id = {}, balance = {}'.format(

asset.asset\_id, asset.balance))

@trace

def get\_roles(irohaObject, signingPrivateKey):

"""

List all the roles

"""

query = irohaObject.query('GetRoles')

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

data = response.roles\_response.roles

for role in data:

print(role)

@trace

def get\_role\_permissions(roleName, irohaObject, signingPrivateKey):

"""

List all the permissions for a role

"""

query = irohaObject.query('GetRolePermissions', role\_id=roleName)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

data = response.role\_permissions\_response.permissions

for permission in data:

print(permission)

@trace

def get\_account\_transactions(accountId, irohaObject, signingPrivateKey):

"""

List all the transactions of the account.

It's important to remember that a transaction can be comprised by one or more commands.

"""

query = irohaObject.query('GetAccountTransactions',

account\_id=accountId, page\_size=1)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

if response.transactions\_page\_response.all\_transactions\_size == 0:

print('There are no transactions for this account')

else:

print('tx\_hash: {}'. format(response.transactions\_page\_response.next\_tx\_hash))

print(response)

while response.transactions\_page\_response.next\_tx\_hash:

print('tx\_hash: {}'. format(

response.transactions\_page\_response.next\_tx\_hash))

query = irohaObject.query('GetAccountTransactions', account\_id=accountId,

page\_size=1, first\_tx\_hash=response.transactions\_page\_response.next\_tx\_hash)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

print(response)

@trace

def get\_account\_asset\_transactions(accountId, assetId, irohaObject, signingPrivateKey):

"""

List all the transactions of a particular asset for an account.

It's important to remember that a transaction can be comprised by one or more commands.

"""

query = irohaObject.query('GetAccountAssetTransactions',

account\_id=accountId, asset\_id=assetId, page\_size=1)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

if response.transactions\_page\_response.all\_transactions\_size == 0:

print('There are no transactions of this particular asset for this account')

else:

print('tx\_hash: {}'. format(response.transactions\_page\_response.next\_tx\_hash))

print(response)

while response.transactions\_page\_response.next\_tx\_hash:

print('tx\_hash: {}'. format(

response.transactions\_page\_response.next\_tx\_hash))

query = irohaObject.query('GetAccountAssetTransactions', account\_id=accountId, asset\_id=assetId,

page\_size=1, first\_tx\_hash=response.transactions\_page\_response.next\_tx\_hash)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

print(response)

@trace

def get\_transaction\_data(transactionHash, irohaObject, signingPrivateKey):

"""

List all the information of a transaction

"""

query = irohaObject.query('GetTransactions', tx\_hashes=transactionHash)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

data = response.transactions\_response.transactions

for transaction in data:

print(transaction)

@trace

def get\_transactions\_data(transactionHashes, irohaObject, signingPrivateKey):

"""

List all the information of multiple transactions

"""

for tempHash in transactionHashes:

hash = [str(tempHash)]

query = irohaObject.query('GetTransactions', tx\_hashes=hash)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

print(response)

@trace

def get\_account\_details(accountId, irohaObject, signingPrivateKey):

"""

Get all the kv-storage entries for an account

"""

query = irohaObject.query('GetAccountDetail', account\_id=accountId)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

data = response.account\_detail\_response

print('Account id = {}, details = {}'.format(accountId, data.detail))

@trace

def get\_account(accountId, irohaObject, signingPrivateKey):

"""

Get account information.

"""

query = irohaObject.query('GetAccount', account\_id=accountId)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

print(response)

@trace

def add\_peer(accountID, addressAndPort, key, irohaObject, signingPrivateKey):

"""

Add a peer to the network

"""

peer1 = primitive\_pb2.Peer()

peer1.address = addressAndPort

peer1.peer\_key = key

tx = irohaObject.transaction([

irohaObject.command('AddPeer', peer=peer1)

], creator\_account=accountID, quorum=1)

IrohaCrypto.sign\_transaction(tx, signingPrivateKey)

send\_transaction\_and\_print\_status(tx)

@trace

def get\_peers(accountID, irohaObject, signingPrivateKey):

query = irohaObject.query('GetPeers', creator\_account=accountID)

IrohaCrypto.sign\_query(query, signingPrivateKey)

response = net.send\_query(query)

print(response)

# REFERENCIAS

[1] *Anuario de migración y remesas México 2020*, 1st ed. Ciudad de México: BBVA, Gobierno de México, CONAPO, 2020, pp. 115 - 131.

[2] *Compilación de informes trimestrales correspondientes al año 2019*, 1st ed. Ciudad de México: BANXICO, 2020, pp. 1-81, 351 - 413.

[3] R. Maldonado, *Reunión de expertos “Uso productivo de las remesas familiares en América Latina y el Caribe”*, 1era edición. Ciudad de México: CEMLA, CEPAL, BID LAB, 2020, pp. 1-50.

[4] F. Gutiérrez, "Criptomonedas ya representan 6% de las remesas que llegan a México: Bitso", El economista, 2020.

[5] A. Canales, "Remesas y desarrollo en México. Una visión crítica desde la macroeconomía", Papeles de población, vol. 12, número 50, 2006. Disponible en: http://www.scielo.org.mx/scielo.php?script=sci\_arttext&pid=S1405-74252006000400009. [Consultado el 25 de octubre del 2020].

[6] M. Gericke, R. Steyer, M. Bulanda and K. Lowe, "Community Currencies Trade facilitation and community development in the age of blockchain & DLT", Scrypt media.

[7] M. Hayashi, "JAPAN’S FUREAI KIPPU TIME-BANKING IN ELDERLY CARE: ORIGINS, DEVELOPMENT, CHALLENGES AND IMPACT.", International Journal of Community Currency Research, vol. 16, 2012. Available: https://ijccr.files.wordpress.com/2012/08/ijccr-2012-hayashi.pdf. [Accessed 2 February 2021].

[8] R. Kosnin and M. Safri, GOLD DINAR AWARENESS IN MALAYSIA: AN EMPIRICAL STUDY, 1st ed. Kelantan: Universiti Teknologi MARA, 2016.

[9] T. Dezyn, "Community Currencies in a Development Context The case of the Sarafu-Credit in Kenya", MASTER OF CULTURAL ANTHROPOLOGY AND DEVELOPMENT STUDIES, KU LEUVEN, 2017.

[10] E. Westervelt, "NPR Cookie Consent and Choices", Npr.org, 2021. [Online]. Available: https://www.npr.org/2010/07/13/128462416/from-stalwart-to-skeptic-germany-rethinks-eu-role. [Accessed: 02- Feb- 2021].

[11]"2.1.19. Entity-relationship model — Hyperledger Iroha documentation", Iroha.readthedocs.io, 2020. [Online]. Available: https://iroha.readthedocs.io/en/stable/concepts\_architecture/er\_model.html?highlight=domain. [Accessed: 03- Apr- 2021].

[12]"2.2. What’s inside Iroha? — Hyperledger Iroha documentation", Iroha.readthedocs.io, 2020. [Online]. Available: https://iroha.readthedocs.io/en/main/concepts\_architecture/architecture.html?highlight=torii. [Accessed: 03- Apr- 2021].

[13]"What is Ethereum? | ethereum.org", ethereum.org, 2021. [Online]. Available: https://ethereum.org/en/what-is-ethereum/. [Accessed: 18- Mar- 2021].

[14]D. Urbančok, Blockchain open-source software comparison, 1st ed. Brno: Masaryk University Faculty of Informatics, 2019.

[15]"Blockchain Technology Projects – Hyperledger", Hyperledger, 2020. [Online]. Available: https://www.hyperledger.org/use/distributed-ledgers. [Accessed: 18- Mar- 2021].

[16]"Hyperledger Besu - Hyperledger Besu - Hyperledger Confluence", Wiki.hyperledger.org, 2021. [Online]. Available: https://wiki.hyperledger.org/display/besu. [Accessed: 18- Mar- 2021].

[17]"Hyperledger Burrow - Hyperledger Burrow - Hyperledger Confluence", Wiki.hyperledger.org, 2019. [Online]. Available: https://wiki.hyperledger.org/display/burrow. [Accessed: 18- Mar- 2021].

[18]"Hyperledger Sawtooth - Hyperledger Sawtooth - Hyperledger Confluence", Wiki.hyperledger.org, 2021. [Online]. Available: https://wiki.hyperledger.org/display/sawtooth. [Accessed: 18- Mar- 2021].

[19]"Hyperledger Fabric - Hyperledger Fabric - Hyperledger Confluence", Wiki.hyperledger.org, 2021. [Online]. Available: https://wiki.hyperledger.org/display/fabric. [Accessed: 18- Mar- 2021].

[20]"IEEE 802.3-2018 - IEEE Standard for Ethernet", *Standards.ieee.org*, 2018. [Online]. Available: https://standards.ieee.org/standard/802\_3-2018.html. [Accessed: 05- Mar- 2021].

[21]"IEEE 802.11-2020 - IEEE Standard for Information Technology--Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks--Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", *Standards.ieee.org*, 2020. [Online]. Available: https://standards.ieee.org/standard/802\_11-2020.html. [Accessed: 05- Mar- 2021].

[22]"ISO/IEC 14882:2020", ISO, 2020. [Online]. Available: https://www.iso.org/standard/79358.html. [Accessed: 22- Apr- 2021].

[23]"SQL Conformance", PostgreSQL Documentation, 2021. [Online]. Available: https://www.postgresql.org/docs/8.3/features.html. [Accessed: 22- Apr- 2021].

[24]"The GNU General Public License v3.0- GNU Project - Free Software Foundation", Gnu.org, 2007. [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.html. [Accessed: 22- Apr- 2021].

[25]"The MIT License | Open Source Initiative", Opensource.org. [Online]. Available: https://opensource.org/licenses/MIT. [Accessed: 22- Apr- 2021].

[26]"History and License — Python 3.9.4 documentation", Docs.python.org, 2021. [Online]. Available: https://docs.python.org/3/license.html. [Accessed: 22- Apr- 2021].

[27]"Apache License, Version 2.0", Apache.org, 2019. [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0. [Accessed: 22- Apr- 2021].

[28]Certified Kubernetes Conformance Program – Terms and Conditions, 1st ed. Kubernetes, 2018.

[29]"Docker run reference", Docker Documentation, 2021. [Online]. Available: https://docs.docker.com/engine/reference/run/. [Accessed: 28- Nov- 2021].

[30]"cmake(1) — CMake 3.22.0 Documentation", Cmake.org, 2021. [Online]. Available: https://cmake.org/cmake/help/latest/manual/cmake.1.html. [Accessed: 28- Nov- 2021].

[31]"cmake-toolchains(7) — CMake 3.22.0 Documentation", Cmake.org, 2021. [Online]. Available: https://cmake.org/cmake/help/latest/manual/cmake-toolchains.7.html. [Accessed: 28- Nov- 2021].

[32]"5. Building Iroha — Hyperledger Iroha documentation", Iroha.readthedocs.io, 2021. [Online]. Available: https://iroha.readthedocs.io/en/develop/build/index.html. [Accessed: 28- Nov- 2021].

[33] M. Jakobsson and A. Juels, *PROOFS OF WORK AND BREAD PUDDING PROTOCOLS (EXTENDED ABSTRACT)*. Boston: Springer, 1999.

[34] W. Maung Maung Thin, N. Dong, G. Bai and J. Song Dong, *Formal Analysis of a PoS Blockchain*, 1st ed. Singapore Institute of Technology and National University of Singapore.

[35] F. Schuh and D. Larimer, *BITSHARES 2.0: GENERAL OVERVIEW*. Blacksburg: Cryptonemex.

[36] D. Bucci, *Blockchain and Its Emerging Role in Health IT and Health-related Research*. Hillsboro: Intel.

[37] *A Peer-to-Peer Crypto-Currency with Proof-of-Burn “Mining without Powerful Hardware”*, 1st ed. Slimcoin, 2014.

[38] I. Bentov, C. Lee, A. Mizrahi and M. Rosenfeld, *Proof of Activity: Extending Bitcoin’s Proof of Work via Proof of Stake*.

[39] V. Arasev, "POA Network Whitepaper", *GitHub*, 2018. [Online]. Available: https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper. [Accessed: 27- Feb- 2021].

[40] *NEM Technical Reference*, 1st ed. NEM, 2018.

[41] S. Dziembowski, S. Faust, V. Kolmogorov and K. Pietrzak, *Proofs of Space*, 1st ed. University of Warsaw, Ruhr-University Bochum, IST Austria, 2013.

[42] M. Castro and B. Liskov, *Practical Byzantine Fault Tolerance*, 1st ed. Cambridge: Massachussets Institute of Technology, 1999.

[43] M. Nabi-Abdolyouseﬁ and M. Mesbahi, *A Sieve Method for Consensus-typeNetwork Tomography*, 1st ed. Seattle: University of Washington, 2011.

[44] D. Schwartz, N. Youngs and A. Britto, *The Ripple Protocol Consensus Algorithm*, 1st ed. Ripple.

[45] D. Ongaro and J. Ousterhout, *In Search of an Understandable Consensus Algorithm (Extended Version)*, 1st ed. California: Stanford University, 2014.

[46]"IEEE 802.3-2018 - IEEE Standard for Ethernet", *Standards.ieee.org*, 2018. [Online]. Available: https://standards.ieee.org/standard/802\_3-2018.html. [Accessed: 05- Mar- 2021].