



Instalación, configuración & administración.

Manual usuario

versión 1.0 Beta

Julio 2020.

MiniBlocklyChain es una marca registrada de OpenQbit Inc, bajo licencia de uso libre y comercial. Términos y condiciones de uso en: www.OpenQbit.com

Contenido

1.	Introducción.....	3
2.	¿Qué es una red pública o privada en el esquema blockchain?	4
3.	¿Qué es la programación Blockly?	4
4.	¿Qué es Termux?.....	5
5.	¿Qué es Mini BlocklyChain?	5
6.	Arquitectura de procesos en Mini BlocklyChain	9
7.	Diagrama de funcionalidad de la cadena de bloques (Mini BlocklyChain).	12
8.	¿Qué es Mini BlocklyCode?	13
9.	Instalación de red comunicaciones de Mini BlocklyChain	14
10.	Sincronización en nodos del sistema (Teléfono móvil) minutos y segundos.....	33
11.	Configuración de almacenamiento “storage” dentro de Termux.....	40
12.	Instalación de red “Tor” e instalación de “Syncthing”	41
13.	Instalación de Base de datos “Redis” y servidor SSH (Secure Shell).	42
14.	Configuración de servidor SSH en teléfono móvil (smartphone).....	43
15.	Configuración de red “Tor” con servicio de SSH (Secure Shell).	49
16.	Configuración de sistema “Peer to Peer” con Syncthing de forma manual.....	52
17.	Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).	63
18.	¿Qué es Prueba de Cuanto (PQu - Proof Quantum)?.....	64
19.	Definición y uso de bloques en Mini BlocklyChain.....	67
20.	Uso de bloques para base de datos SQLite (versión MiniSQLite).	93
21.	Definición y uso de bloques de seguridad.....	97
22.	Configuración de parámetros de seguridad en Mini BlocklyChain.	107
23.	Anexo “Creación de bases de datos KeyStore & PublicKeys”.....	111
24.	Anexo “RESTful SQLite comandos GET/POST”.....	123
25.	Anexo “Conector Código Java SQLite-Redis”.....	129
26.	Anexo “Mini BlocklyChain para desarrolladores”.....	132
27.	Anexo “BlocklyCode Smart Contracts”.....	144
28.	Anexo “Computación Cuántica con OpenQbit”.....	150
29.	Anexo “Bloques extendidos para base de datos SQLite”.....	154
30.	Anexo “Ejemplo de creación de sistema Mini BlocklyChain”.....	154
31.	Anexo “Integracion con ambientes Ethereum & Bitcoin”.	178
32.	Licenciamiento y uso de software.....	197

1. Introducción.

La blockchain generalmente se asocia con el Bitcoin y otras criptomonedas, pero estas son solo la punta del iceberg ya que no solo se usa únicamente para el caso de dinero digital, sino puede ser usado para cualquier información que pueda tener un valor para usuarios y/o empresas. Y es que esta tecnología, que tiene sus orígenes en 1991, cuando Stuart Haber y W. Scott Stornetta describieron el primer trabajo sobre una cadena de bloques asegurados criptográficamente, no fue notoria hasta 2008, cuando se hizo popular con la llegada del bitcoin. Pero actualmente su utilización está siendo demandada en otras aplicaciones comerciales y se proyecta un crecimiento en el futuro mediano en varios mercados, como el de las instituciones financieras o el de Internet de las Cosas IoT entre otros sectores.

La cadena de bloques, más conocida por el término en inglés blockchain, es un registro único, consensuado y distribuido en varios nodos (dispositivos electrónicos como PC, smartphones, tabletas, etc) de una red. En el caso de las criptomonedas, podemos pensarlo como el libro contable donde se registra cada una de las transacciones.

Su funcionamiento puede resultar complejo de entender si profundizamos en los detalles internos de su implementación, pero la idea básica es sencilla de seguir.

En cada bloque se almacena:

- 1.- una cantidad de registros o transacciones válidas,
- 2.- información referente a ese bloque,
- 3.- su vinculación con el bloque anterior y el bloque siguiente a través del hash de cada bloque –un código único que sería como la huella digital del bloque.

Por lo tanto, **cada bloque tiene un lugar específico e inamovible dentro de la cadena**, ya que cada bloque contiene información del hash del bloque anterior. La cadena completa se guarda en cada nodo de la red que conforma la blockchain, por lo que **se almacena una copia exacta de la cadena en todos los participantes de la red**.

A medida que se crean nuevos registros, estos son primeramente verificados y validados por los nodos de la red y luego añadidos a un nuevo bloque que se enlaza a la cadena.

Ahora imagina que esta red de dispositivos que se comunican entre ellos, tiene la capacidad de interactuar sin intervención de una persona, es decir a gran ventaja de la blockchain es que puede tomar decisiones autónomas, lo que beneficia en tiempos de respuesta de servicio a usuarios, disponible las 24 horas todos los días, minimiza costos en las empresas y antes de todo tiene un nivel de seguridad ya probado, por esto y otras razones se ha hecho tan popular su uso en diferentes sectores públicos y privados.

2. ¿Qué es una red pública o privada en el esquema blockchain?

Red pública. - Es una red de computadoras o dispositivos móviles que se comunican entre ellos y que conservan el anonimato, no se conocen quienes interactúan en esta red de una manera formal en sus transacciones, en este tipo de redes cualquier persona o empresa puede interactuar y conectarse en cualquier momento ya que no se necesitan permisos para conectarse, un ejemplo es la blockchain de Bitcoin, cualquiera puede entrar a comprar o vender. Normalmente este tipo de redes están orientadas o dirigidas a la compra y venta de activos monetarios digitales o su sinónimo “criptomonedas”, ejemplos: DogCoin, Ethereum, LiteCoin, BitCoin, Waves, etc.

Red privada. - Es una red de computadoras o dispositivos móviles que se comunican entre ellos, sin embargo, a diferencia de las redes públicas, las privadas se necesitan tener una autorización previa de alguna entidad (empresa o persona) para poder conectarse y ser parte de este tipo de red. Normalmente las redes privadas de blockchain son usadas en empresas o corporativos para realizar transacciones u operaciones con una variedad de diferentes tipos de información que puede tener un valor tangible en forma de documentos, procesos, autorizaciones y/o decisiones de negocio aplicadas y supervisadas por blockchain, ejemplos: sector financiero, sector seguro, gobierno, entre otros.

3. ¿Qué es la programación Blockly?

Blockly es un **lenguaje de programación visual** compuesto por un sencillo conjunto de comandos que podemos combinar como si fueran las piezas de un rompecabezas. Es una herramienta muy útil para el que quiera **aprender a programar** de una forma intuitiva y simple o para quien ya sabe programar y quiera ver el potencial de este tipo de programación.

Blockly es una forma de programación en donde no se necesita algún antecedente de ningún tipo de lenguaje de computación o informática, esto se debe a que únicamente es unir bloques gráficos como si estuviéramos jugando lego o un rompecabezas, solo se necesita tener un poco de lógica y ¡Listo!!!

Cualquiera puede crear programas para teléfonos móviles (smartphones) sin meterse con esos lenguajes de programación difíciles de entender, solo arma bloques de forma gráfica de forma sencilla, fácil y rápida de crear.

4. ¿Qué es Termux?

Termux es un emulador de terminal Android y una aplicación de entorno Linux que funciona directamente sin necesidad de enrutamiento o configuración. Un sistema base mínimo se instala automáticamente.

Usaremos Termux por su estabilidad y fácil instalación y manejo, sin embargo, se puede usar un ambiente instalado de Linux Ubuntu para Android.

En este ambiente de Linux tendrá el “core” de los procesos de comunicación del MiniBlocklyChain.

5. ¿Qué es Mini BlocklyChain?

Mini BlocklyChain es un blockchain totalmente funcional es una tecnología desarrollada para teléfonos móviles con OS (Sistema Operativo) **Android** que serán los nodos que realizarán en el envío y recepción de transacciones. Creamos la primera tecnología de blockchain que está estructurada de forma “modular” a través de la programación Blockly en donde cualquier persona con unos mínimos conocimientos y sin saber programar podrá crear y desarrollar programas para teléfonos móviles y crear su propio blockchain ya sea en modo de red pública o privada. Si desea crear su propia moneda digital lo podrá hacer o una solución para usarla en una empresa, las posibilidades son en base a las necesidades de cada caso real y a la lógica de negocios que el usuario adopte o cree según sus requerimientos.

Mini BlocklyChain es el primer blockchain modular para teléfonos móviles en donde se puede implementar un sistema transaccional en poco tiempo.

Antes de entrar a la definición y uso de bloques “módulos” se necesitamos tener los conceptos básicos de los componentes de Mini BlocklyChain para saber cuándo, cómo y dónde aplicarlos según el caso real que deseamos implementar.

Los siguientes conceptos hacen hincapié para que tipo de usuario están dirigidos, por lo que las personas que no tengan conocimientos de programación no es importante que sean comprendidos en su totalidad los conceptos para usuarios de desarrollo.

Conceptos básicos:

Nodo. – Cada dispositivo móvil (teléfonos, tabletas, etc) con sistema operativo Android que sea integrante de la red pública o privada de Mini BlocklyChain es nombrado como un nodo de esta.

Hash. – Es una firma digital que se asocia a un conjunto de datos, cadena de caracteres, documentos o algún tipo de información digital, esta firma digital es única e irrepetible la

cual ayuda a enviar o recibir información de manera que no se pueda alterar el contenido inicial de información enviada.

Activo. – Cualquier tipo de dato o información digital que pueda ser ponderado con algún valor tangible o intangible para personas o empresas. (documentos, monedas digitales, música, video, imágenes, autorizaciones digitales, etc).

Transacción. – Intercambio de información entre nodos ocupando algún tipo de activo.

Transacción UXTO. - Es un tipo de transacción que empaqueta una o varias transacciones de los nodos y son todas las transacciones no gastadas de cada nodo (UXTO: Unspent Transaction Outputs) estas pueden ser gastada como input en una nueva transacción. Estas transacciones se agrupan en una cola para ser procesadas cada determinado tiempo que se puede regular según sean los requerimientos de cada flujo de información.

Transacción (input). - Es un tipo de transacción que se basa en las transacciones UXTO, cuando entra una cola de una transacción UXTO, esta es procesada por una transacción **input** esta clasifica la transacción como un deposito o un gasto y así poder tener un balance del resultado final para cada usuario.

Dirección origen. - Es la dirección de quien genera o envía la transacción para ser procesada en la cola de SQLite Master. Es un número alfanumérico de 64 caracteres que es creado y verificado por el sistema.

Dirección destino. -Es la dirección de quien recibe la transacción y se suma a su balance o almacena el activo enviado. Es un número alfanumérico de 64 caracteres que es creado y verificado por el sistema.

SQLite Master. - Base de datos API REST que recibe las transacciones y crea una cola para ser procesada cada determinado tiempo variable o fijo según sea la necesidad del negocio.

Transacción DataBase. - Son las transacciones que son reflejadas en la base de datos SQLite que reserva o almacena información de transacciones de Mini BlocklyChain.

AES (Advanced Encryption Standard). - Es un proceso de seguridad que cifra los datos que son almacenados en la base de datos SQLite de Mini BlocklyChain.

Balance. – Después de realizar cualquier proceso de transacción en el Mini BlocklyChain se obtiene un balance de la operación ya sea como valor tangible (criptomoneda) o valor intangible (procesos de negocio entre personas o empresas).

Proceso de negocio. – Es cualquier proceso que involucra algún tipo de flujo de información para obtener un resultado a un usuario final, el usuario final puede ser un persona o empresa de una diversidad de sectores en el ámbito privado o público.

Llave pública y privada. – Es un tipo de cifrado de información en donde se genera dos claves alfanuméricas asociadas entre ellas y es usado para enviar información sensible a través de

redes públicas o privadas. La llave privada es usara para cifrar información y al enviarla a través de la red no pueda ser alterada o ser legible a simple vista, la llave publica es la que se comparte y es vista por cualquier persona o empresa y esta ayudara a verificar la información enviada, así como para poder leerla únicamente por el destinatario valido.

Firma digital. – Es una firma que puede ser creada con la llave privada, con esta firma el destinatario se asegura que la información no ha sido alterada y se confirma que la información es para el destinatario valido.

Dirección digital (Dirección Mini BlocklyChain). – Es una dirección que se compone de caracteres alfanuméricos única para cada usuario de Mini BlocklyChain, esta dirección en utilizada para realizar las transacciones entre usuarios y sea red pública o privada.

Magic Number. - Es un número aleatorio definido por reglas de negocio de cada transacción UXTO ejecutándose que sirve para dar autorización a que el nodo pueda ser un candidato para ejecutar la transacción UXTO y crear un nuevo bloque el Mini BlocklyChain.

Creacion de un nuevo bloque. – Un nuevo bloque en Mini BlocklyChain es un hash creado y anexado por el nodo que salió como candidato ganador a procesar la transacción UXTO.

Protocolo de consenso PoW (prueba de trabajo). – Es una prueba que ejecuta todos los nodos y el primer nodo que finaliza la prueba exitosamente es el elegido para ejecutar la transacción UXTO. Es un algoritmo que está compuesto de cálculos matemáticos para obtener un resultado (hash) según reglas dados en cada transacción ejecutada por Mini BlocklyChain está basado en el desgaste o demanda de los recursos de procesamiento de información de computadoras, en el caso de Mini BlocklyChain se ha estructurado y modificado para obtener un “magic number” este es un número para poder tener autorización o consenso de la mayoría de los nodos para ser candidato a ser el que pueda ejecutar la transacción UXTO. El PoW tiene establecido un nivel de dificultad máximo de 5 ya que únicamente es usado para obtener el “magic number”.

Protocolo de consenso PoQu (prueba cuántica). – Es una prueba que ejecutar todos los nodos y que está compuesta inicialmente por el PoW para obtener el “magic number” y posteriormente ejecuta un algoritmo de probabilidad basado en un QRNG (Quantum Random Number Generator) un generador de números aleatorios cuánticos, este proceso asegura la igualdad de probabilidad para todos los nodos y así elegir que nodo será el que ejecuta la transacción UXTO y la creación del nuevo bloque.

Árbol de merkle. – Este es un método de seguridad para asegurar a Mini BlocklyChain que las transacciones son válidas u no han sido modificadas por alguna entidad externa. Un árbol hash de Merkle o árbol de merkle o árbol hash es una estructura de datos en árbol, binario o no, en el que cada nodo que no es una hoja está etiquetado con el hash de la concatenación de las etiquetas o valores de sus nodos hijo. Son una generalización de las listas hash y las cadenas hash.

Redis DB. - Base de datos de transacciones en tiempo real usada para procesar y enviar a los nodos las nuevas transacciones que han sido solicitadas.

SQLite DB. - Base de datos donde se almacenan los balances y los nuevos bloques para Mini BlocklyChain que aseguran la integridad de la red.

Centinela. - Conector de seguridad e integridad de datos entre Redis y SQLite. Este conector o programa es el encargado de revisar, procesar, validar, distribuir y traducir las transacciones hacia los nodos.

OpenSSH. - Conector de seguridad para ejecutar tareas dentro del sistema operativo Android.

Terminal Shell Termux. - Programa donde se encuentra dependencias de terceros para procesar las transacciones de Mini BlocklyChain, en esta versión 1.0 se usa por facilidad de instalación sin embargo en futuras versiones será reemplazado por el sistema BlocklyShell que está actualmente en desarrollo.

Cartera o “Wallet”. - Es el repositorio digital en donde se guardan dos datos fundamentales en cualquier transacción; las claves públicas y privadas que serán usadas como dirección origen y firma digital respectivamente para cualquier transacción realizada, así como el balance de las transacciones enviadas o recibidas. Es un repositorio donde se guardan las direcciones Mini BlocklyChain, así como los resultados de las transacciones UXTO (Balance).

Desarrollador o programador de aplicaciones:

Programación Orientada a Objetos (Java). - Mini BlocklyChain está realizado en Java.

BlocklyCode. - Es el término de los contratos inteligentes que pueden ser ejecutados en el ambiente Mini BlocklyChain, los BlocklyCode están creados en programación java.

OpenJDK para Android. - Es la suite de JDK y JRE para Android donde se crean y ejecutan los BlocklyCode.

QRNG (Quantum Random Number Generator). - Es un generador de números aleatorios cuánticos, Mini BlocklyChain cuenta actualmente con dos API para la generación de estos.

rsync. - Es una aplicación libre para sistemas de tipo Unix y Microsoft Windows que ofrece transmisión eficiente de datos incrementales, que opera también con datos comprimidos y cifrados.

ffsend. - Es una aplicación para compartir archivos de forma fácil y segura desde la línea de comandos.

NTP. - Network Time Protocol, es un protocolo de Internet para sincronizar los relojes de los sistemas informáticos a través del enrutamiento de paquetes en redes con latencia variable.

Termux (development). - Terminal Shell con un amplio catálogo de aplicaciones opensource y librerías.

Módulos Blockly (datos). – Los desarrolladores pueden integrar módulos para mejorar las características de Mini BlocklyChain.

Peer to Peer. - Este término esta referenciado a la comunicación entre nodos de forma directa, es decir la actualización de información no depende de un servidor central sino cada nodo funge como servidor central que se comunican entre todos teniendo la misma información lo que ayuda a no tener puntos de fallas.

Syncthing. - herramienta para la sincronización de datos o archivos entre dos dispositivos usando el tipo de comunicación “Peer to Peer”.

Red Tor. - Es una red de comunicaciones distribuida de baja latencia y superpuesta sobre internet, en la que el encaminamiento de los mensajes intercambiados entre los usuarios no revela su identidad, es decir, su dirección IP (anonimato a nivel de red).

Rootear móvil. - Proceso de instalación de software externo en tu móvil para poder entrar como administrador del sistema en los sistemas operativos tipo Linux (Android) el usuario de administrador se llamo “root” al poder rootear tu móvil tendrás acceso a cualquier proceso. Importante al realizar este proceso algunos fabricantes de móviles (Smartphone) comentan que la garantía se pierde por cualquier falla que tenga el móvil.

6. Arquitectura de procesos en Mini BlocklyChain

Mini BlocklyChain está formada por tres procesos que forman su arquitectura, la primera son los procesos de negocio, la segunda los procesos de comunicaciones y en tercero es el ambiente de desarrollo para módulos complementarios y/o la creación de BlocklyCode “contratos inteligentes”.

Los procesos de negocio son los bloques que forman una serie de rutinas para crear una transacción ya sea en red pública o privada, este tipo de proceso es la planeación del negocio, el cómo, cuándo, que, quienes, donde y más atributos serán ordenados, planeados y distribuidos para que se logre la funcionalidad principal de cada transacción enviada, recibida, almacenada y/o rechazada. El primer proceso está compuesto por los siguientes tipos de bloques divididos en cuatro categorías:

1. Bloques de entrada de datos
2. Bloques de proceso de datos
3. Bloques de seguridad.
4. Bloques de datos modulares (desarrolladores)
5. Bloques de comunicación.

Los bloques de entrada de datos, son los que reciben la transacción con mínimo cuatro parámetros de entrada (**dirección origen, dirección destino, activo, datos**) pudiendo tener más dependiendo de la variable “datos”, este puede ser un bloque desarrollado por terceros o con valor nulo (NULL).

Los bloques de procesos de datos desarrollan la logística, calculo, normalización de datos, decisiones lógicas y verificaciones de flujo cada transacción. Este tipo de bloques se usan para darle inteligencia al proceso de negocio y se basan principalmente como su nombre lo indica en procesar todo tipo de información, así como su conversión de diferentes tipos de datos.

Los bloques de seguridad son usados para validar la información y asegurar que no han sido alteradas las transacciones desde su origen al destino, siempre son procesadas con diversos algoritmos de seguridad usados en tecnologías de blockchain, entre las herramientas más usadas se encuentra (firma hash, firma digital, cifrado de datos AES, creación y validación de direcciones digitales, entre otros).

Los bloques de datos modulares, estos son los bloques que son desarrollados por terceros, recordemos que Mini BlocklyChain fue creado de forma modular para ser enriquecido por nuevos bloques según las necesidades de cada sector sea público o privado. Para saber más de cómo crear módulos revisar el Anexo de desarrolladores.

Como comentamos anteriormente la arquitectura de Mini BlocklyChain en su segundo componente son los procesos de comunicaciones, estos procesos son los encargados de los canales de comunicación vía TCP y Sockets de comunicación para dar flexibilidad de envío y recepción de transacciones ya sea por los diferentes medios que se usan actualmente los más usados son: Internet móvil, Wifi actualmente se trabaja para incluir el medio de comunicación Bluetooth.

Los bloques de comunicaciones básicamente están basados en el intercambio de datos con seguridad implementada en el canal de transferencia y esta se basa en una comunicación a través del protocolo SSH (Secure Shell) con las diferentes etapas que recorre una transacción enviada o recibida.

La parte de comunicaciones son fundamentales ya que estos procesos tienen la función de actualizar la información en todos los nodos vía TCP y la conexión llamada **“Peer to Peer”** los bloques que intervienen en la comunicación se basan en intercambio de información entre los nodos sin la intervención de servidor intermediarios por lo que cada nodo lo hace independiente pudiendo crear una red de nodos donde los puntos de falla son mínimos o casi nulo. Así mismo esta independencia de cada nodo los ayuda a tomar decisiones de forma individual o en conjunto según sean la necesidad del negocio.

La arquitectura “Peer to Peer” está formada por tres partes que forma la red pública o privada que se decida crear, en ambos casos el canal de comunicación está cifrado de nodo a nodo.

El primer componente para la comunicación entre dispositivos móviles (smartphones) o wifi es proporcionarles a los nodos o dispositivos una red en donde puedan ser encontrados en cualquier parte del mundo, la red de comunicación en donde está montado MiniBlocklyChain es la red “Tor”.

¿Qué es la red Tor? – (<https://www.torproject.org>)

“**Tor** (sigla de **T**e **O**nion **R**outer -en español (**E**l **R**úter **C**ebolla) es un proyecto cuyo objetivo principal es el desarrollo de una red de comunicaciones distribuida de baja latencia y superpuesta sobre internet, en la que el encaminamiento de los mensajes intercambiados entre los usuarios no revela su identidad, es decir, su dirección IP (anonimato a nivel de red) y que, además, mantiene la integridad y el secreto de la información que viaja por ella.”

El segundo componente y tarea no menos importante es conseguir que todos los nodos en MiniBlocklyChain tengan en cualquier momento los mismos datos o tener sincronizada sus bases de datos y archivos para realizar esta tarea entre los nodos se implementara “syncthing”

¿Qué es la red Syncing? – (<https://syncing.net>)

Syncing es una aplicación gratuita de código abierto de sincronización de archivos punto a punto disponible para Windows, Mac, Linux, Android, Solaris, Darwin y BSD. Puede sincronizar archivos entre dispositivos en una red local o entre dispositivos remotos a través de Internet.

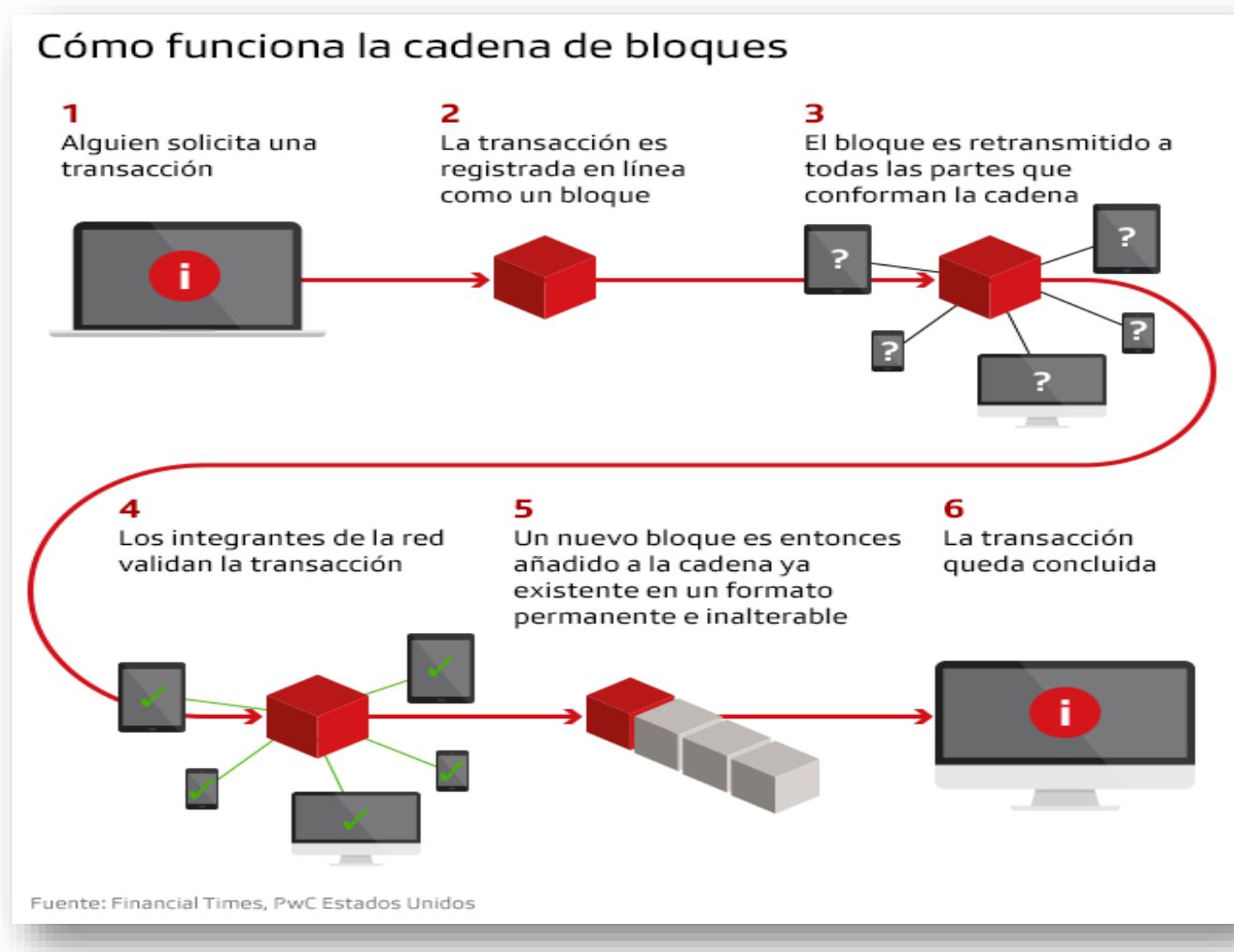
Teniendo como base los dos anteriores componentes de comunicaciones podemos empezar a desarrollar una red de confianza entre nodos y con el nivel de sincronización de datos que será la espina medular o “core” de los procesos del negocio a fin a cada necesidad del usuario o empresa.

El tercer componente en la red de comunicaciones es la base de datos Redis, esta realizará el aviso a todos los nodos en tiempo real de nuevas transacciones que son recibidas y nuevas transacciones que enviadas.

¿Qué es la red Redis DB? – (<https://redis.io>)

Redis es un motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes (clave/valor) pero que opcionalmente puede ser usada como una base de datos durable o persistente.

7. Diagrama de funcionalidad de la cadena de bloques (Mini BlocklyChain).



8. ¿Qué es Mini BlocklyCode?

Mini BlocklyCode son programas creados en el lenguaje Java y son guardados en un repositorio independiente al de los nodos, son normalmente nombrados como contratos inteligentes, estos programas están diseñados con condiciones lógicas para cuando dichas condiciones se cumplan se ejecuten de forma automática sin depender de alguna autorización o intervención humana externa.

“Un contrato inteligente (en inglés Smart contract) es un programa informático que facilita, asegura, hace cumplir y ejecuta acuerdos registrados dos o más partes (por ejemplo, personas u organizaciones). Como tales ellos les ayudarían en la negociación y definición de tales acuerdos que causarán que ciertas acciones sucedan como resultado de que se cumplan una serie de condiciones específicas.

Un contrato inteligente es un programa que vive en un sistema no controlado por ninguna de las partes, o sus agentes, y que ejecuta un contrato automático el cual funciona como una sentencia if-then (si-entonces) de cualquier otro programa de ordenador. Con la diferencia de que se realiza de una manera que interactúa con activos reales. Cuando se dispara una condición pre-programada, no sujeta a ningún tipo de valoración humana, el contrato inteligente ejecuta la cláusula contractual correspondiente.

Tienen como objetivo brindar una seguridad superior a la ley de contrato tradicional y reducir costos de transacción asociados a la contratación. La transferencia de valor digital mediante un sistema que no requiere confianza (ej. bitcoins) abre la puerta a nuevas aplicaciones que pueden hacer uso de los contratos inteligentes. “

Mini BlocklyCode está dirigido a desarrolladores con experiencia en programación java. En el Mini BlocklyChain se restringen algunos tipos de librerías por seguridad del mismo sistema, sin embargo, las librerías para crear transacciones para enviar o recibir algún valor contractual creado o acordado por dos partes o más puede ser creado.

Todo Mini BlocklyChain cumple con las siguientes premisas:

- ✓ Todo Mini BlocklyChain creado está basado en ejecutar únicamente mensajes y no ejecuciones en el sistema, sin embargo, estos mensajes pueden contener autorizaciones, aprobaciones de contratos físicos o acciones físicas.
- ✓ Todo Mini BlocklyChain creado tiene que pasar por el bloque de comunicaciones “auditor” este bloque se encarga de la supervisión de código malicioso o código prohibido para revisar sentencias u órdenes no autorizadas en el sistema.
- ✓ Todo Mini BlocklyChain es ejecutado únicamente en la JVM del nodo origen, los programas no se ejecutan en forma global para protección del sistema.

Para más detalles consultar el anexo “Mini BlocklyChain para desarrolladores”.

9. Instalación de red comunicaciones de Mini BlocklyChain

1. Instalación y configuración red Mini SQLSync.

La red Mini SQLSync es la encargada de recibir las transacciones de los nodos para que estos puedan procesar dichas transacciones, esta red está formada una red principal que es a través de “Peer to Peer” entre nodos y una red de respaldo llamada Mini Centinela RESTful.

Iniciaremos con la instalación de la red de respaldo Mini Centinela RESTful, este servicio es el que recibirá las transacciones de los nodos, este servicio está basado en almacenar las transacciones por un tiempo determinado para posteriormente ser convertida la información a través de un conector que inyectará una cola de todas las transacciones cifradas hacia un servicio Redis. Posteriormente el servicio de la base de datos Redis ejecutara en tiempo real la liberación de la cola cifrada de transacciones a todos los nodos que integren la red de Mini BlocklyChain.

Un servicio o diseño tipo RESTful es una forma sencilla y simple de consultar, modificar, borrar y/o insertar información a una base de datos a través de una URL o dirección en internet, en nuestro caso usaremos la base de datos SQLite por sus características de estar toda la información en un solo archivo. Para un uso de requerimientos con necesidades de mayor escalamiento podremos usar otro tipo de base de datos como MySQL, Oracle, DB2 u otra base de datos comercial, simplemente tendremos que cambiar el conector de Mini BlocklyChain de SQLite (versión libre) por el conector de Mini BlocklyChain DB otras (versión comercial).

NOTA IMPORTANTE: La configuración de Mini Centinela RESTful en ningún momento procesa ningún tipo de transacciones, únicamente es el servicio que formatea “da forma a la información” para que los nodos puedan realizar el procesamiento de las transacciones de forma adecuada y en un tiempo planeado y sincronizado para todos los nodos.

Instalación del servicio de RESTful está basado en una base de datos SQLite esta instalación la realizaremos en ambiente Windows para fines prácticos, sin embargo, este modelo se puede fácilmente replicar en un sistema operativo tipo Linux.

Para quien desee revisar el desempeño y soporte de consultar e inserciones a SQLite puede consultar estas pruebas de desempeño:

<https://www.sami-lehtinen.net/blog/sqlite3-performance-testing>

<https://sqlite.org/src4/doc/trunk/www/lsmperf.wiki>

Instalación de red de respaldo Mini Centinela RESTful. Este servicio la estaremos instalando y configurando en un equipo Windows 10, sin embargo, el proceso también se ha instalado en un ambiente de Linux Ubuntu 18.09 fácilmente.

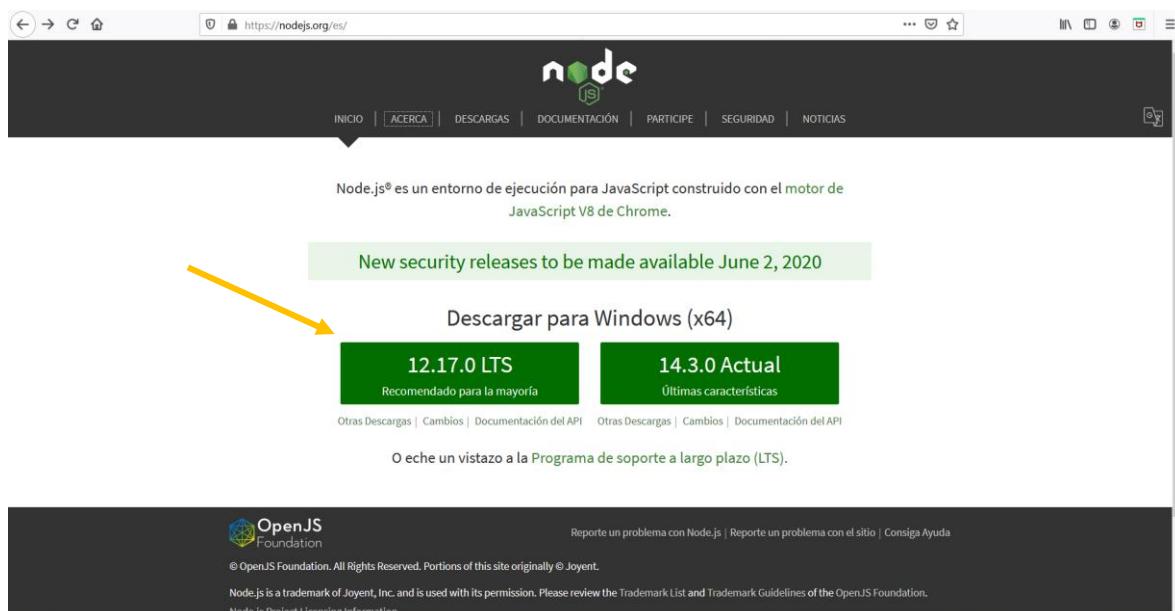
Requerimientos:

- ✓ Servidor de base de datos SQLite en modo RESTful (<https://github.com/olsonpm/sqlite-to-rest>).
- ✓ Conector Centinela SQLite-Redis
- ✓ Servidor de base de datos Redis en modo Master-Slave (<https://redis.io/topics/replication>).

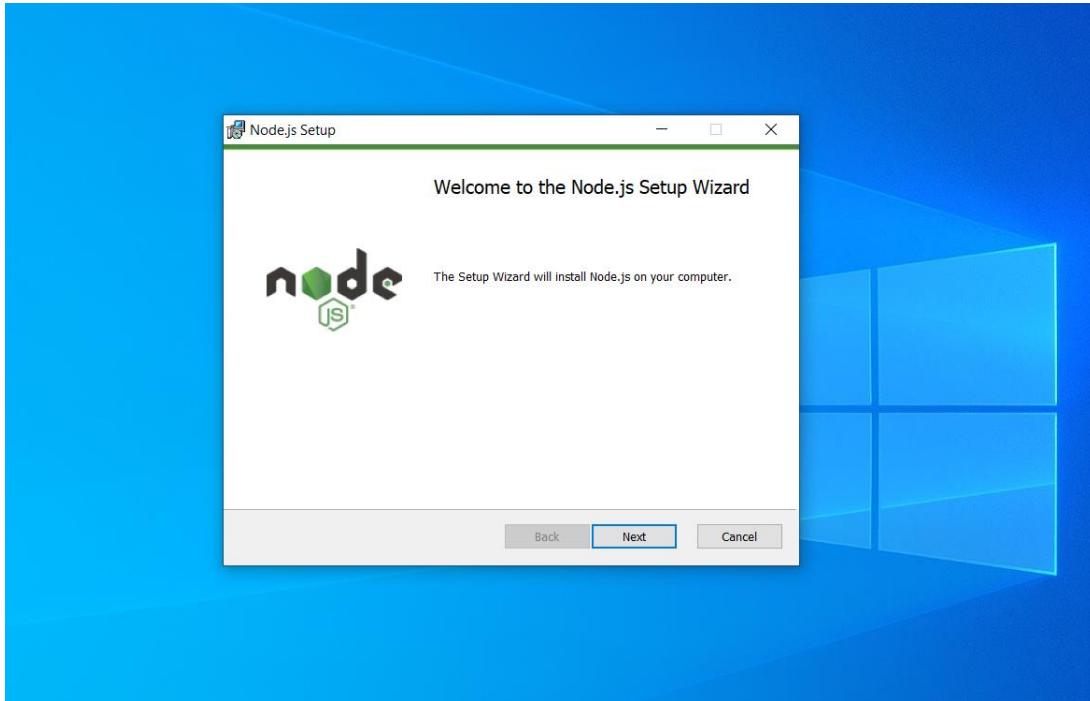
Instalación de Servidor de base de datos SQLite en modo RESTful.

Para la instalación necesitamos empezar contando con los siguientes paquetes (software), Nodejs, sqlite3 y Git Bash para Windows.

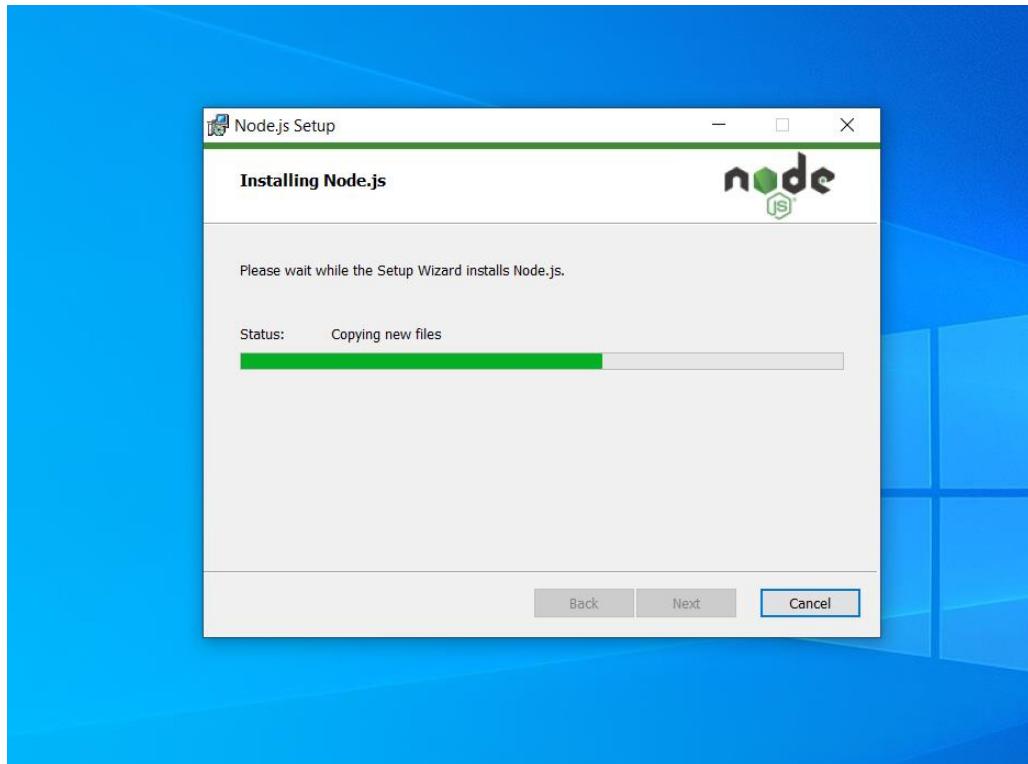
Para obtener Nodejs consultamos su sitio oficial <https://nodejs.org/es/> y escogemos la versión de “Recomendado para la mayoría”.

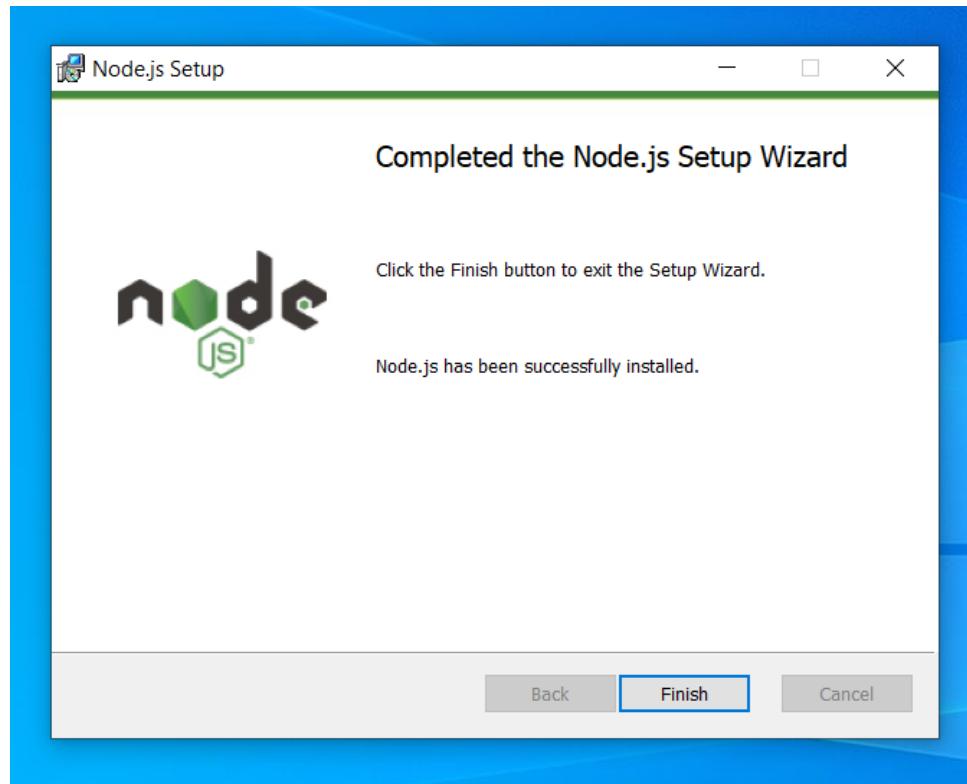


Después de bajar el archivo con extensión .msi damos doble click para instalarlo.



Realizamos la instalacion por default, simplemente damos click de “Next” sin escoger ningun opcion adicional que nos pregunte.





Ya que hemos finalizado la instalación de Nodejs proseguimos con la instalación de la base de datos SQLite.

Nos vamos a su sitio oficial, <https://www.sqlite.org/index.html> y hacemos click en la parte donde este “download”.

A screenshot of the SQLite.org homepage. The URL in the address bar is https://www.sqlite.org/index.html. A yellow arrow points to the "Download" button in the navigation menu. The page content includes sections like "What Is SQLite?", "Latest Release" (with a link to Version 3.32.1), and "Source Code".

What Is SQLite?

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured database engine in the world. SQLite is built into all mobile phones and most computers and is used by people every day. [More Information...](#)

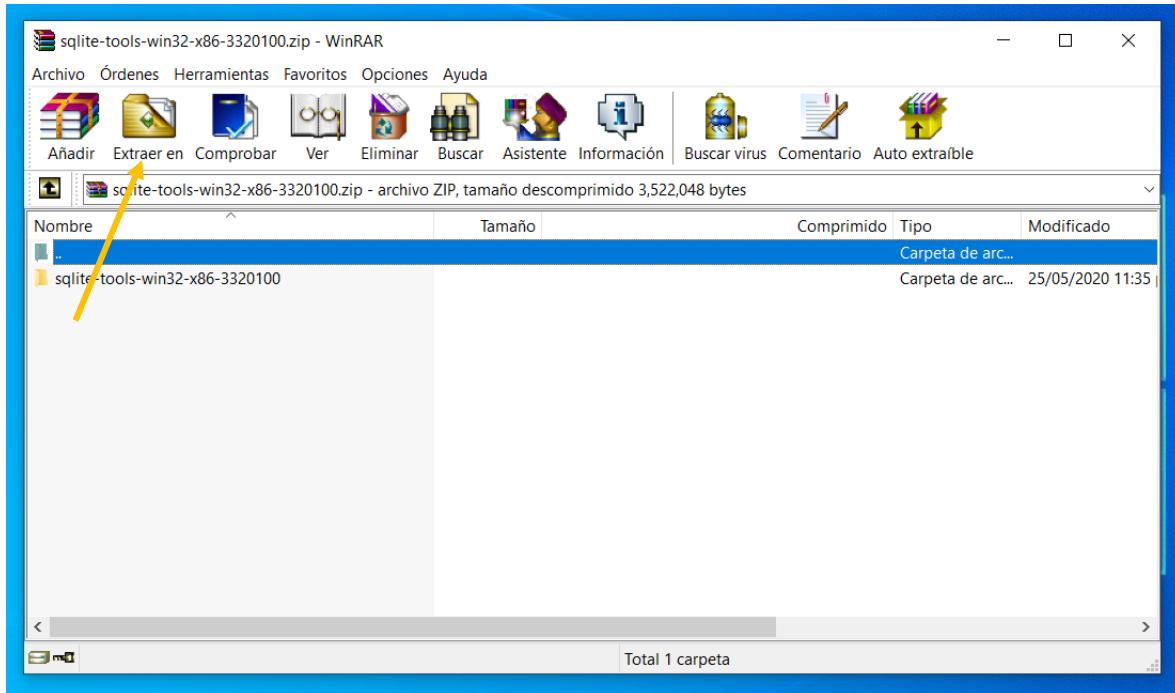
The SQLite [file format](#) is stable, cross-platform, and backwards compatible and the developers pledged to support it until 2050. SQLite database files are commonly used as containers to transfer rich content between systems. The SQLite file format is a simple, efficient, and reliable way to store data in a file. There are over 1 trillion (1e12) SQLite databases in active use [5].

SQLite [source code](#) is in the [public-domain](#) and is free to everyone to use for any purpose.

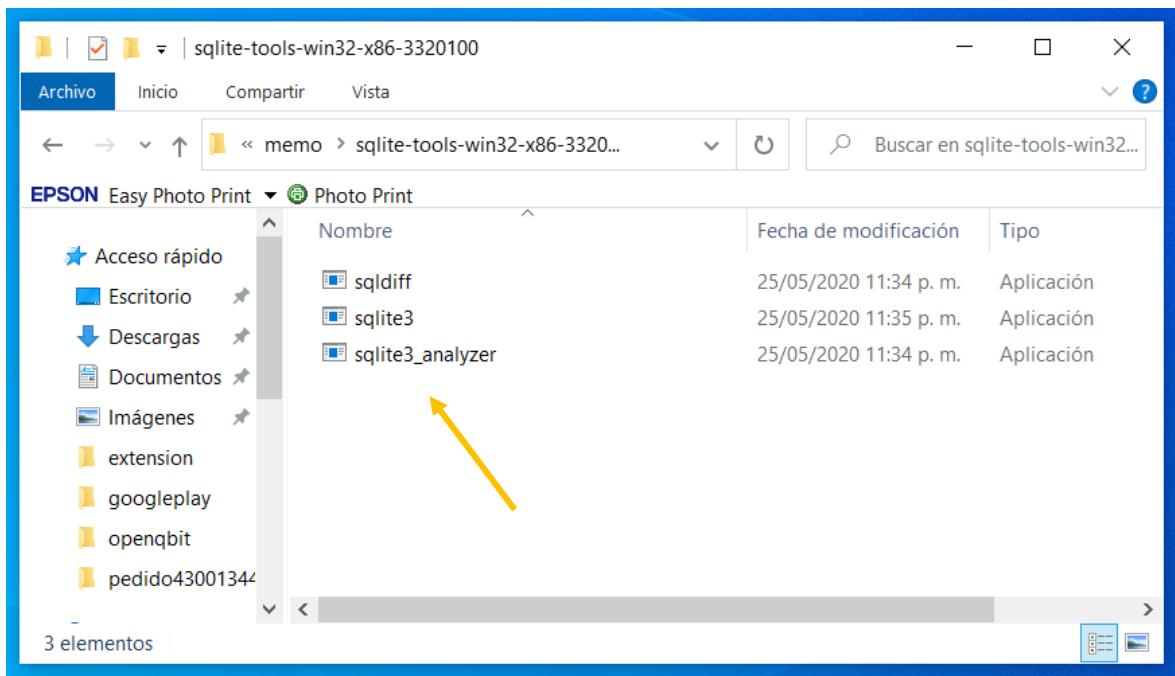
Latest Release

[Version 3.32.1](#) (2020-05-25). [Download](#) [Prior Releases](#)

Enseguida escogemos buscamos en la opción donde dice “**Precompiled Binaries for Windows**” escogemos y damos un click en la versión de software “**sqlite-tools-win32-x86-3320100.zip**” al terminar la descarga a nuestro equipo procedemos a descomprimirlo el archivo esto lo hacemos de una manera simple, abrimos el folder en donde se descargó el archivo y le damos doble click al archivo para descomprimirlo.

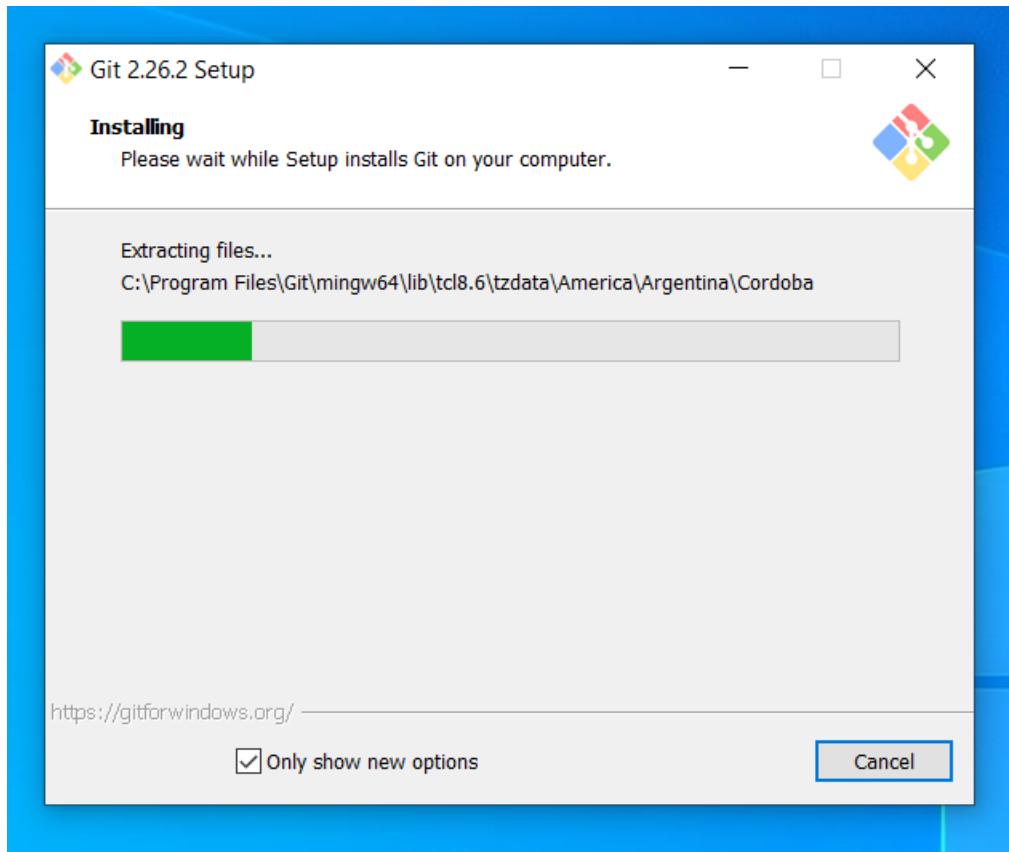


Terminado de descomprimirlo tendremos tres archivos, estos son nuestro ambiente para crear nuestra base de datos SQLite que posteriormente usaremos.



Comenzaremos con la instalación de Git Bash este programa es un emulador de terminal tipo Linux que nos ayudara a ejecutar el servicio de respaldo RESTful de una manera adecuada.

Vamos a descargarlo de su sitio oficial, <https://gitforwindows.org/> y procedemos a instalarlo con las opciones por default que nos indica.



Ahora que ya tenemos nodejs, sqlite y Git Bash en nuestro equipo Windows 10 instalados procedemos con la instalación del ambiente que soportara nuestra base de datos SQLite en modo RESTful.

En momento de descargar el software que dará la funcionalidad de RESTful a SQLite. Para esto debemos ir al siguiente sitio, <https://github.com/olsonpm/sqlite-to-rest> en este daremos click en el botón verde derecho “Clone or download” y escogemos la opción de “Download ZIP” este nos descargara el software en nuestro equipo.

No description or website provided.

automatic-api

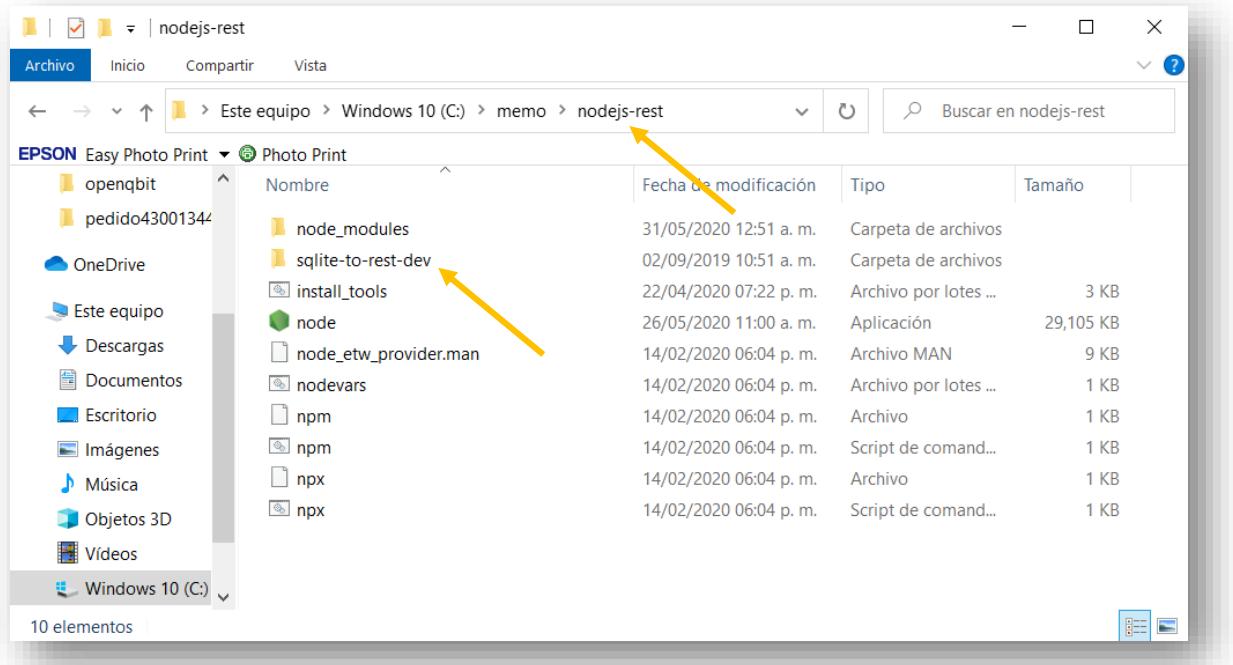
Branch: dev New pull request Find file Clone or download ▾

	olsonpm add prettier and eslint	...	Clone with HTTPS ⓘ
	bin	add prettier and eslint	Use Git or checkout with SVN using the web URL. https://github.com/olsonpm/sqlite-to-r
	cli/commands	add prettier and eslint	
	docs	add prettier and eslint	
	lib	add prettier and eslint	
	tests	add prettier and eslint	
	.gitignore	add prettier and eslint	

Open in Desktop Download ZIP 9 months ago 9 months ago 9 months ago

Después de descargarlo en nuestro equipo procedemos a descomprimirlo dentro del directorio o folder en donde instalamos el programa de **nodejs** y nos quedara un directorio con el nombre “**sqlite-to-rest-dev**”.

NOTA: Es importante que el directorio **sqlite-to-rest-dev** este dentro del directorio donde se instaló **nodejs**.



Teniendo todo instalado procedemos con la configuración de la base de datos SQLite que usaremos para almacenar las transacciones de los nodos en el ambiente de respaldo RESTful.

Diseño de tablas y estructura de datos. Comandos para ejecutar en línea en lista de comandos CMD:

```
sqlite3 op.sqlite3 "CREATE TABLE trans (id integer primary key, addro, addrd, value);"
```

```
sqlite3 op.sqlite3 "CREATE TABLE sign (id integer primary key, trans_id references trans (id), sign, hash);"
```

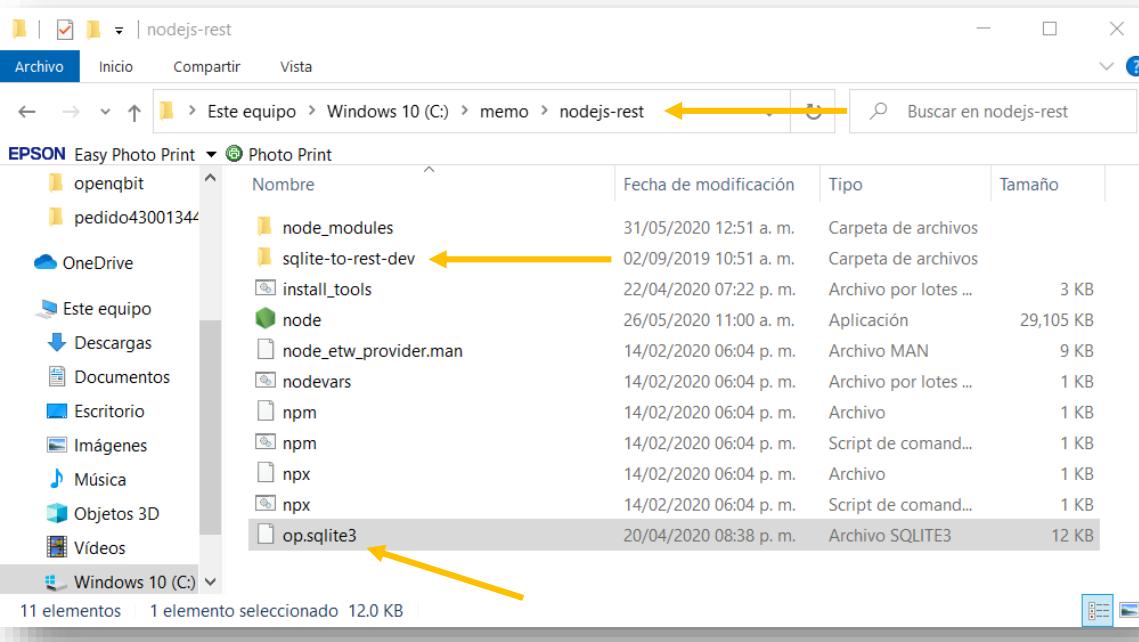
```
C:\memo\sqlite-tools-win32-x86-3320100>sqlite3 op.sqlite3 "CREATE TABLE trans (id integer primary key, addro, addrd, value);"
C:\memo\sqlite-tools-win32-x86-3320100>sqlite3 op.sqlite3 "CREATE TABLE sign (id integer primary key, trans_id references trans (id), sign, hash);"
C:\memo\sqlite-tools-win32-x86-3320100>dir
El volumen de la unidad C es Windows 10
El n mero de serie del volumen es: E019-5C05

Directorio de C:\memo\sqlite-tools-win32-x86-3320100

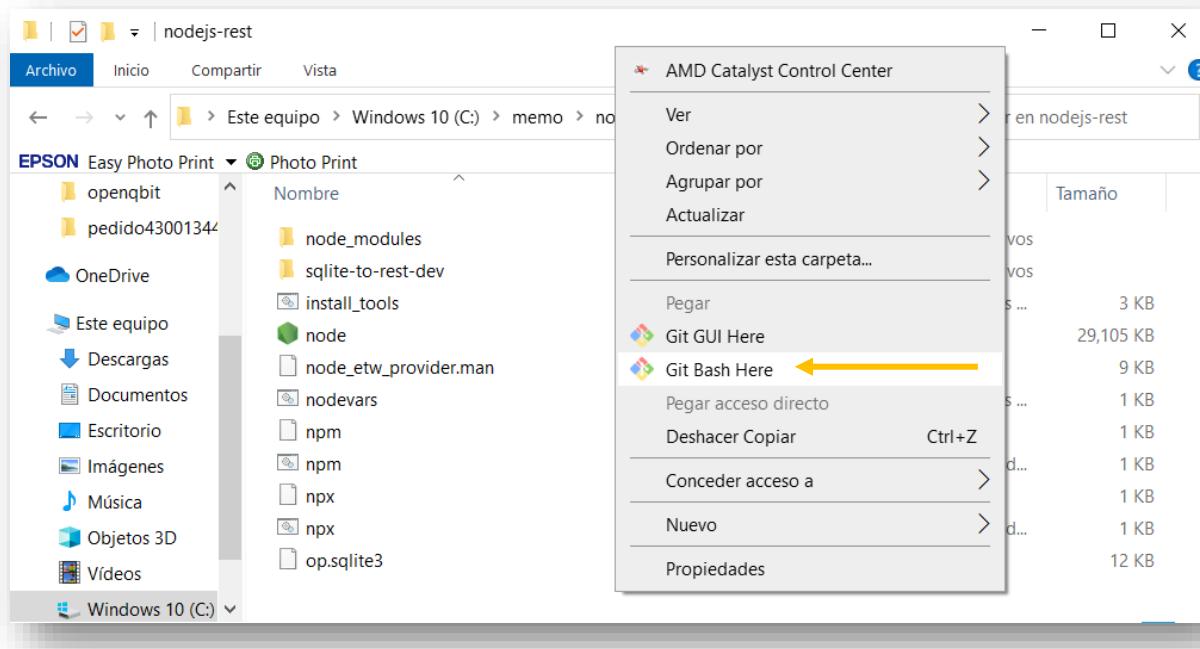
31/05/2020 02:29 a. m.    <DIR>        .
31/05/2020 02:29 a. m.    <DIR>        ..
31/05/2020 02:29 a. m.          12,288 op.sqlite3
25/05/2020 11:34 p. m.      516,608 sqldiff.exe
25/05/2020 11:35 p. m.      972,800 sqlite3.exe
25/05/2020 11:34 p. m.      2,032,640 sqlite3_analyzer.exe
              4 archivos       3,534,336 bytes
              2 dirs   137,272,078,336 bytes libres

C:\memo\sqlite-tools-win32-x86-3320100>
```

Después de crear la base de datos op.sqlite3 debemos hacer una copia en el directorio en donde se instaló el **nodejs**. En el directorio de **nodejs** debe estar también la copia de “**sqlite-to-rest-dev**”.

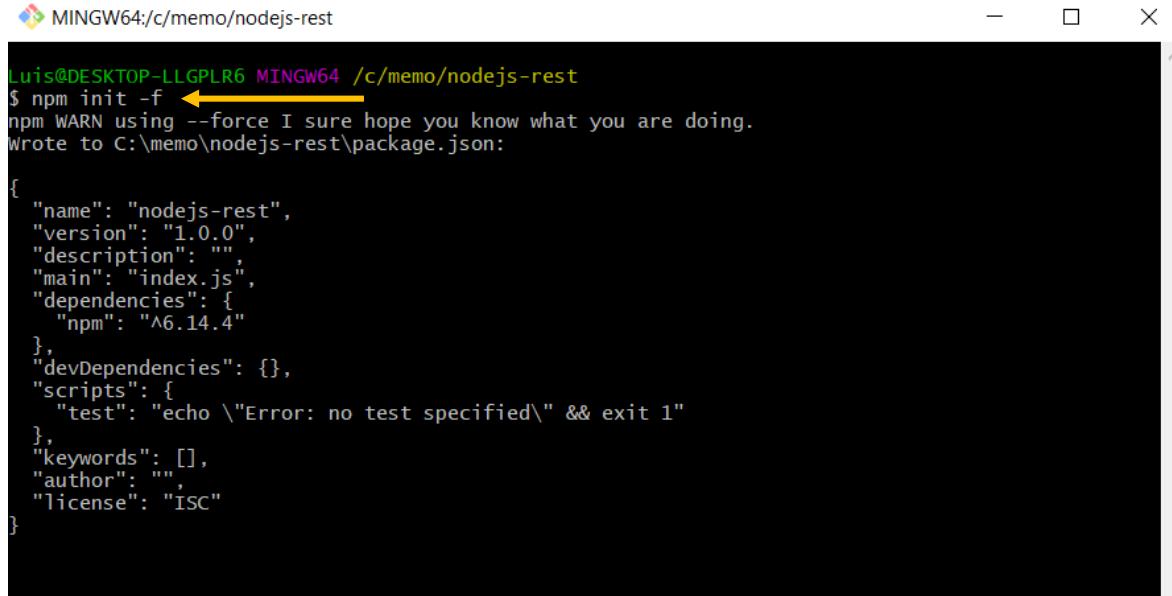


Nos situamos en el folder en donde está la instalación de **nodejs** y donde también debe estar el software “**sqlite-to-rest-dev**”. Señalamos el folder con el apuntador y damos click derecho para mostrar menú y escogemos donde dice “**Git Bash**” para abrir una terminal.



En la terminal nueva Git Bash abierta ejecutamos los siguientes comandos:

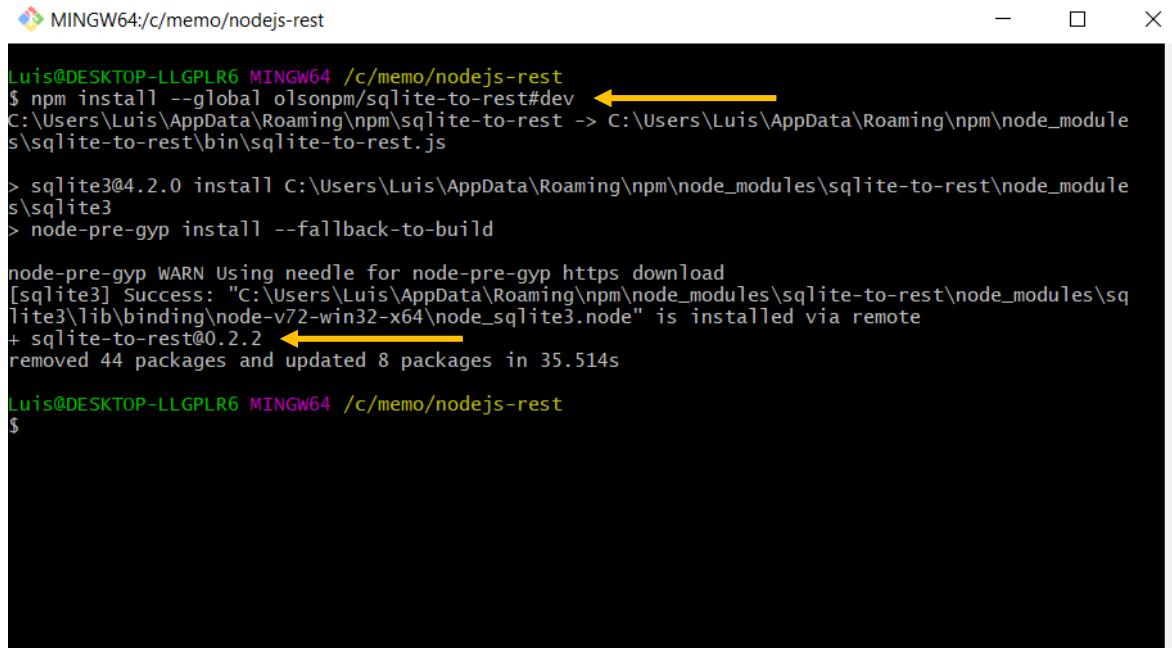
\$ npm init -f



```
Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$ npm init -f ←
npm WARN using --force I sure hope you know what you are doing.
Wrote to C:\memo\nodejs-rest\package.json:

{
  "name": "nodejs-rest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "dependencies": {
    "npm": "^6.14.4"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

\$ npm install --global olsonpm/sqlite-to-rest#dev



```
Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$ npm install --global olsonpm/sqlite-to-rest#dev ←
C:\Users\Luis\AppData\Roaming\npm\sqlite-to-rest -> C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\bin\sqlite-to-rest.js

> sqlite3@4.2.0 install C:\Users\Luis\AppData\Roaming\npm\node_modules\sqlite-to-rest\node_modules\sqlite3
+ sqlite3@0.2.2 ←
removed 44 packages and updated 8 packages in 35.514s

Luis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
$
```

Después de la ejecución del comando aparecerá la instalación del paquete “**sqlite-to-rest**”.

Generamos ambiente RESTful para SQLite con la base **op.sqlite3** que creamos anteriormente.

Ejecutamos el comando: **\$ sqlite-to-rest generate-skeleton --db-path ./op.sqlite3**



```
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
sqlite-to-rest generate-skeleton --db-path ./op.sqlite3
package.json found in working directory.
Installing dependencies
Writing the skeleton server to: skeleton.js
finished!
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
```

Iniciamos servicio RESTful SQLite en puerto por default 8085. Ejecutamos el siguiente comando: **\$ node skeleton.js**



```
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
node skeleton.js
listening on port: 8085
```

Probamos el servicio RESTful con agregar datos y consultar datos.



```
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
curl -s -H "Content-Type: application/json" -d '{"addr": "QWERTY1234", "addrd": "ASDFG4567", "value": "999"}' http://localhost:8085/transactions
"id":6,"addr": "QWERTY1234","addrd": "ASDFG4567","value": "999"}  
uis@DESKTOP-LLGPLR6 MINGW64 /c/memo/nodejs-rest
curl -s http://localhost:8085/transactions
"id":1,"addr": "CO","addrd": "Boulder","value": "Avery"}  
"id":2,"addr": "WI","addrd": "New Glarus","value": "New Glarus"}  
"id":3,"addr": "WI","addrd": "Madison","value": "One Barrel"}  
"id":4,"addr": "WI","addrd": "Madison","value": "One Barrel"}  
"id":5,"addr": "WI","addrd": "Madison","value": "One Barrel"}  
"id":6,"addr": "QWERTY1234","addrd": "ASDFG4567","value": "999"}
```

Para probar el servicio de RESTful de SQLite nos apoyamos con la ejecución de los siguientes comandos:

Para insertar datos en la table trans que se encuentra dentro de la base de datos op.sqlite3:

```
$ curl -s -H "Content-Type: application/json" -d '{"addr": "QWERTY1234", "addrd": "ASDFG4567", "value": "999"}' http://localhost:8085/trans
```

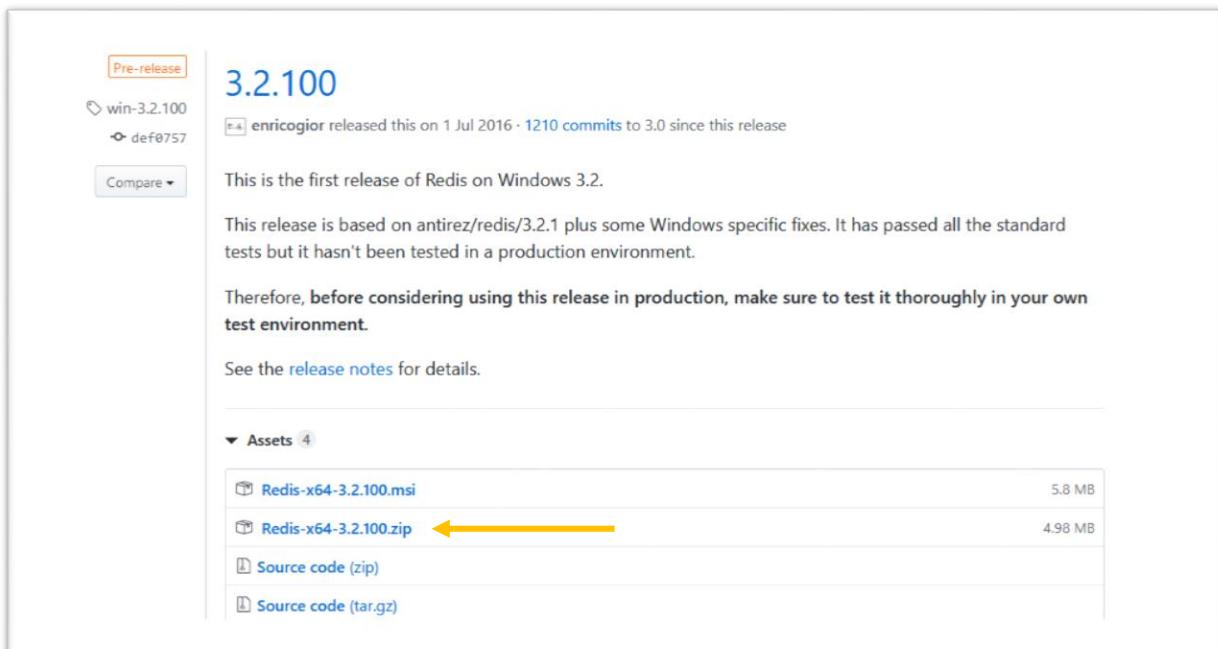
Para hacer una consulta de todos los datos de la tabla trans:

```
$ curl -s -H 'range: rows=0-2' http://localhost:8085/trans
```

Para revisar cómo usar a detalle todos los servicios de RESTful habilitados (consulta, insertar, actualizar y/o borrar datos) revisar el **Anexo “Restful SQLite comandos GET/POST”**

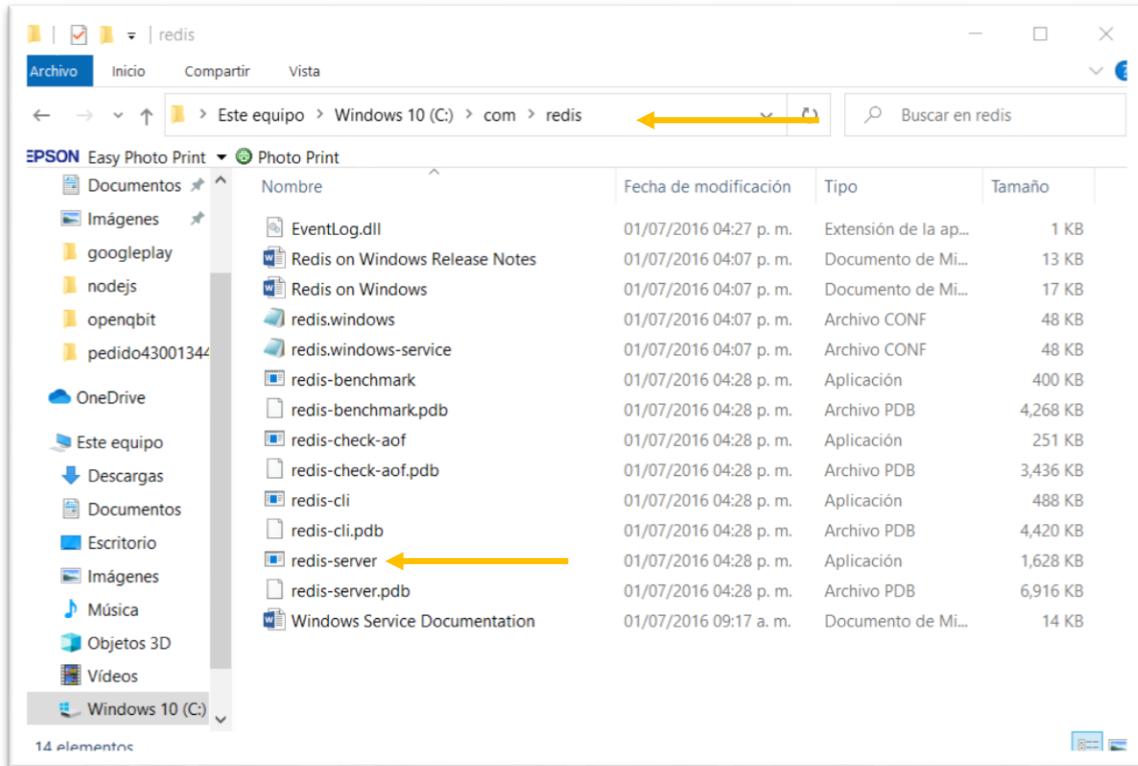
NOTA: En la instalación anterior todas las consultar las estamos realizando en el URL con dirección de “localhost” sin embargo al exponer el servidor con IP publica (internet) o IP privada (Wifi) funcionara sin ningún problema. Esto lo probaremos cuando estemos realizando las pruebas de comunicación entre el servicio RESTful de SQLite con los nodos de la red de comunicaciones que forman Mini BlocklyChain.

Instalación de base de datos Redis para servicio de red de respaldo, para descargar el software de redis para Windows tenemos que ir al sitio, <https://github.com/microsoftarchive/redis/releases/tag/win-3.2.100> y escogemos el paquete ZIP.



Para poder ubicar la instalación crearemos un directorio llamado “redis” dentro de Windows y descargamos el archivo con extensión ZIP, lo descomprimimos en el directorio creado con anterioridad, ya tenemos la instalación finalizada de redis.

Probamos la instalación ejecutando el servidor dando doble click en el comando “redis-server”.



Después de ejecutar el comando “redis-server” veremos ejecutándose el servidor en una terminal de comando Windows CMD en el puerto por default 6379:

```
C:\com\redis>redis-server.exe
[10192] 31 May 13:50:45.340 # Warning: no config file specified, using the default config. In order to specify a config
file use C:\com\redis\redis-server.exe /path/to/redis.conf

Redis 3.2.100 (00000000/0) 64 bit
Running in standalone mode
Port: 6379 ←
PID: 10192

http://redis.io

[10192] 31 May 13:50:45.355 # Server started, Redis version 3.2.100
[10192] 31 May 13:50:45.355 * The server is now ready to accept connections on port 6379
```

En esta prueba tenemos una versión de Redis 3.2.100 para Windows 10, en el caso de sistema operativo Linux se tiene la versión 6.0.4 (liberada mayo 2020) sin embargo para nuestra configuración de Mini BlocklyChain la versión de Windows tiene las características suficientes de funcionalidad que necesitamos.

Para detener la ejecución hacemos la combinación de teclas Ctrl + C. Procedemos a realizar la configuración de la base de datos Redis 3.2.100 para Windows 10 con una configuración entre Redis y los nodos de la red de comunicaciones Mini SQLSync, tipo **Master(Maestro)-Slave(Esclavo)** se distribuye de la siguiente forma:

El **Master (Maestro)** es el servidor Redis para Windows 10 que estamos configurando y este replicará los datos en tiempo real y sincronizados con la misma información a todos los nodos (teléfonos móviles) estos nodos tendrán instalado un servidor Redis en modo **Slave (Esclavo)**.

En este momento empezamos a ver cómo funciona la red de respaldo que estamos instalando y configurando. Esta configuración nos servirá para comunicarnos con todos los nodos en tiempo real, nos ayudará a comunicar a todos los nodos de la red cuando haya una nueva cola de transacciones para ser procesada por los nodos, en una igualdad en la transferencia de información a todos los nodos para que cualquier nodo tenga la misma probabilidad de poder procesar la información “cola de transacciones”.

Iniciamos la configuración del conector **Centinela SQLite-Redis**.

Este conector es un programa desarrollado en lenguaje Java y como su nombre lo indica conecta las bases de datos SQLite y Redis (**Maestro**).

La función del conector es transmitir la información (transacciones) de SQLite a Redis (**Maestro**) y este envía la “cola de transacciones” a los nodos (**Esclavos**).

Para más detalles de código java del conector **Centinela SQLite-Redis** revisar el Anexo “Conector Código Java SQLite-Redis”.

Configuración del archivo **redis.conf** de la base de datos Redis (**Maestro**) para Windows 10.

Agregar los siguientes cambios o directivas en el archivo, guardar cambios e iniciar servidor Redis

Comience por encontrar la configuración **tcp-keepalive** y establecerla en 60 segundos como sugieren los comentarios. Esto ayudará a Redis a detectar problemas de red o servicio:

tcp-keepalive 60

Encuentre la directiva **requirepass** y configúrela con una frase de contraseña fuerte. Si bien su tráfico de Redis debe estar protegido de terceros, esto proporciona autenticación a Redis. Dado que Redis es rápido y no califica los intentos de contraseña límite, elija una frase de contraseña fuerte y compleja para protegerse contra los intentos de fuerza bruta:

requirepass escribe_tu_redis_master_password

ejemplo:

requirepass FPqwedsLMdf76ass7asddfd2g45vBN8ty99

Finalmente, hay algunas configuraciones opcionales que quizás desee ajustar dependiendo de su escenario de uso.

Si no desea que Redis pone automáticamente las claves más antiguas y menos utilizadas a medida que se llena, puede desactivar el desalojo automático de claves:

maxmemory-policy noevasion

Para obtener garantías de durabilidad mejoradas, puede activar la persistencia de archivos de solo agregar. Esto ayudará a minimizar la pérdida de datos en caso de una falla del sistema a expensas de archivos más grandes y un rendimiento ligeramente más lento:

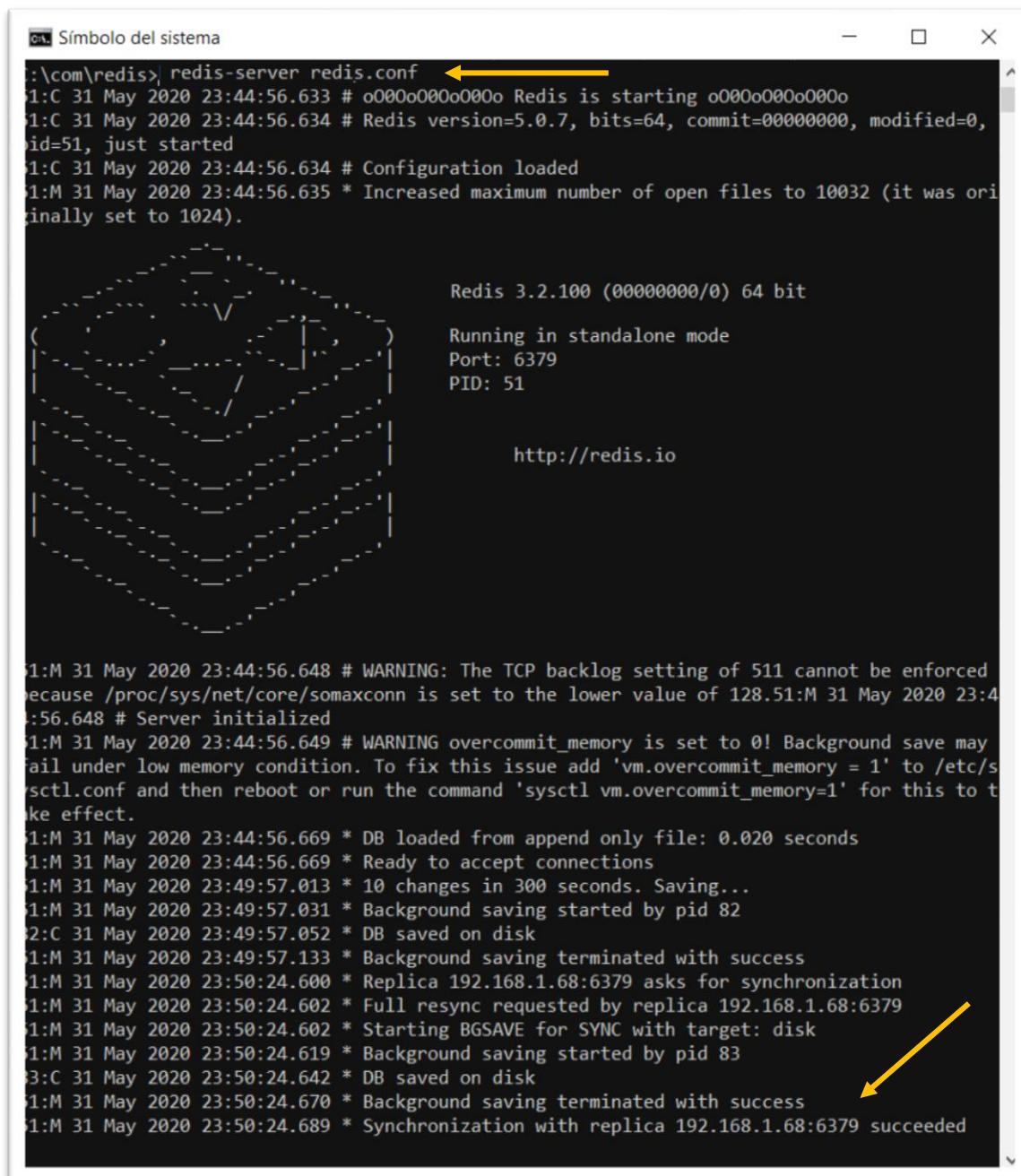
appendonly yes

appendfilename "redis-staging-ao.aof"

Procedemos a salvar los cambios y reiniciar el servicio de Redis para Windows 10, paramos con las teclas Ctrl + C y volvemos a ejecutar el comando en línea de comando Windows CMD:

C: \directorio_redis > redis-server redis.conf

En nuestro ejemplo podemos ver que tenemos conectado un nodo (**Esclavo**).



```

C:\com\redis> redis-server redis.conf
1:C 31 May 2020 23:44:56.633 # o000o000o000 Redis is starting o000o000o000
1:C 31 May 2020 23:44:56.634 # Redis version=5.0.7, bits=64, commit=00000000, modified=0,
id=51, just started
1:C 31 May 2020 23:44:56.634 # Configuration loaded
1:M 31 May 2020 23:44:56.635 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 3.2.100 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 51

http://redis.io

1:M 31 May 2020 23:44:56.648 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.51:M 31 May 2020 23:44:56.648 # Server initialized
1:M 31 May 2020 23:44:56.649 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
1:M 31 May 2020 23:44:56.669 * DB loaded from append only file: 0.020 seconds
1:M 31 May 2020 23:44:56.669 * Ready to accept connections
1:M 31 May 2020 23:49:57.013 * 10 changes in 300 seconds. Saving...
1:M 31 May 2020 23:49:57.031 * Background saving started by pid 82
2:C 31 May 2020 23:49:57.052 * DB saved on disk
1:M 31 May 2020 23:49:57.133 * Background saving terminated with success
1:M 31 May 2020 23:50:24.600 * Replica 192.168.1.68:6379 asks for synchronization
1:M 31 May 2020 23:50:24.602 * Full resync requested by replica 192.168.1.68:6379
1:M 31 May 2020 23:50:24.602 * Starting BGSAVE for SYNC with target: disk
1:M 31 May 2020 23:50:24.619 * Background saving started by pid 83
3:C 31 May 2020 23:50:24.642 * DB saved on disk
1:M 31 May 2020 23:50:24.670 * Background saving terminated with success
1:M 31 May 2020 23:50:24.689 * Synchronization with replica 192.168.1.68:6379 succeeded

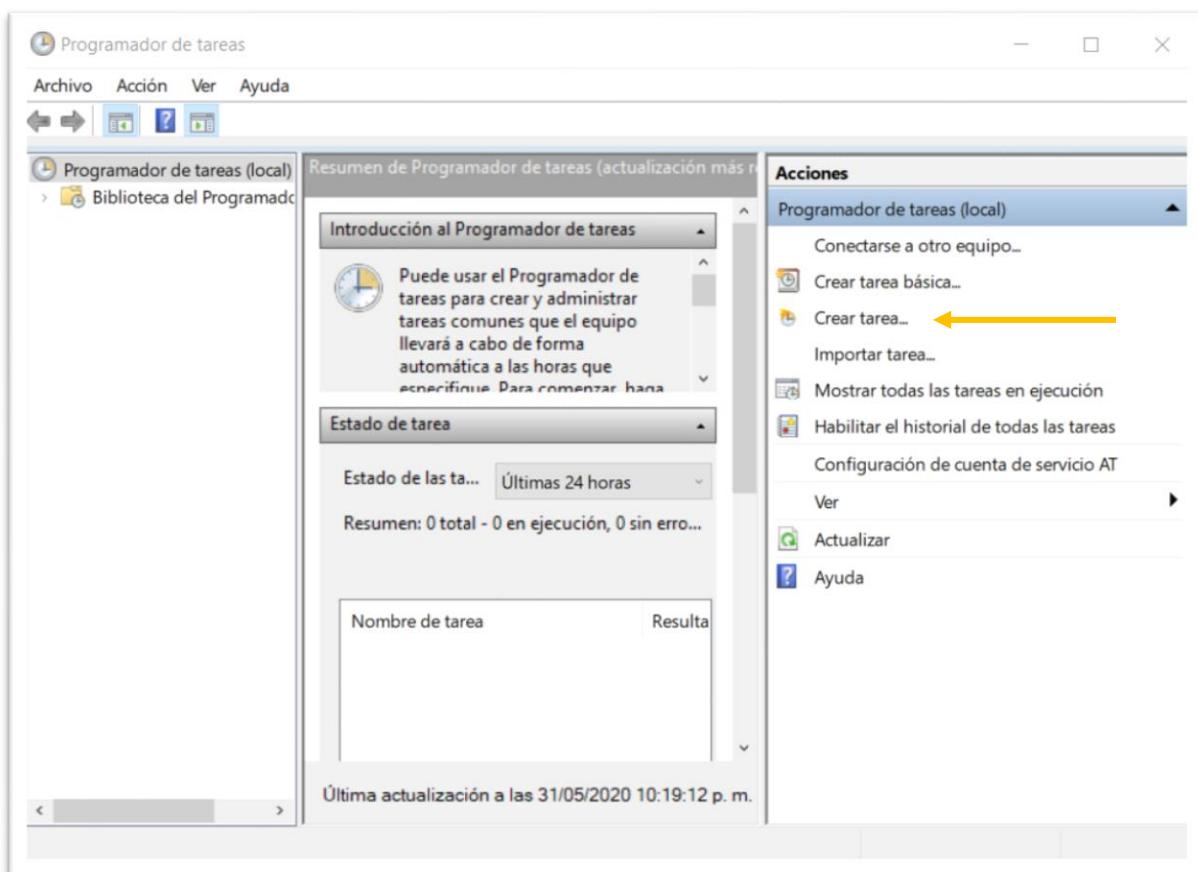
```

Podemos ver que tenemos sincronizado un nodo con la IP 192.168.1.68 en el puerto por default 6379.

Ahora necesitamos calendarizar en el sistema operativo Window 10 la tarea de ejecutar en forma automática el conector **Centinela SQLite-Redis**. Esto lo realizamos con la herramienta de Windows 10 está la ejecutamos desde la parte inferior izquierda escribiendo “Programador de tareas”.



Crearemos una nueva tarea “Crear tarea” en donde incluiremos la ejecución del conector.

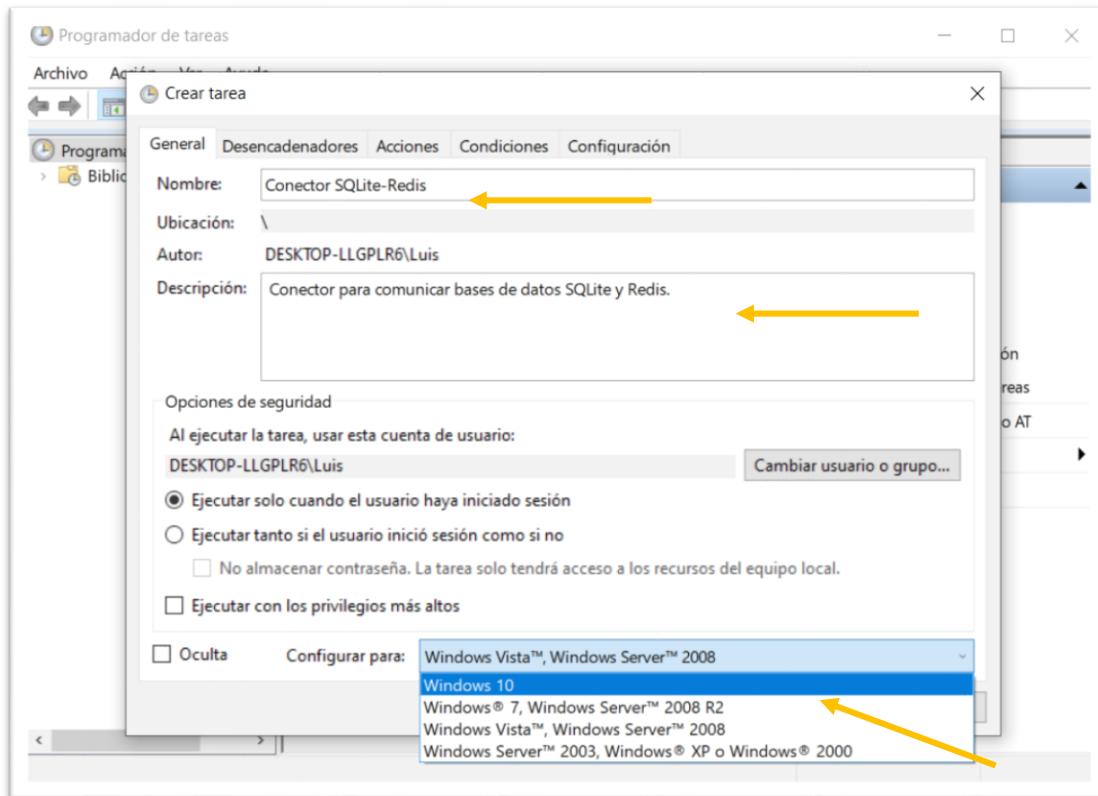


Al dar click en “Crear tarea” nos abrirá una ventana adicional en donde tenemos que dar los siguientes parámetros en pestaña “General”:

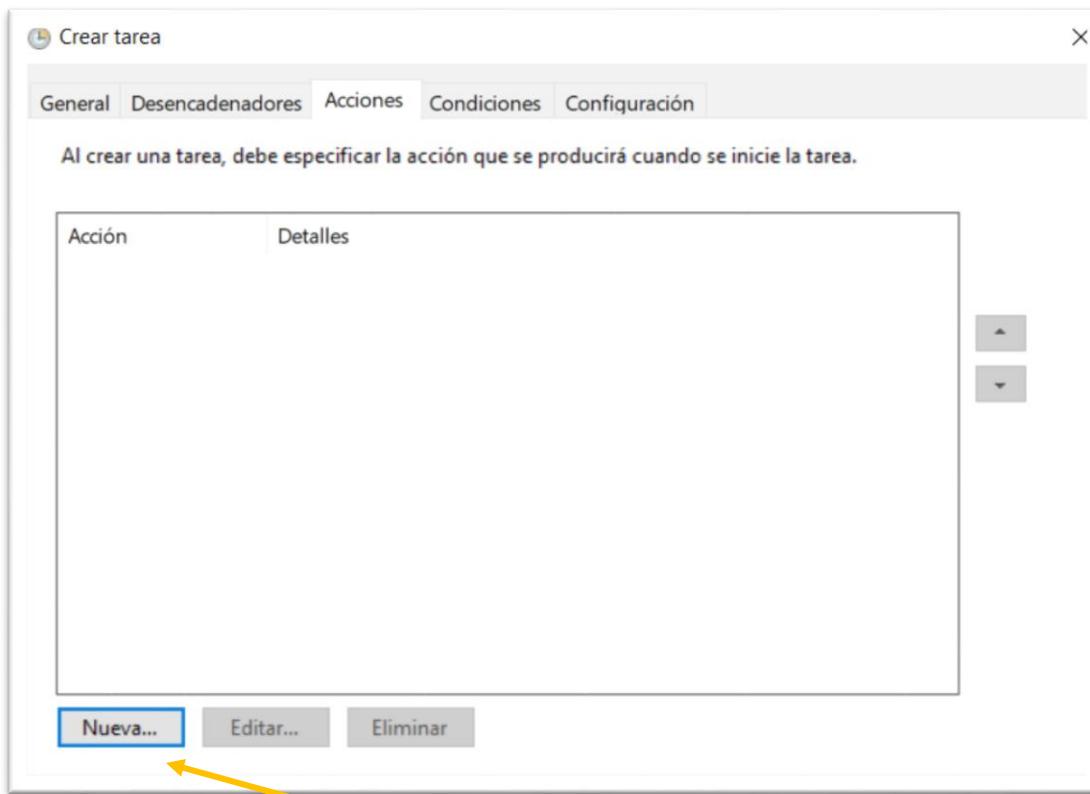
Nombre: nombre_de_tarea

Descripción: opcional

Configurar para: seleccionar_sistema_operativo_version_windows



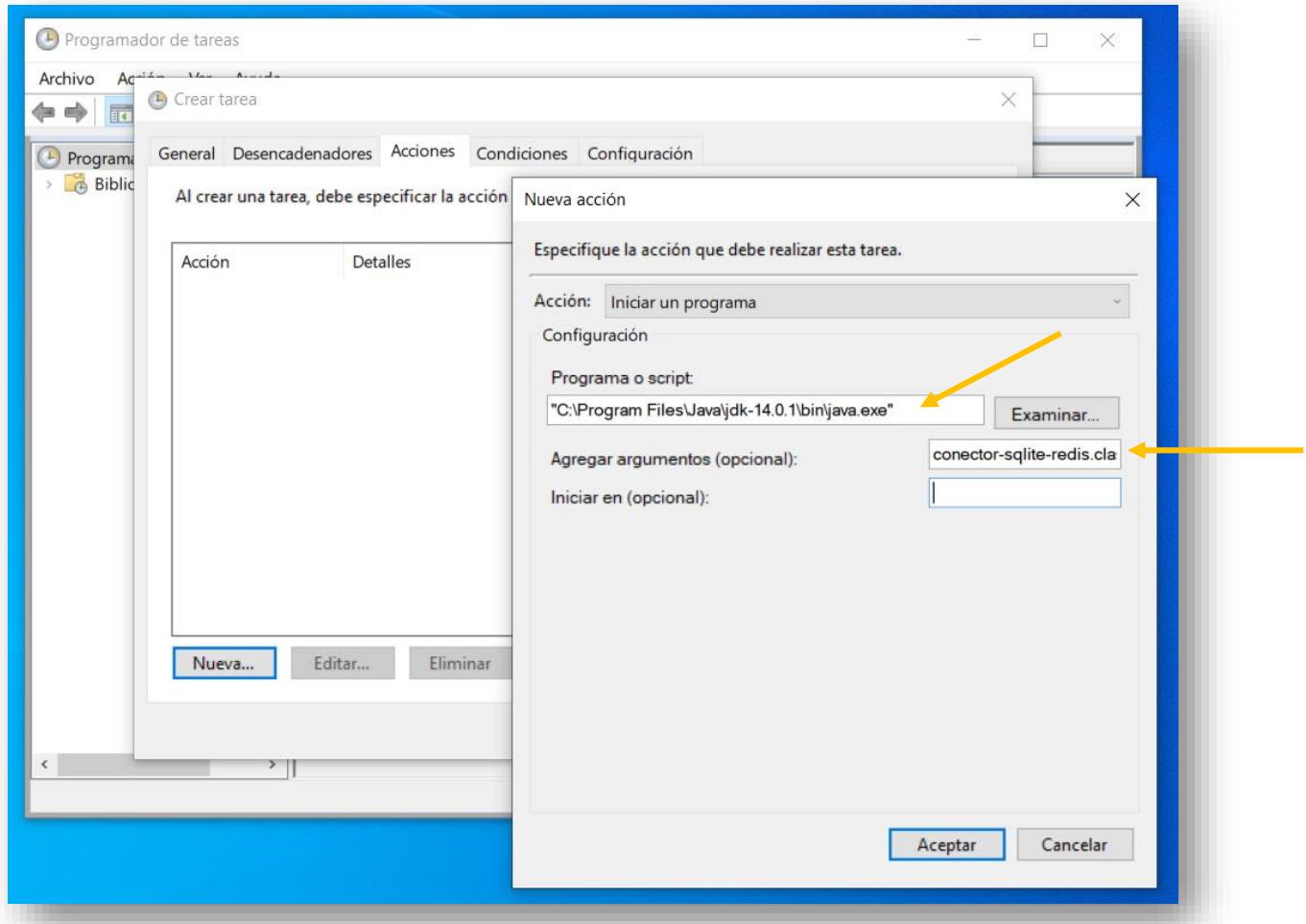
Cambiamos dando click en la pestaña “Acciones” y damos click en botón “Nueva”:



Damos los siguientes parámetros:

Programa o script: ruta_de_conector

Agregar argumentos: nombre del conector



Los anteriores parámetros pueden variar según ubicación del conector. La idea principal es la ejecución del programa **conector-sqlite-redis-v1.class** como normalmente se realiza en línea de comando:

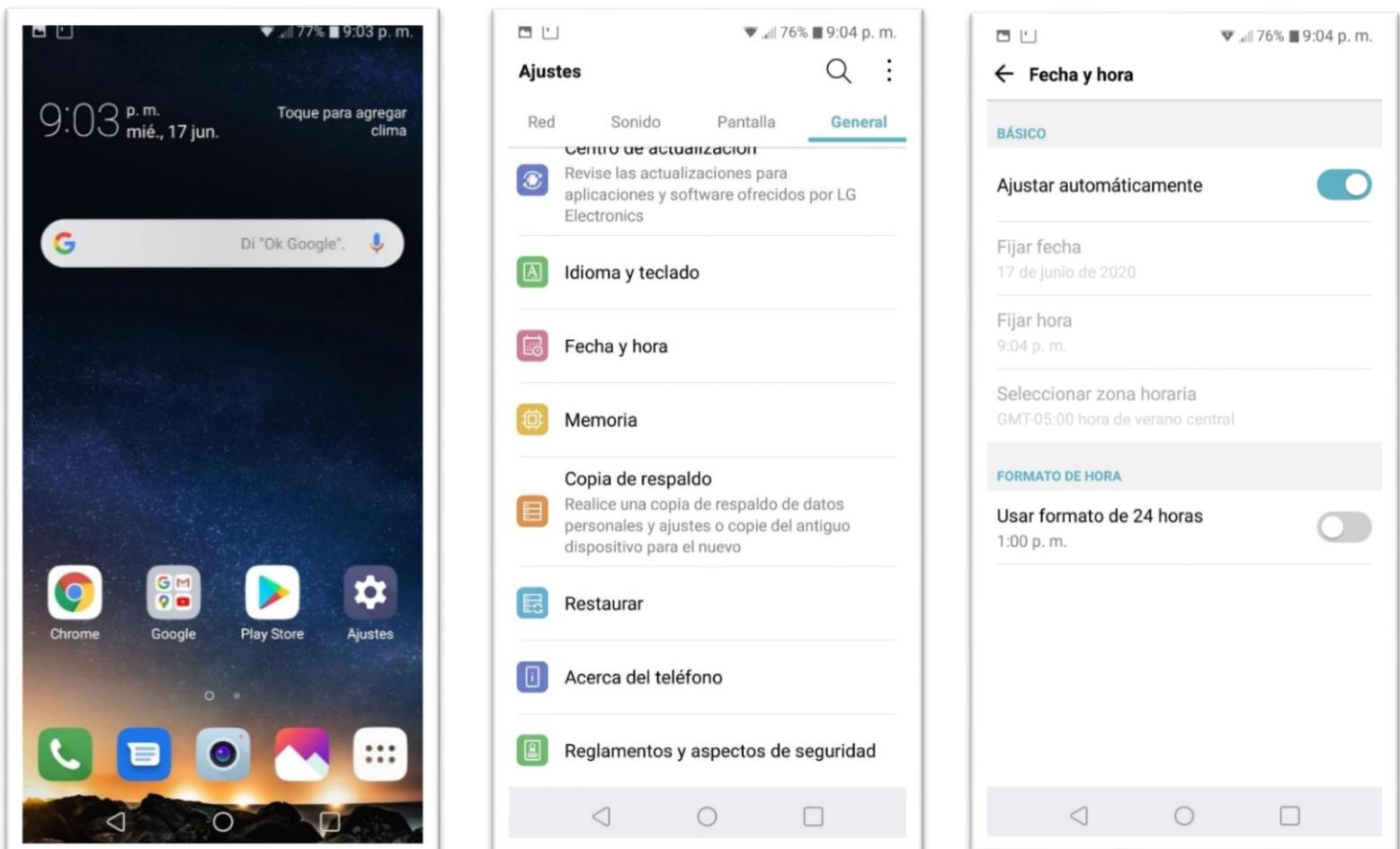
```
C: \directorio_de_jdk>java conector-sqlite-redis-v1
```

Para finalizar debemos escoger la pestaña “Desencadenadores”, en esta daremos los parámetros de cuando (día, hora, minutos) queremos que se ejecute la tarea, estos parámetros están basados dependiendo de las reglas de negocio que tendrá el sistema Mini BlocklyChain que sea creado.

10. Sincronización en nodos del sistema (Teléfono móvil) minutos y segundos.

Es muy importante que todos los nodos estén sincronizados principalmente en la parte de minutos y segundos. Ya que cuando se realice él envío o publicación en un tiempo determinado de la cola de transacciones todos los nodos deberán estar sincronizados ya que esto dependerá que el administrador de tareas que se establecerá en el sistema local con la herramienta CRON se ejecute al mismo tiempo en todos los nodos, esto hará que todos los nodos tengan la misma probabilidad para poder ganar el derecho a ser el elegido para poder procesar la cola de transacciones y poder generar el nuevo bloque para agregarlo a la cadena de bloques del sistema. Para sincronizar los nodos tenemos dos opciones.

La primera opción de la sincronización del nodo, la podremos hacer de forma sencilla. En nuestro dispositivo es a través de la opción interna que incluye el sistema de Android, tendremos que ir a la parte de **Ajustes > Fecha y hora > Ajustar automáticamente**



Con la configuración anterior nos servirá para tener sincronizados en minutos y segundos todos los nodos del sistema no importa en qué país del mundo estén ya la sincronización está basada en cada zona geográfica posiblemente lo que cambie es la hora pero en función de minutos y segundo deberán sincronizados esto nos bastara para que cuando se ejecute el proceso de tareas de forma programada con la herramienta cron en todos los nodos se ejecute cada determinado tiempo en minutos, es decir podemos crear una tarea en crontab para ejecutarla cada 10 minutos o cada 30 minutos dependiendo cada diseño de sistema.

Lo anterior servirá cuando usemos la red de comunicación “Peer to Peer”, en caso de usar la red de respaldo este proceso no aplicara ya que la distribución de la cola de transacción se realiza en un modelo de cliente-servidor y el servidor es el que controlar la herramienta cron.

Referencia: <https://appinventor.mit.edu/explore/blogs/karen/2016/08.html>

La segunda opción es apoyarnos de un API externo en donde realizaremos una ejecución del comando Curl a través de la extensión (**ConectorSSHClient**).

El sitio donde estaremos usando los servicios externos de NTP (Network Time Protocol) es:

<http://worldtimeapi.org/>

Ahora veremos una forma de obtener la hora de los servidores NTP que se encuentra a nivel mundial y que nos servirán para que todos los nodos tengan una consulta en un determinado tiempo de la misma fecha y hora.

Ejemplo de consulta con extensión (**ConectorSSHClient**).

\$ curl “http://worldtimeapi.org/api/timezone/America/Mexico_City”

Realizamos la conexión a la terminal Termux:



Ejecutamos el comando Curl:



Debemos tener en cuenta que el resultado del comando Curl estará en formato JSON algo semejante al siguiente resultado:

```

abbreviation: "CDT"
client_ip: "200.77.16.151"
datetime: "2020-06-18T14:16:57.750466-05:00"
day_of_week: 4
day_of_year: 170
dst: true
dst_from: "2020-04-05T08:00:00+00:00"
dst_offset: 3600
dst_until: "2020-10-25T07:00:00+00:00"
raw_offset: -21600
timezone: "America/Mexico_City"
unixtime: 1592507817
utc_datetime: "2020-06-18T19:16:57.750466+00:00"
utc_offset: "-05:00"
week_number: 25

```

También el resultado podría ser en el formato JSON sin los datos formateados en o JSON en forma lineal como se muestra a continuación:

```
{"abbreviation": "CDT", "client_ip": "200.77.16.151", "datetime": "2020-06-18T14:19:07.216800-05:00", "day_of_week": 4, "day_of_year": 170, "dst": true, "dst_from": "2020-04-05T08:00:00+00:00", "dst_offset": 3600, "dst_until": "2020-10-25T07:00:00+00:00", "raw_offset": -21600, "timezone": "America/Mexico_City", "unixtime": 1592507947, "utc_datetime": "2020-06-18T19:19:07.216800+00:00", "utc_offset": "-05:00", "week_number": 25}
```

Cualquiera de las dos formas anteriores tendremos que filtrar la información mediante una extensión JSON ya existente como “JSONTOOLS” o usar filtros en App Inventor en el procesamiento de Texto para solamente obtener la hora, día o segundos según sea la necesidad de cada sistema. Despues de procesar el resultado se podrá realizar una comparación lógica y en base a esa comparación podremos ejecutar una tarea ya programada con el servicio “cron” que más adelante veremos su configuración en cada nodo.

Referencia de extensión JSONTOOLS:

<https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html>

Ahora ya hemos revisado dos opciones para sincronizar el tiempo de los nodos. Continuaremos la configuración del servicio “cron” en cada nodo.

Configuración de calendarización de tareas automatizadas para su ejecución con el servicio CRON en sistemas Android (nodos del sistema).

Primero necesitamos entender cómo funcionan el administrador de tareas programadas de forma automática.

El servicio cron normalmente todos los sistemas tienen este preinstalado y en donde se calendarizan todas las tareas a ejecutar de forma automática están en un archivo llamado crontab.

Editamos archivo crontab con **\$ crontab -e**, usaremos el editor **vi**, dentro usamos el formato:

```
# m h dom mon dow user command
```

donde:

- **m** corresponde al minuto en que se va a ejecutar el script, el valor va de 0 a 59
- **h** la hora exacta, se maneja el formato de 24 horas, los valores van de 0 a 23, siendo 0 las 12:00 de la medianoche.
- **dom** hace referencia al día del mes, por ejemplo, se puede especificar 15 si se quiere ejecutar cada día 15
- **dow** significa el día de la semana, puede ser numérico (0 a 7, donde 0 y 7 son domingo) o las 3 primeras letras del día en inglés: mon, tue, wed, thu, fri, sat, sun.
- **user** define el usuario que va a ejecutar el comando, puede ser root, u otro usuario diferente siempre y cuando tenga permisos de ejecución del script.
- **command** refiere al comando o a la ruta absoluta del script a ejecutar, ejemplo: /home/usuario/scripts/actualizar.sh, si acaso llama a un script este debe ser ejecutable

Para que quedara claro unos cuantos ejemplos de tareas de cron explicados:

```
15 10 * * * usuario /home/usuario/scripts/actualizar.sh  
Ejecutará el script actualizar.sh a las 10:15 a.m. todos los días
```

```
15 22 * * * usuario /home/usuario/scripts/actualizar.sh  
Ejecutará el script actualizar.sh a las 10:15 p.m. todos los días
```

```
00 10 * * 0 root apt-get -y update Usuario root  
Ejecutará una actualización todos los domingos a las 10:00 a.m
```

```
45 10 * * sun root apt-get -y update  
Usuario root ejecutará una actualización todos los domingos (sun) a las 10:45 a.m
```

Guardamos cambios en el editor y con esto finalizamos la configuración del servicio cron. En caso de no tener instalado el cron en el sistema se podrá realizar con el siguiente comando:

\$ apt install cron

2. Instalación y configuración red Nodos – Teléfonos móviles.

Empecemos con la red de comunicaciones para nodos que utilizara Mini BlocklyChain.

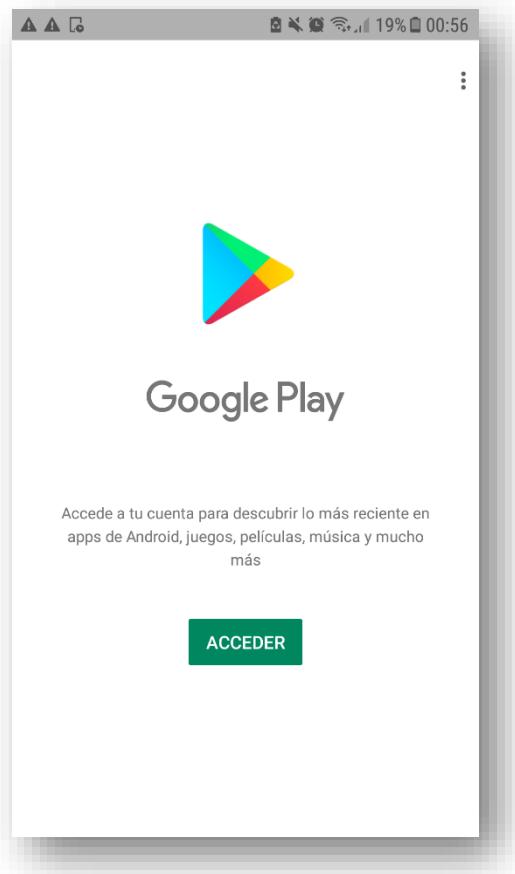
Necesitamos primero un ambiente Linux ya que todo sistema Android está basado en Linux por seguridad y flexibilidad en herramientas, usaremos la terminal “Termux” que contiene dicho ambiente en donde instalaremos la red de comunicaciones.

Termux es un emulador de Linux donde instalaremos los paquetes necesarios para crear nuestra red de comunicación entre nodos.

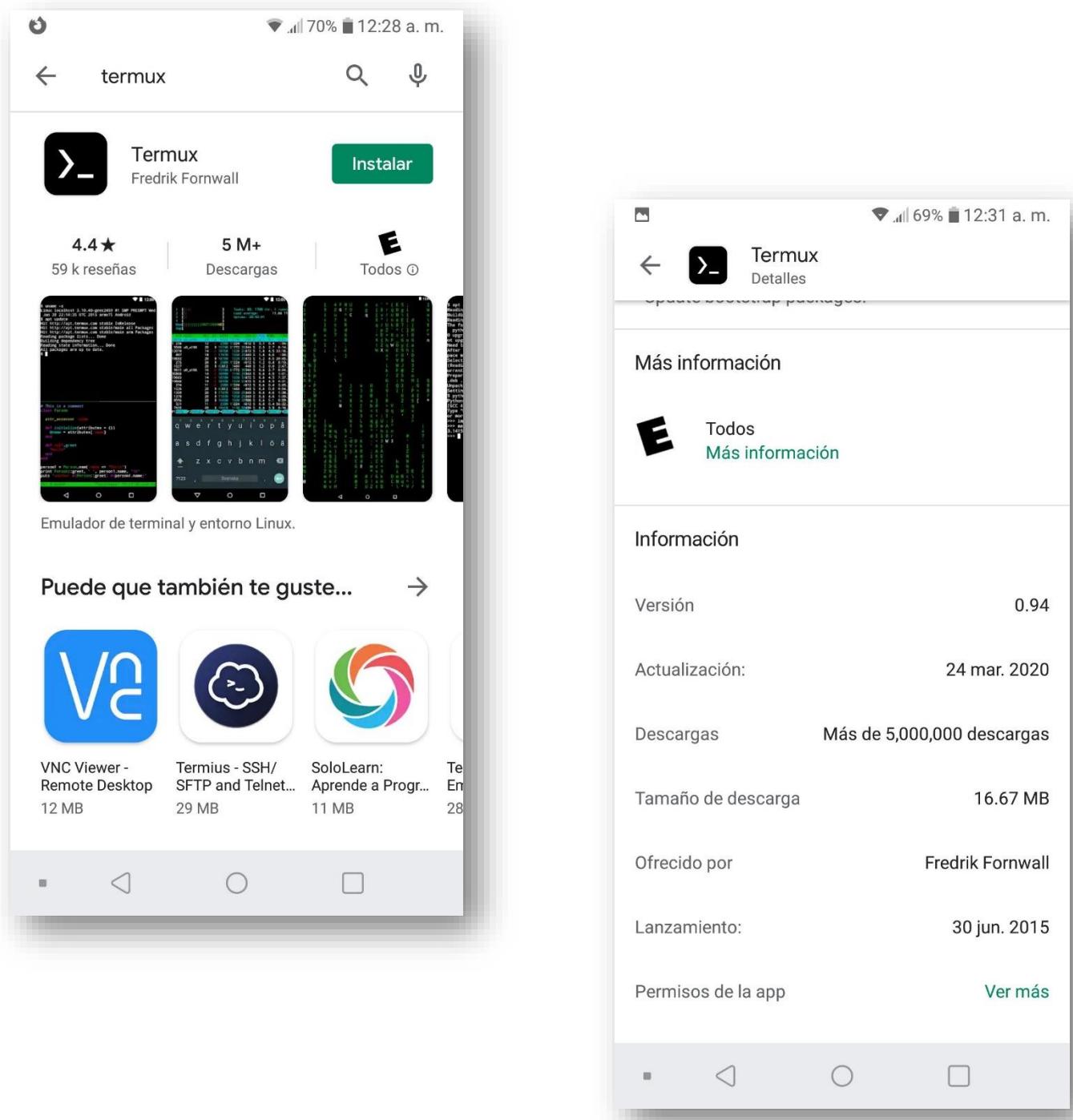
Una de las principales ventajas al usar Termux es que podrás instalar programas sin tener que “rootear” el móvil (Smartphone) esto asegura que por esta instalación no se pierde ningún tipo de garantía del fabricante.

Instalación de Termux.

Desde tu móvil ve a la aplicación icono de Google Pay (play.google.com).



Busca por aplicación “Termux”, seleccionala e iniciar el proceso de instalación.



Inicio de la aplicación Termux.

Después de iniciar tendremos que ejecutar los siguientes dos comandos para realizar actualizaciones del emulador del sistema operativo Linux:

`$ apt update`

`$ apt upgrade`

Confirmar todas las opciones Y(Yes)....

Inicio de Termux

```
Welcome to Termux!

Wiki: https://wiki.termux.com
Community forum: https://termux.com/community
Gitter chat: https://gitter.im/termux/termux
IRC channel: #termux on freenode

Working with packages:

* Search packages: pkg search <query>
* Install a package: pkg install <package>
* Upgrade packages: pkg upgrade

Subscribing to additional repositories:

* Root: pkg install root-repo
* Unstable: pkg install unstable-repo
* X11: pkg install x11-repo

Report issues at https://termux.com/issues
$ 
```

\$ apt update

```
Welcome to Termux!

Wiki: https://wiki.termux.com
Community forum: https://termux.com/community
Gitter chat: https://gitter.im/termux/termux
IRC channel: #termux on freenode

Working with packages:

* Search packages: pkg search <query>
* Install a package: pkg install <package>
* Upgrade packages: pkg upgrade

Subscribing to additional repositories:

* Root: pkg install root-repo
* Unstable: pkg install unstable-repo
* X11: pkg install x11-repo

Report issues at https://termux.com/issues
$ apt update ←
```

\$ apt upgrade

```
Welcome to Termux!

Wiki: https://wiki.termux.com
Community forum: https://termux.com/community
Gitter chat: https://gitter.im/termux/termux
IRC channel: #termux on freenode

Working with packages:

* Search packages: pkg search <query>
* Install a package: pkg install <package>
* Upgrade packages: pkg upgrade

Subscribing to additional repositories:

* Root: pkg install root-repo
* Unstable: pkg install unstable-repo
* X11: pkg install x11-repo

Report issues at https://termux.com/issues
$ apt upgrade ←
```

11. Configuración de almacenamiento “storage” dentro de Termux.

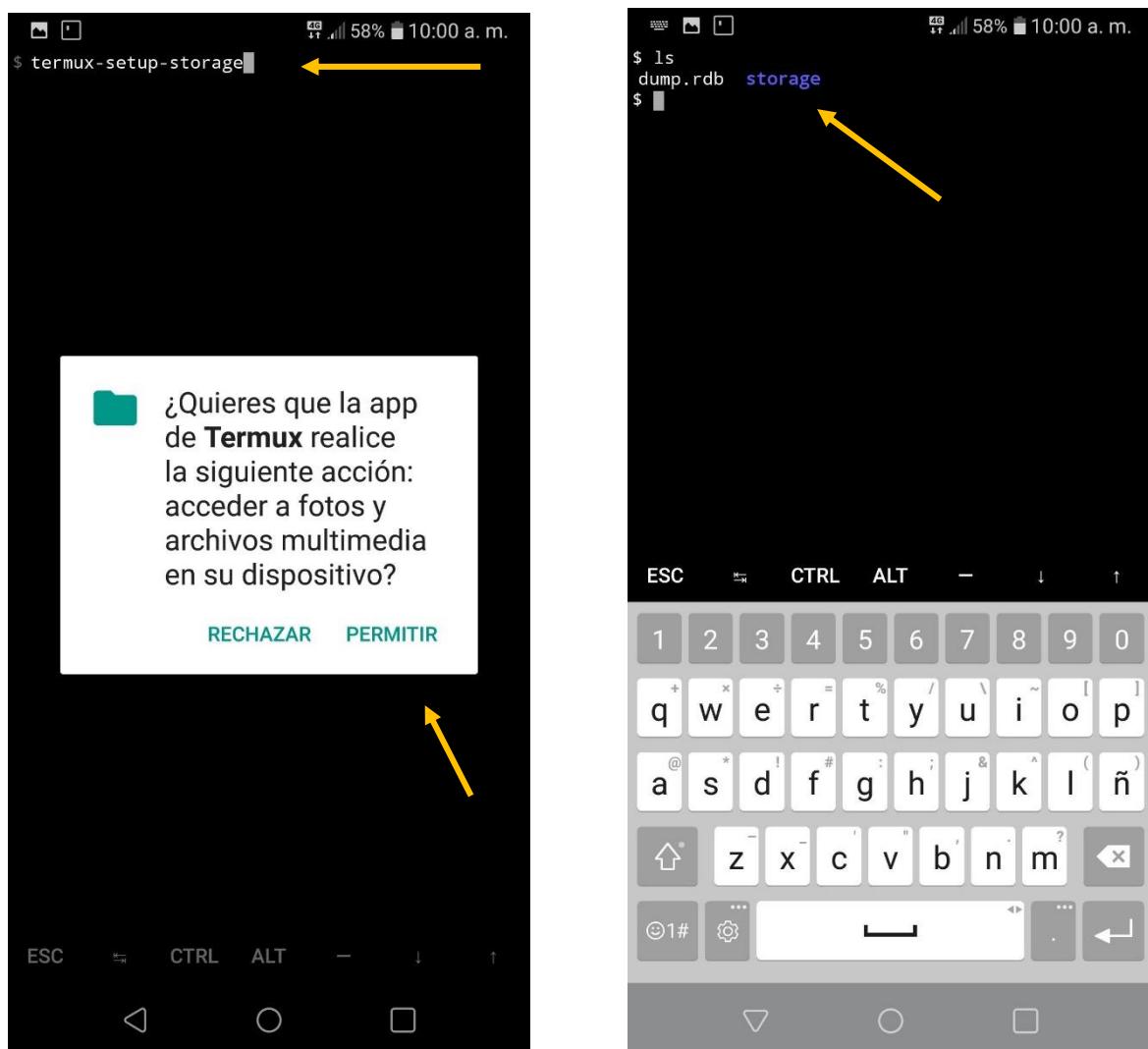
Después de haber realizado el update y upgrade del sistema Termux, empezaremos con la configuración de como poder ver el almacenamiento interno del teléfono en el sistema de Termux esto os ayudara a poder realizar intercambio de información entre Termux y nuestra información que tenemos en el teléfono.

Esto lo podemos realizar de una forma sencilla y rápida ejecutando el siguiente comando en una terminal de Termux.

`$ termux-setup-storage`

Al ejecutar el anterior comando nos aparece una ventana pidiendo la confirmación de la creación de un **storage** virtual (directorio) en Termux damos click botón “PERMITIR” y creara el enlace al almacenamiento del teléfono. Verificamos dando el comando:

`$ ls`



12. Instalación de red “Tor” e instalación de “Syncthing”.

\$ apt install tor

\$ apt install syncthing

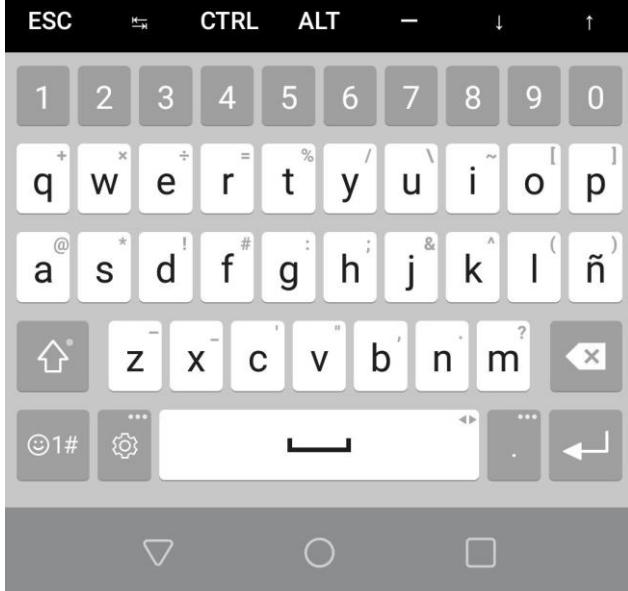
Aceptar instalación tecleando Y mayúscula en ambos casos si lo pide...

\$ apt install tor

```
$ apt install tor
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
libevent
The following NEW packages will be installed:
libevent tor
0 upgraded, 2 newly installed, 0 to remove and 0
not upgraded.
Need to get 2319 kB of archives.
After this operation, 12.6 MB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
```

\$ apt install syncthing

```
$ apt install syncthing
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
syncthing
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 6407 kB of archives.
After this operation, 19.3 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ages-24 stable/main arm syncthing arm 1.5.0 [6407
kB]
27% [1 syncthing 2193 kB/6407 kB 34%]
```



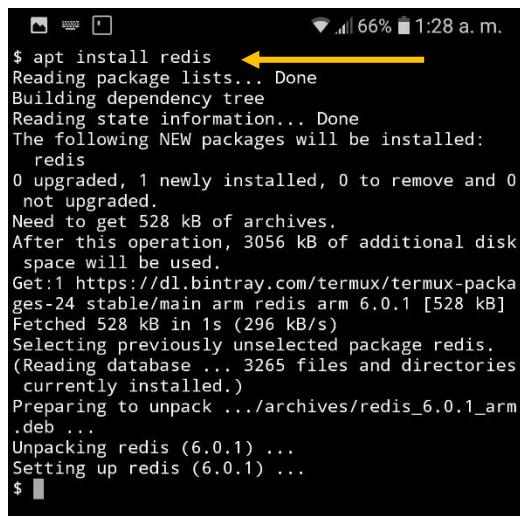
13. Instalación de Base de datos “Redis” y servidor SSH (Secure Shell).

```
$ apt install redis
```

```
$ apt install openssh
```

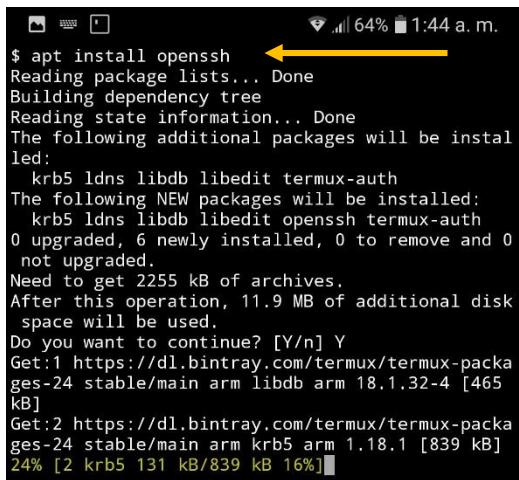
```
$ apt install sshpass
```

\$ apt install redis



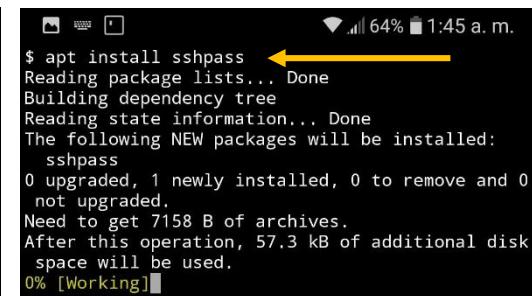
```
$ apt install redis
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  redis
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 528 kB of archives.
After this operation, 3056 kB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm redis arm 6.0.1 [528 kB]
Fetched 528 kB in 1s (296 kB/s)
Selecting previously unselected package redis.
(Reading database ... 3265 files and directories
currently installed.)
Preparing to unpack .../archives/redis_6.0.1_arm
.deb ...
Unpacking redis (6.0.1) ...
Setting up redis (6.0.1) ...
$
```

\$ apt install openssh



```
$ apt install openssh
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be instal
led:
  krb5-libs libdb5 libedit termux-auth
The following NEW packages will be installed:
  krb5-libs libdb5 libedit openssh termux-auth
0 upgraded, 6 newly installed, 0 to remove and 0
not upgraded.
Need to get 2255 kB of archives.
After this operation, 11.9 MB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm libdb5 arm 18.1.32-4 [465
kB]
Get:2 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm krb5-libs arm 1.18.1 [839 kB]
24% [2 krb5-libs 131 kB/839 kB 16%]
```

\$ apt install sshpass



```
$ apt install sshpass
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  sshpass
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 7158 B of archives.
After this operation, 57.3 kB of additional disk
space will be used.
0% [Working]
```

Hemos terminado con la instalación de la red de comunicaciones, continuamos con la configuración de los paquetes: Tor, Syncthing y Redis DB.

14. Configuración de servidor SSH en teléfono móvil (smartphone).

Habilitaremos el servidor de SSH en el teléfono móvil para poder conectarnos desde nuestra PC al móvil y poder trabajar de una forma más rápida y cómoda, así mismo nos servirá para comprobar que el servicio del servidor SSH en el móvil funciona correctamente ya que este lo utilizaremos en la red de comunicación en el Mini BlocklyChain.

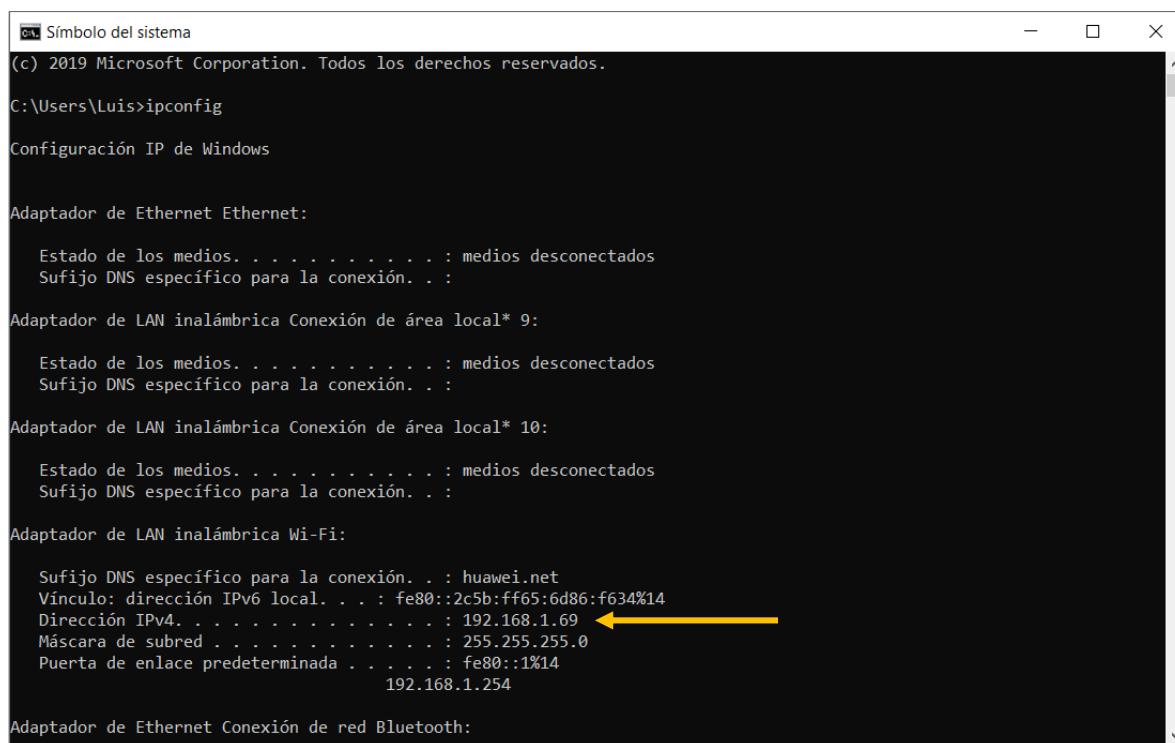
Lo primero que tenemos que hacer es conectar **a la misma red WiFi** el móvil y la PC para que se puedan ver. Las IPs o direcciones deben ser similares a 192.168.XXX.XXX los valores XXX son números variables que se asignan aleatoriamente en cada equipo.

Este ejemplo se probó en un móvil LG Q6 y una PC con Windows 10 Home.

Revisar la IP o dirección que tiene la PC conectada al WiFi deberemos abrir una terminal en Windows.

En el panel inferior donde está la lupa de buscar escribir cmd y presionar la tecla Enter. Se abrirá una terminal y en esta escribimos el comando:

C:\Users\nombre_usuario> ipconfig



```

Símbolo del sistema
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Luis>ipconfig

Configuración IP de Windows

Adaptador de Ethernet Ethernet:

Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . :

Adaptador de LAN inalámbrica Conexión de área local* 9:

Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . :

Adaptador de LAN inalámbrica Conexión de área local* 10:

Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . :

Adaptador de LAN inalámbrica Wi-Fi:

Sufijo DNS específico para la conexión. . . : huawei.net
Vínculo: dirección IPv6 local. . . : fe80::2c5b:ff65:6d86:f634%14
Dirección IPv4. . . . . : 192.168.1.69 ←
Máscara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . : fe80::1%14
192.168.1.254

Adaptador de Ethernet Conexión de red Bluetooth:
  
```

Nos mostrara la IP que tiene asignada la PC en este es la 192.168.1.69 sin embargo esta los más probable es que sea diferente en cada caso.

NOTA: debe tomarse la dirección donde dice “Dirección IPv4” no confundir con la Puerta de enlace.

Ahora en el caso del teléfono móvil en la terminal de Termux debemos teclear el siguiente comando para saber cómo se llama nuestro usuario que usaremos para conectarnos al servidor SSH que tiene nuestro teléfono, ejecutamos el siguiente comando:

\$ whoami

Posteriormente debemos darle un password a este usuario por lo que tenemos que ejecutar el siguiente comando:

\$ passwd

Nos pedirá que tecleemos un password y damos Enter, nuevamente nos pide el password confirmamos le damos el mismo y damos Enter, si ha sido exitosamente “**New password was successfully set**” en caso de marcar un error es posible que el password no se haya tecleado correctamente. Volver a realizar el procedimiento.

Y después para saber que IP tenemos en Termux tecleamos el siguiente comando, la IP esta después de la palabra “**inet**”:

\$ ifconfig -a

```
$ whoami
u0_a263
$ passwd
New password:
Retype new password:
New password was successfully set.
$ 
```



```
e 0
      TX packets 0 bytes 0 (0.0 B)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST
> mtu 1500
      inet 192.168.1.68 netmask 255.255.255.0
        broadcast 192.168.1.255
        inet6 fe80::257:c1ff:fee6:3051 prefixlen 64 scopeid 0x20<link>
          unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          txqueuelen 1000 (UNSPEC)
            RX packets 908745 bytes 947916536 (904.0 MiB)
            RX errors 0 dropped 0 overruns 0 frame errors 0
      TX packets 601034 bytes 93496881 (89.1 MiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
$ 
```

Ahora es momento de iniciar el servicio del servidor de SSH en el teléfono para que pueda recibir sesiones desde la PC. Ejecutamos el siguiente comando en la terminal de Termux, este comando no arroja ningún resultado.

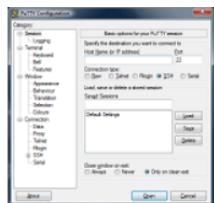
\$ sshd



Ahora tendremos que instalar un programa en la PC que se comunicara con el servidor SSH del teléfono desde la PC.

Tenemos que ir a la página <https://www.putty.org>

Seleccionar donde se encuentra el enlace “You can download PuTTY here”

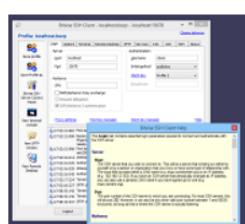


Download PuTTY

PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the with source code and is developed and supported by a group of volunteers.

You can download PuTTY [here](#).
A yellow arrow points to the download link.

Below suggestions are independent of the authors of PuTTY. They are *not* to be seen as



Bitvise SSH Client

Bitvise SSH Client is an SSH and SFTP client for Windows. It is developed and supported prof supports all features supported by PuTTY, as well as the following:

- graphical SFTP file transfer;
- single-click Remote Desktop tunneling;
- auto-reconnecting capability;
- dynamic port forwarding through an integrated proxy;
- an FTP-to-SFTP protocol bridge.

Bitvise SSH Client is **free to use**. You can [download it here](#).

Escoger la versión de 32 bit, no importa si tu sistema es de 64 bits funcionara bien.

Download PuTTY: latest release

[Home](#) | [FAQ](#) | [Feedback](#) | [Licence](#) | [Updates](#) | [Mirrors](#)
Download: [Stable](#) · [Snapshot](#) | [Docs](#) | [Changelog](#)

This page contains download links for the latest released version of PuTTY. Currently this is 0.73, released on 2019-09-29.

When new releases come out, this page will update to contain the latest, so this is a good page to bookmark or link to. Alternative Release versions of PuTTY are versions we think are reasonably likely to work well. However, they are often not the most up-to-date out the [development snapshots](#), to see if the problem has already been fixed in those versions.

Package files

You probably want one of these. They include versions of all the PuTTY utilities.

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

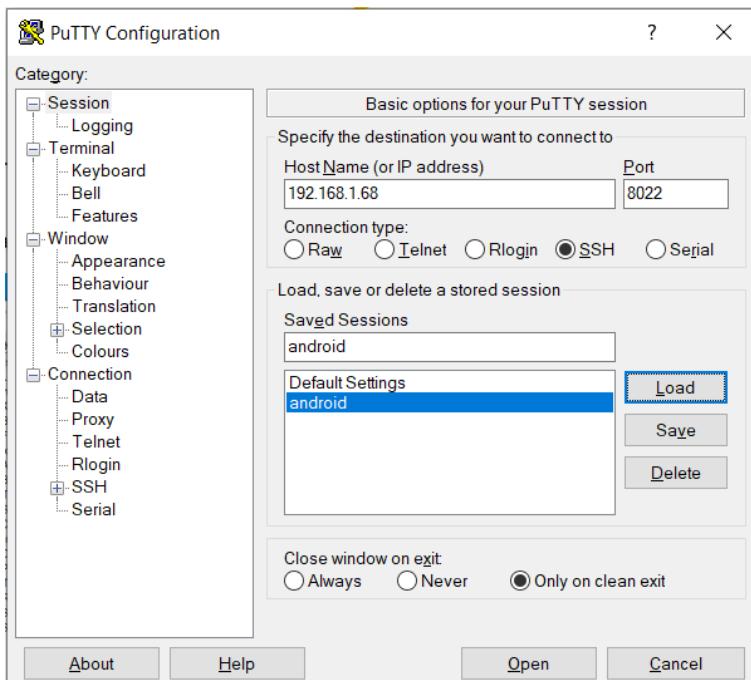
MSI ('Windows Installer')

32-bit:	putty-0.73-installer.msi	(or by FTP)	(signature)
64-bit:	putty-64bit-0.73-installer.msi	(or by FTP)	(signature)

Unix source archive

.tar.gz:	putty-0.73.tar.gz	(or by FTP)	(signature)
----------	-----------------------------------	------------------------------	-------------------------------

Ya que se haya bajado en tu PC ejecútalo e instalado con las opciones por default. Despues inicia la aplicación de PuTTY.



En esta sesión introduciremos los datos de nuestro servidor Openssh que instalamos en el teléfono móvil.

Introducir la IP del teléfono móvil.

HostName or IP address:

192.168.1.68 (ejemplo IP)

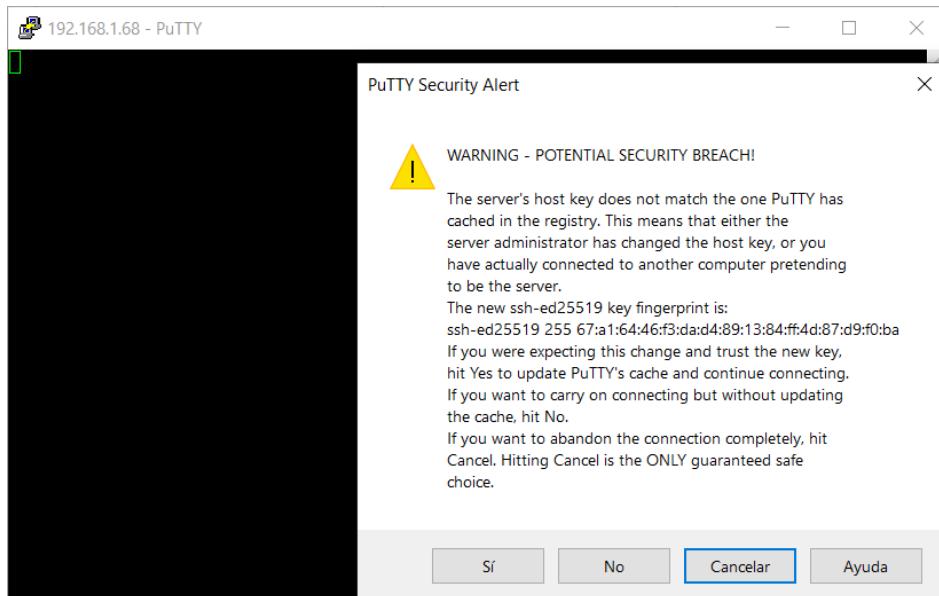
Port:

8022 (Puerto por default del servidor SSH del móvil).

Podemos dar un nombre de la sesión en “Saved Sessions” y damos click en el botón Save.

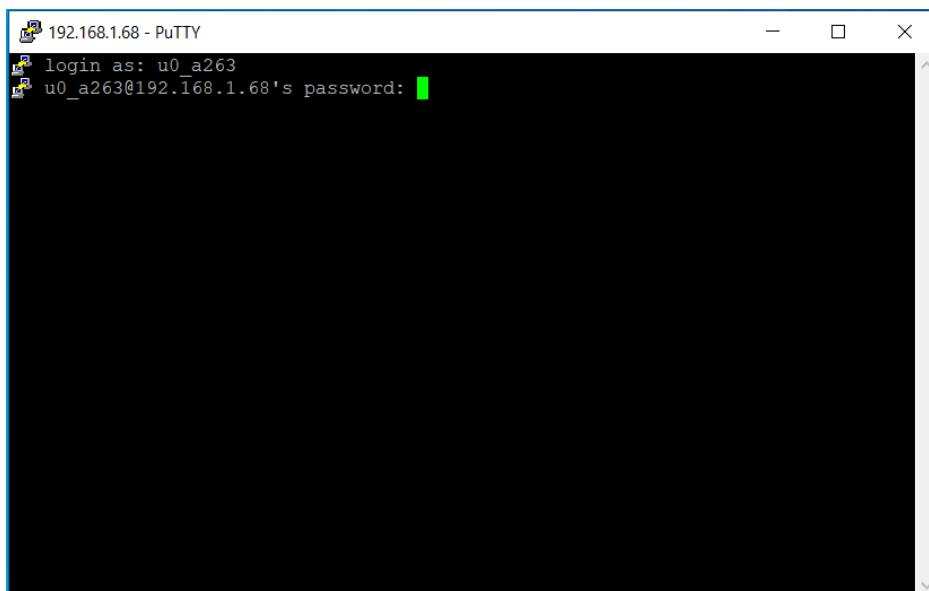
Posteriormente en la parte inferior presionamos para abrir una conexión al servidor dando el botón “Open”.

En la PC al conectarse por primera vez nos **pedirá por única vez** que confirmen la llave de cifrado de información le damos confirmación en el botón de “Sí”.

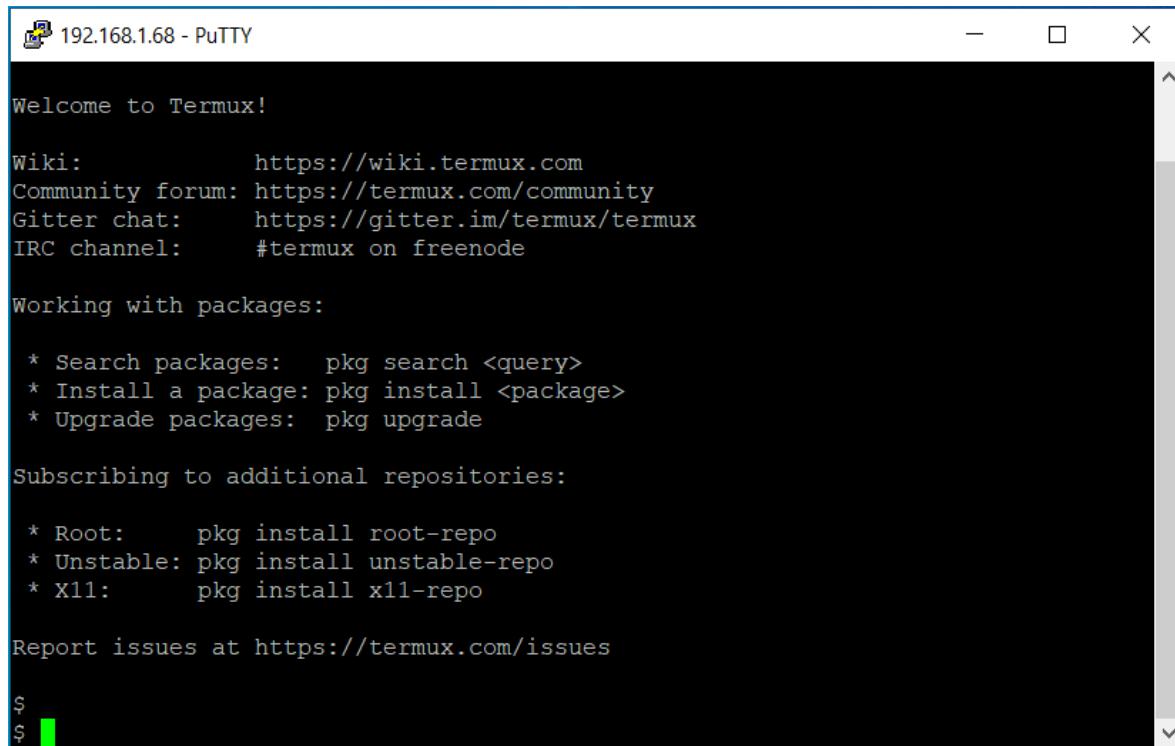


Posteriormente nos pedirá el usuario con el que nos vamos a conectar. Utilizaremos la información que sacamos con anterioridad (usuario y password).

En el **Login as:** debemos introducir nuestro usuario y dar Enter, después nos pedirá el password le damos nuevamente el botón de Enter.



Si los datos fueron los correctos, estaremos en una sesión de SSH (Secure Shell) realizada desde la PC (Cliente) en el teléfono (Servidor SSH).



```
Welcome to Termux!

Wiki: https://wiki.termux.com
Community forum: https://termux.com/community
Gitter chat: https://gitter.im/termux/termux
IRC channel: #termux on freenode

Working with packages:

* Search packages: pkg search <query>
* Install a package: pkg install <package>
* Upgrade packages: pkg upgrade

Subscribing to additional repositories:

* Root: pkg install root-repo
* Unstable: pkg install unstable-repo
* X11: pkg install x11-repo

Report issues at https://termux.com/issues

$
```

NOTA IMPORTANTE: Recordemos que la IP (dirección) de la PC y la IP (dirección) teléfono móvil conectados en la misma WiFi estarán cambiando probablemente cada vez que nos desconectemos y volvamos a conectar por lo que hay que volver a verificar que direcciones tienen cada dispositivo, esto nos asegurar el éxito de la conexión entre dispositivos por medio del servidor de SSH del teléfono y PC (Cliente).

Hasta este momento hemos podido realizar la conexión únicamente en la misma red WiFi, sin embargo, si nos movemos con el teléfono fuera de la misma red donde está la PC no podremos tener conexión ya que son diferentes redes lo que involucra pasar por otros dispositivos de comunicación más complicados, esto lo resolveremos cuando configuremos la red "Tor".

Recordemos que esta conexión la realizamos únicamente para verificar el servicio del servidor que instalamos en el teléfono y para tener un ambiente más cómodo de trabajo con una sesión remota de a PC al teléfono.

15. Configuración de red “Tor” con servicio de SSH (Secure Shell).

Con la sesión remota desde la PC empezaremos a configurar la red “Tor” con el servicio de SSH habilitado.

La importancia de tener la red “Tor” es darles la propiedad a los dispositivos de poder comunicarse en cualquier parte del mundo a través de internet sin estar en la misma red WiFi, no importa donde estemos podremos conectarnos y formar la red “Peer to Peer” entre nodos que es una funcionalidad esencial de la arquitectura Mini BlocklyChain.

La red “Tor” tiene mucha flexibilidad en su configuración ya que cuenta con varios parámetros en su archivo de configuración “torrc” este archivo se encuentra en la ruta o path (`$PREFIX/etc/tor/torrc`) en nuestro caso con la sesión de Termux desde nuestra PC lo sabremos con escribir el siguiente comando:

```
$ echo $PREFIX
```

```
/data/data/com.termux/files/usr
```

Por lo anterior el archivo que necesitamos editar estará en la ruta o path:

```
PREFIX/etc/tor/torrc que es igual a /data/data/com.termux/files/usr/etc/tor/torrc
```

Procedemos a editar el archivo de configuración usando el editor en línea de comando “vi” ejecutando el siguiente comando:

```
$ vi /data/data/com.termux/files/usr/etc/tor/torrc
```

Nos editara dicho archivo, como ejemplo:



Archivo “torrc” editado:

```

# Configuration file for a typical Tor user
## Last updated 28 February 2019 for Tor 0.3.5.1-alpha.
## (may or may not work for much older or much newer versions of Tor.)
##
## Lines that begin with "## " try to explain what's going on. Lines
## that begin with just "#" are disabled commands: you can enable them
## by removing the "#" symbol.
##
## See 'man tor', or https://www.torproject.org/docs/tor-manual.html,
## for more options you can use in this file.
##
## Tor will look for this file in various places based on your platform:
## https://www.torproject.org/docs/faq#torrc

## Tor opens a SOCKS proxy on port 9050 by default -- even if you don't
## configure one below. Set "SOCKSPort 0" if you plan to run Tor only
## as a relay, and not make any local application connections yourself.
#SOCKSPort 9050 # Default: Bind to localhost:9050 for local connections.
#SOCKSPort 192.168.0.1:9100 # Bind to this address:port too.

## Entry policies to allow/deny SOCKS requests based on IP address.
## First entry that matches wins. If no SOCKSPolicy is set, we accept
## all (and only) requests that reach a SOCKSPort. Untrusted users who
## can access your SOCKSPort may be able to learn about the connections
## you make.
#SOCKSPolicy accept 192.168.0.0/16
#SOCKSPolicy accept6 FC00::/7
#SOCKSPolicy reject *

## Logs go to stdout at level "notice" unless redirected by something
## else, like one of the below lines. You can have as many Log lines as
## you want.
##
## We advise using "notice" in most cases, since anything more verbose
## may provide sensitive information to an attacker who obtains the logs.
##
## Send all messages of level 'notice' or higher to /data/data/com.termux/files/usr/var/log/tor/notices.log
#Log notice file /data/data/com.termux/files/usr/var/log/tor/notices.log
## Send every possible message to /data/data/com.termux/files/usr/var/log/tor/debug.log
#Log debug file /data/data/com.termux/files/usr/var/log/tor/debug.log
## Use the system log instead of Tor's logfiles
#Log notice syslog
## To send all messages to stderr:
#Log debug stderr
- /data/data/com.termux/files/usr/etc/tor/torrc 1/252 0%

```

En este archivo “torrc” tendremos que agregar o usamos las líneas que tiene el archivo realizando los cambios siguientes, tres líneas que son las siguientes:

Sintaxis: SOCKSPort <número de puerto de la aplicación>

Ejemplo: SOCKSPort 9050

La variable **SOCKSPort** nos indica que este socket de comunicación sobre el protocolo TCP-IP usará por default el dispositivo móvil (teléfono) y será abierto o en uso el puerto 9050.

Sintaxis: HiddenServiceDir <Directorio donde se guardará configuración de la aplicación>

Ejemplo: HiddenServiceDir /data/data/com.termux/files/

La variable **HiddenServiceDir** nos indica que será el directorio en donde se almacenará la configuración del servicio que será usado a través de la red Tor, en este directorio se encuentran archivo de configuración y seguridad del servicio y en este directorio se encuentra un archivo llamado **hostname**, en este se encuentra la dirección para el dispositivo móvil que servirá para ser localizado en internet o en una red Wifi.

Podemos ver la dirección asignada por la red Tor para el servicio específico creado en nuestro caso está creando un servicio SSH que estará implementado sobre la red Tor, el directorio que estamos nombrando como **hidden_ssh** contendrá el archivo **hostname**. Este directorio no es necesario crearlo ya que cuando iniciemos el servicio de la red Tor será creado automáticamente, esto pasará para cada servicio que demos de alta y tendremos que asignar un directorio diferente para cada servicio.

Para ver la dirección proporcionada por la red Tor podemos usar el comando “more”, antes de usar este comando, debemos terminar la configuración del archivo “torrc” y dar de alta el servicio de la red Tor para que sea creado el directorio **hidden_ssh** y los archivos dentro de este como es el caso del archivo **hostname**.

```
$ more /data/data/com.termux/files/
```

El anterior comando nos entregara como resultado una dirección con una cadena de caracteres alfanuméricos con una extensión .onion similar a:

```
uqwthf6ojdmoybyzvudoq3x2weq7k7rjitblhba3pjbawk2nvjmx3wer.onion
```

Por ultimo debemos agregar al archivo de configuración “torrc” la variable **HiddenServicePort** este parámetro indica a la red Tor cual puerto usara la aplicación que estamos dando de alta y que usaremos a través de la red Tor.

Sintaxis: **HiddenServicePort <Puerto de salida> <IP local o IP publica>: <Puerto de salida>**

O

HiddenServicePort <Puerto de salida>

Ejemplos:

HiddenServicePort 22 127.0.0.1:8022

O

HiddenServicePort 8022

Con lo anterior hemos terminado la configuración de la red Tor para servicios genéricos y servicio SSH (Secure Shell) este servicio se utilizará para la comunicación de actualización de base de datos SQLite en donde estaremos guardando los procesos de validación de nuestro sistema Mini BlocklyChain mas adelante veremos este punto con más detalles cuando entremos en la configuración de procesos de negocio de las bases de datos Redis y SQLite.

Continuamos con la configuración de la red de comunicaciones de Mini BlocklyChain.

16. Configuración de sistema “Peer to Peer” con Syncing de forma manual.

Una arquitectura “Peer to Peer” es fundamental en un sistema de tecnológica blockchain ya que esta arquitectura da dos puntos medulares en los procesos de comunicación del sistema, uno es proporcionar la misma información actualizada (sincronizada) en cualquier momento en todos los nodos no importando si los nodos están en una red privada (Wifi) o una red pública (internet) o un híbrido de estas dos y un segundo punto de este tipo de arquitectura es que no dependen de un intermediario (servidor) para transferir, actualizar o consultar información entre nodos. Todo lo anterior se debe a que la comunicación se realiza de forma directa entre los nodos, en nuestro caso Mini BlocklyChain la comunicación se realiza directamente entre los teléfonos que forman la red sin un intermediario.

Para esta tarea usaremos la herramienta opensource Syncing que funciona basada en una arquitectura “Peer to Peer”.

La configuración de Syncing para dispositivos (teléfonos móviles) la veremos de forma manual y forma automática. La forma manual se aplica en sincronización con máximo 5 nodos, en caso de tener un número de gran volumen lo óptimo será la configuración automática, el proceso se enfoca en el registro de los nodos con los que deseamos realizar la sincronización de la información seleccionada.

Los requisitos para empezar son:

1. Tener iniciado el servicio de la red Tor.
2. (opcional – recomendable) Tener instalado una aplicación que pueda leer códigos QR, sugerimos la aplicación para Android de App inventor que es fácil y simple de instalar desde Google Play y que en más adelante en este manual lo usaremos en el apartado de “Definición y uso de bloques en Mini Blocklychain”.
3. Tener iniciado el servicio de Syncing.



Mostramos la aplicación de App Inventor que usaremos para la lectura de códigos QR que nos servirá en el futuro para el registro de los nodos en Syncthing.

El siguiente ejemplo fue realizado entre dos dispositivos móviles (teléfonos) usando los siguientes modelos:

Dispositivo móvil #1: Modelo LG Q6

Dispositivo móvil #2: Modelo Samsung

Empezaremos con iniciar los servicios de la red Tor y los servicios de Syncthing usando los siguientes comandos, abrimos dos terminales dentro de Termux y ejecutamos por separado cada comando:

\$ tor

Después de haber tecleado el comando de inicio del programa de la red Tor, debemos revisar que se haya realizado la ejecución correctamente, esto lo hacemos revisando que fue realizado en su totalidad al 100% nos marcará como porcentaje.

Ejecución del programa Tor en una terminal de Termux, verificamos que este ejecutando al 100%.

\$ tor



```
80% 11:01 p.m.
$ tor
May 23 23:00:30.932 [notice] Tor 0.4.3.5 running
on Linux with Libevent 2.1.11-stable, OpenSSL 1
.1.1g, Zlib 1.2.11, Liblzma 5.2.5, and Libzstd N
/A.
May 23 23:00:30.934 [notice] Tor can't help you
if you use it wrong! Learn how to be safe at ht
ps://www.torproject.org/download/download#warnin
g
May 23 23:00:30.939 [notice] Read configuration
file "/data/data/com.termux/files/usr/etc/tor/to
rrc".
May 23 23:00:30.970 [notice] I think we have 8 C
PUS, but only 6 of them are available. Telling T
or to only use 6. You can override this with the
NumCPUs option
May 23 23:00:30.973 [notice] Opening Socks liste
ner on 127.0.0.1:9050
May 23 23:00:30.974 [notice] Opened Socks liste
ner on 127.0.0.1:9050
May 23 23:00:30.000 [notice] Parsing GEOIP IPv4
file /data/data/com.termux/files/usr/share/tor/g
eoip.
May 23 23:00:32.000 [notice] Parsing GEOIP IPv6
file /data/data/com.termux/files/usr/share/tor/g
eoip6.
May 23 23:00:34.000 [notice] Bootstrapped 0% (st
arting): Starting
May 23 23:00:34.000 [notice] Starting with guard
context "default"
May 23 23:00:35.000 [notice] Bootstrapped 5% (co
nn): Connecting to a relay
May 23 23:00:36.000 [notice] Bootstrapped 10% (c
onn_done): Connected to a relay
May 23 23:00:36.000 [notice] Bootstrapped 14% (h
andshake): Handshaking with a relay
May 23 23:00:36.000 [notice] Bootstrapped 15% (h
andshake_done): Handshake with a relay done
May 23 23:00:36.000 [notice] Bootstrapped 20% (o
nehop_create): Establishing an encrypted directo
ry connection
May 23 23:00:36.000 [notice] Bootstrapped 25% (r
```



```
75% 1:33 a.m.
ps://www.torproject.org/download/download#warnin
g
May 24 01:33:32.982 [notice] Read configuration
file "/data/data/com.termux/files/usr/etc/tor/to
rrc".
May 24 01:33:33.007 [notice] I think we have 8 C
PUS, but only 6 of them are available. Telling T
or to only use 6. You can override this with the
NumCPUs option
May 24 01:33:33.010 [notice] Opening Socks liste
ner on 127.0.0.1:9050
May 24 01:33:33.010 [notice] Opened Socks liste
ner on 127.0.0.1:9050
May 24 01:33:33.000 [notice] Parsing GEOIP IPv4
file /data/data/com.termux/files/usr/share/tor/g
eoip.
May 24 01:33:34.000 [notice] Parsing GEOIP IPv6
file /data/data/com.termux/files/usr/share/tor/g
eoip6.
May 24 01:33:35.000 [notice] Bootstrapped 0% (st
arting): Starting
May 24 01:33:37.000 [notice] Starting with guard
context "default"
May 24 01:33:38.000 [notice] Bootstrapped 5% (co
nn): Connecting to a relay
May 24 01:33:38.000 [notice] Bootstrapped 10% (c
onn_done): Connected to a relay
May 24 01:33:39.000 [notice] Bootstrapped 14% (h
andshake): Handshaking with a relay
May 24 01:33:39.000 [notice] Bootstrapped 15% (h
andshake_done): Handshake with a relay done
May 24 01:33:39.000 [notice] Bootstrapped 75% (e
nough_dirinfo): Loaded enough directory info to
build circuits
May 24 01:33:39.000 [notice] Bootstrapped 90% (a
p_handshake_done): Handshake finished with a rel
ay to build circuits
May 24 01:33:39.000 [notice] Bootstrapped 95% (c
ircuit_create): Establishing a Tor circuit
May 24 01:33:42.000 [notice] Bootstrapped 100% (d
one): Done
```

Posteriormente ejecutamos el comando de syncthing:

\$ syncthing

Después de ejecutarse dicho comando nos abrirá una página de administración en el browser de nuestro teléfono en su caso si no se abre automáticamente, podemos nosotros ir a cualquier browser (navegador) que tengamos instalado donde normalmente navegamos en internet y podemos poner la siguiente l:

<http://127.0.0.1:8384>

Nos abrirá la pantalla de administración de la herramienta que nos ayudará a sincronizar nuestra información entre todos los nodos (teléfonos) del sistema.



```
$ syncthing
[monitor] 04:02:07 INFO: Default folder created
and/or linked to new config
[start] 04:02:07 INFO: syncthing v1.5.0 "Fermium
Flea" (go1.14.2 android-arm) builder@6bdf862223
8a 2020-05-11 08:38:11 UTC
[start] 04:02:07 INFO: Generating ECDSA key and
certificate for syncthing...
[start] 04:02:08 INFO: Default folder created an
d/or linked to new config
[start] 04:02:08 INFO: Default config saved. Edi
t /data/data/com.termux/files/home/.config/synct
hing/config.xml to taste (with Syncthing stopped
) or use the GUI
[OWEPS] 04:02:08 INFO: My ID: OWEPSNA-6RZI657-SK
VOU65-V44BC5Z-CXZOJI4-C3JC7HM-IPY4V35-JQAKPAW
[OWEPS] 04:02:09 INFO: Single thread SHA256 perf
ormance is 12 MB/s using crypto/sha256 (12 MB/s
using minio/sha256-simd).
[OWEPS] 04:02:11 INFO: Hashing performance is 11
.17 MB/s
[OWEPS] 04:02:11 INFO: Migrating database to sch
ema version 1...
```



Otros dispositivos

- [Cambios recientes](#)
- [Añadir un dispositivo](#)

Ya que tenemos abierta la pantalla de administración de “syncthing” procedemos a dar de alta el nodo o nodos que deseamos sincronizar la información. En este punto es donde ocuparemos el programa que lee códigos QR.

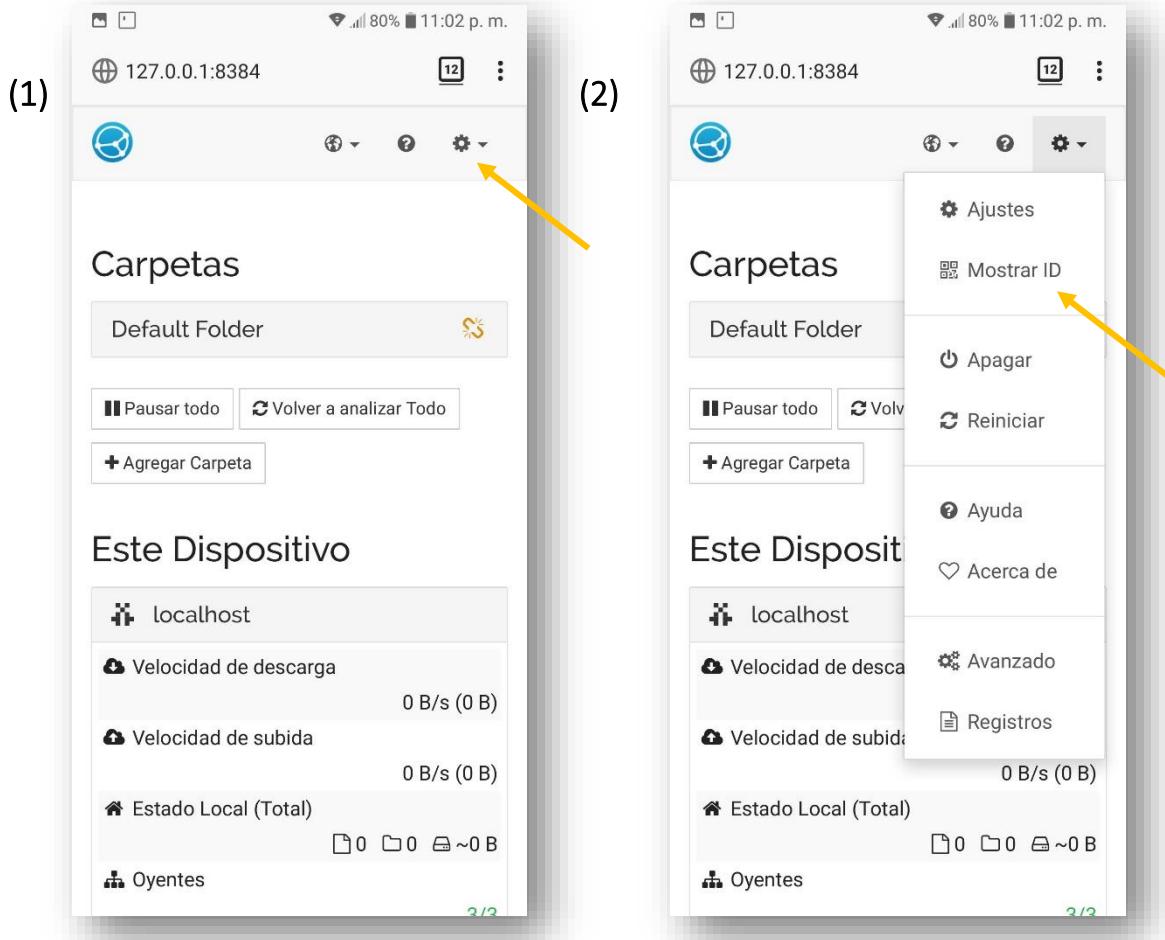
El programa de syncthing cuando se inicia por primera vez se crea un identificador único del teléfono que está compuesto por un grupo de ocho conjuntos de caracteres alfanuméricicos en mayúsculas, este identificador (ID) es el que registraremos en el nodo o nodos que deseamos sincronizar la información.

En nuestro caso el ID del teléfono LG Q6 se tendrá que registrar en el teléfono Sansumg y el ID del teléfono Sansumg se tendrá que registrar en el LG Q6. Deben estar en ambos teléfonos para que pueda funcionar correctamente.

Realizaremos los pasos del registro del teléfono móvil Sansumg en el teléfono LG Q6.

Primero (1) en la parte superior de la pantalla de administración (navegador de internet) del teléfono Sansumg con syncthing daremos click en la pestaña de menú lado superior derecho.

Segundo (2) paso nos aparecerá un menú, en este damos click en “Mostrar ID” del Sansumg.

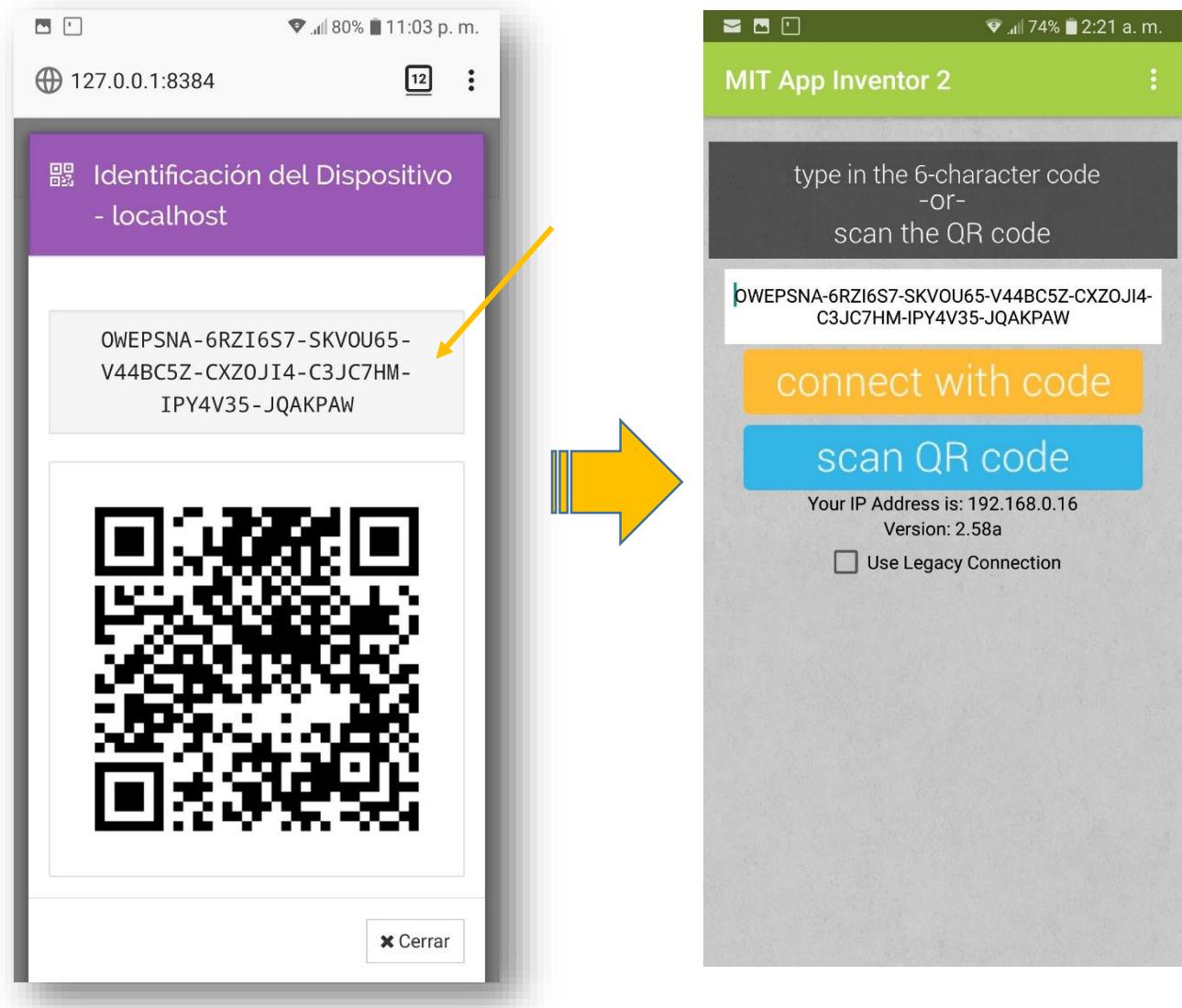


Al dar click en “Mostrar ID” nos aparecerá la siguiente pantalla que es un código QR del teléfono Sansumg en nuestro caso el ID que identifica al teléfono es:

OWEPSNA-6RZI6S7-SKVOU65-V44BC5Z-CXZOJI4-C3JC7HM-IPY4V35-JQAKPAW

Después en el teléfono LG Q6, el programa que nos ayudara a capturar este código QR del Sansumg es el que sugerimos de la aplicación App Inventor (opcional), aunque en caso de no tenerlo también simplemente podemos introducirlo de forma manual cuando iniciemos el registro en el LG Q6, sin embargo, para evitar algún error sugerimos usarlo.

Usando programa de App inventor instalado en el móvil LG Q6 para capturar código QR del teléfono remoto Sansumg que deseamos sincronizar.



Posteriormente copiamos al portapapeles el código QR en el programa App inventor dando click en el código capturado, escogemos “SELEC TODO” → “COPIAR”.

Con esta información procedemos a registrar el Sansumg en el LG Q6. En la pantalla de administración nos desplazamos a la parte inferior nos situamos en donde dice “Otros dispositivos” y damos click en el botón de “Añadir un dispositivo”, introducimos el ID del Sansumg y le damos nombre de registro.

Posteriormente en la pestaña superior de “Compartir” damos click y en esta marcamos la opción inferior en folder “Default” con esto estaremos compartiendo un directorio que se creó por Default llamado Sync en nuestro dispositivo.

Después en la parte superior damos click en la pestaña “Avanzado” y seleccionamos la opción de compresión de datos en “Todos los datos”.

Por ultimo damos click en el botón inferior de guardar, Terminamos el registro en un nodo, este mismo proceso debe hacerse en el teléfono o teléfonos remotos que deseamos sincronizar.

Screenshot 1: Main Dashboard

ID del Dispositivo: /IQXYQ-ILHCOID-BUPFIWU-MJJUNQD
El ID del dispositivo que hay que introducir aquí se puede encontrar en el diálogo "Acciones > Mostrar ID" en el otro dispositivo. Los espacios y las barras son opcionales (ignorados). Cuando añada un nuevo dispositivo, tenga en cuenta que este debe añadirse también en el otro lado.

Nombre del Dispositivo: SANSUMG
Se muestra en lugar del ID del dispositivo en el estado del grupo (cluster). Se actualizará al nombre que el dispositivo anuncia si se deja vacío.

Screenshot 2: Add Device Dialog

Agregar el dispositivo SANSUMG

Presentador
Añadir dispositivos desde el introductor a nuestra lista de dispositivos, para las carpetas compartidas mutuamente.

Aceptar automáticamente
Crear o compartir automáticamente carpetas que este dispositivo anuncia en la ruta por defecto.

Compartir carpetas con dispositivo
Selecciona las carpetas para compartir con este dispositivo. Select All Deselect All
✓ Default Folder

Screenshot 3: Advanced Tab

General Compartiendo

Avanzado

Direcciones: dynamic
Enter comma separated ("tcp://ip:port", "tcp://host:port") addresses or "dynamic" to perform automatic discovery of the address.

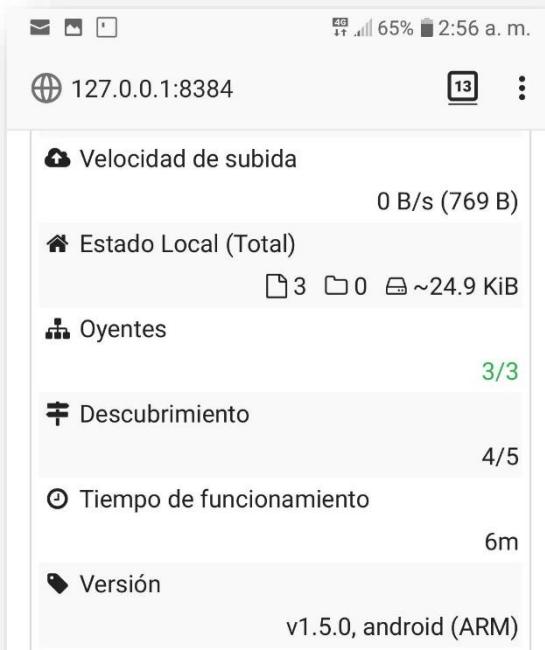
Compresión: Todos los datos

Límites de velocidad del dispositivo
Límite de descarga (KiB/s)
Límite de subida (KiB/s)

Guardar Mostrar QR Cerrar

Al guardar e registrar en ambos teléfonos esperaremos un periodo máximo de 30 segundos en lo que se encuentran los dispositivos y aparecerá como buena la conexión entre dispositivos dando una confirmación en color verde.

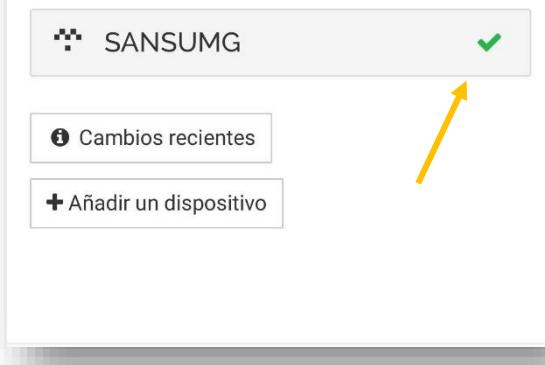
Dispositivo #1 LG Q6



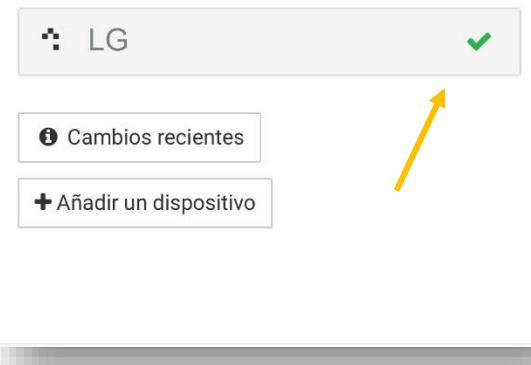
Dispositivo #2 Sansumg



Otros dispositivos



Otros dispositivos



Toda la información que se encuentre en el directorio **Sync** situado en la ruta `/data/data/com.termux/file/home/Sync` se sincronizará, cualquier cambio será copiado y sincronizado.

Configuración de sistema “Peer to Peer” con Syncthing de forma automática para uso en Mini BlocklyChain.

Ahora realizaremos la configuración de forma automática, anteriormente realizamos el proceso manual esto es útil si manejamos una cantidad mínima de nodos, sin embargo, en el caso de tener una cantidad en gran volumen de nodos sería ineficiente por lo que tendremos la herramienta de SyncthingManager mas adelante para configurarlo de forma automática usando comandos automatizados en línea.

Configuración de base de datos Redis para nodos modo (Esclavo).

Configuración del archivo `/data/data/com.termux/files/usr/etc/redis.conf` de la base de datos Redis (**Esclavo**) para teléfonos móviles Android.

Agregar los siguientes cambios o directivas en el archivo, guardar cambios e iniciar servidor Redis.

Primero, encuentre y descómete quite el carácter # de la línea **slaveof**. Esta directiva toma la dirección IP y el puerto que utiliza para contactar de forma segura con el servidor maestro Redis, separados por un espacio. Por defecto, el servidor Redis escucha en 6379 en la interfaz local, pero cada uno de los métodos de seguridad de la red modifica el valor predeterminado de alguna manera para terceros.

Los valores que use dependerán del método que utilizó para proteger su tráfico de red:

red aislada: utilice la dirección IP de la red aislada y el puerto Redis (6379) del servidor maestro (por ejemplo, esclavo de dirección_IP_simple 6379).

stunnel o spiped: use la interfaz local (127.0.0.1) y el puerto configurado para tunelizar el tráfico.

PeerVPN: utilice la dirección IP VPN del servidor maestro y el puerto Redis normal.

El general los cambios serían:

slaveof ip_de_contacto_servidor_maestro puerto_para_contacto_maestro

Ejemplo: **slaveof** 192.168.1.69 6379

masterauth tu_redis_maestro_password

Ejemplo: **masterauth** sdfssdfsdf12WqE34Rfgthtdfd

requirepass tu_redis_esclavo_password

Ejemplo: **requirepass** asdsjdsh34scs67sdFGbbnh

Guardar y ejecutar el servidor desde terminal Termux con el siguiente comando.

```
$ redis-server redis.conf
```

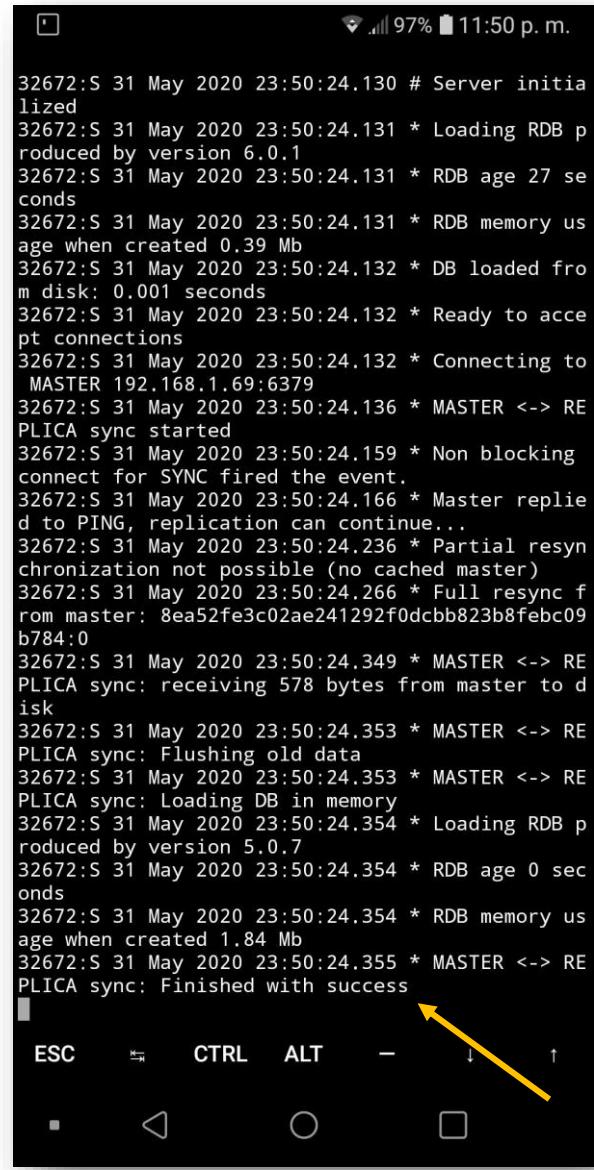
Después de la ejecución podremos ver como se sincroniza con el servidor (**Maestro**) de Windows 10.

Archivo de configuración **redis.conf**



```
$ pwd
/data/data/com.termux/files/usr/etc
$ ls
alternatives      inputrc    redis.conf
apt              krb5.conf  ssh
bash.bashrc       motd      tls
bash_completion.d profile   tmux.conf
dump.rdb          profile.d wgetrc
$
```

\$ redis-server redis.conf



```
32672:S 31 May 2020 23:50:24.130 # Server initialized
32672:S 31 May 2020 23:50:24.131 * Loading RDB produced by version 6.0.1
32672:S 31 May 2020 23:50:24.131 * RDB age 27 seconds
32672:S 31 May 2020 23:50:24.131 * RDB memory usage when created 0.39 Mb
32672:S 31 May 2020 23:50:24.132 * DB loaded from disk: 0.001 seconds
32672:S 31 May 2020 23:50:24.132 * Ready to accept connections
32672:S 31 May 2020 23:50:24.132 * Connecting to MASTER 192.168.1.69:6379
32672:S 31 May 2020 23:50:24.136 * MASTER <-> REPLICAS sync started
32672:S 31 May 2020 23:50:24.159 * Non blocking connect for SYNC fired the event.
32672:S 31 May 2020 23:50:24.166 * Master replied to PING, replication can continue...
32672:S 31 May 2020 23:50:24.236 * Partial resynchronization not possible (no cached master)
32672:S 31 May 2020 23:50:24.266 * Full resync from master: 8ea52fe3c02ae241292f0dcbb823b8feb09b784:0
32672:S 31 May 2020 23:50:24.349 * MASTER <-> REPLICAS sync: receiving 578 bytes from master to disk
32672:S 31 May 2020 23:50:24.353 * MASTER <-> REPLICAS sync: Flushing old data
32672:S 31 May 2020 23:50:24.353 * MASTER <-> REPLICAS sync: Loading DB in memory
32672:S 31 May 2020 23:50:24.354 * Loading RDB produced by version 5.0.7
32672:S 31 May 2020 23:50:24.354 * RDB age 0 seconds
32672:S 31 May 2020 23:50:24.354 * RDB memory usage when created 1.84 Mb
32672:S 31 May 2020 23:50:24.355 * MASTER <-> REPLICAS sync: Finished with success
```

Instalación de herramienta de administración de syncthing para nodos. Procedemos a la instalación de **SyncthingManager**.

Primero instalamos lo que necesitara para que funcione correctamente SyncthingManager.

```
$ apt install Python
```

```
$ pip3 install --upgrade pip
```

```
$ npm install syncthingmanager
```

La herramienta SyncthingManager nos ayudara para la sincronización “peer to Peer” de una forma automática y no manual para nuevos y existentes nodos.

```
$ apt install python
Reading package lists... Done
Building dependency tree
Reading state information... Done
python is already the newest version (3.8.3).
0 upgraded, 0 newly installed, 0 to remove and 0
not upgraded.
$ 

$ pip3 install --upgrade pip
Requirement already up-to-date: pip in /data/dat
a/com.termux/files/usr/lib/python3.8/site-pac
kes (20.1.1)
$ 

$ pip3 install syncthingmanager
Requirement already satisfied: syncthingmanager
in /data/data/com.termux/files/usr/lib/python3.8/
site-packages (0.1.0)
Requirement already satisfied: syncthing in /dat
a/data/com.termux/files/usr/lib/python3.8/site-p
ackages (from syncthingmanager) (2.3.1)
Requirement already satisfied: python-dateutil==
2.6.1 in /data/data/com.termux/files/usr/lib/pyt
hon3.8/site-packages (from syncthing->syncthingm
anager) (2.6.1)
Requirement already satisfied: requests==2.18.4
in /data/data/com.termux/files/usr/lib/python3.8/
site-packages (from syncthing->syncthingmanager
) (2.18.4)
Requirement already satisfied: six>=1.5 in /data
/data/com.termux/files/usr/lib/python3.8/site-pa
ckages (from python-dateutil==2.6.1->syncthing->
syncthingmanager) (1.15.0)
Requirement already satisfied: chardet<3.1.0,>=3
.0.2 in /data/data/com.termux/files/usr/lib/pyth
on3.8/site-packages (from requests==2.18.4->sync
thing->syncthingmanager) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in
/data/data/com.termux/files/usr/lib/python3.8/si
te-packages (from requests==2.18.4->syncthing->
syncthingmanager) (2.6)
Requirement already satisfied: certifi>=2017.4.1
7 in /data/data/com.termux/files/usr/lib/python3
.8/site-packages (from requests==2.18.4->syncthi
ng->syncthingmanager) (2020.4.5.1)
Requirement already satisfied: urllib3<1.23,>=1.
21.1 in /data/data/com.termux/files/usr/lib/pyth
on3.8/site-packages (from requests==2.18.4->sync
thing->syncthingmanager) (1.22)
$ 
```

17. Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).

App Inventor es un entorno de desarrollo de software creado por Google Labs para la elaboración de aplicaciones destinadas al sistema operativo Android. El usuario puede, de forma visual y a partir de un conjunto de herramientas básicas, ir enlazando una serie de bloques para crear la aplicación. El sistema es gratuito y se puede descargar fácilmente de la web. Las aplicaciones creadas con App Inventor son muy fáciles de crear debido a que no se necesita conocimiento de ningún tipo de lenguaje de programación.

Todos los ambientes actuales que usan la tecnología de Blockly como son AppyBuilder y Thunkable entre otros tienen su versión gratuita, su forma de uso puede ser a través de internet en sus diversos sitios o también puede ser instalado en casa.

Los bloques que forman la arquitectura de Mini BloclyChain han sido probados en App inventor y AppyBuilder sin embargo por su optimización de código deberán funcionar en las otras plataformas.

Versiones de por internet:

App Inventor.

<https://appinventor.mit.edu/>

AppyBuilder.

<http://appybuilder.com/>

Thunkable.

<https://thunkable.com/>

Versión para ser instalada en tu equipo (PC):

<https://sites.google.com/site/aprendeappinventor/instala-app-inventor>

Ambiente para desarrolladores de bloques Blockly.

<https://editor.appybuilder.com/login.php>

18. ¿Qué es Prueba de Cuanto (PQu - Proof Quantum)?

PoQu. - “Proof of Quantum” es un algoritmo de consenso desarrollada para Mini BlocklyChain, esta prueba es una variante de la Prueba de Trabajo (PoW - “Proof of Work”) que funciona de la siguiente forma.

La Prueba de Cuanto (PoQu) al inicio se ejecuta con el mismo algoritmo de la “Prueba de Trabajo” (PoW) se fundamenta en poner a trabajar el procesador del dispositivo (PC, Servidor, Tableta o Teléfono móvil) para obtener una cadena de caracteres que es un acertijo matemático llamado “hash”.

Recordemos que un “hash” es un algoritmo o proceso matemático que al introducir una frase o algún tipo de información digital como archivos de texto, programa, imagen, video, sonido u otro tipo diverso de información digital nos da como resultado un carácter alfanumérico que representa la firma digital que lo representa de forma única y no repetible del dato, el algoritmo de “hash” es unidireccional esto quiere decir que al introducir un dato para obtener su firma “hash” su proceso inverso no podemos realizarlo, al tener una firma “hash” no podemos saber de qué información fue obtenido esta propiedad nos da un ventaja de seguridad para procesar la información que enviamos por internet. ¿Cómo funciona? imaginemos enviar cualquier tipo de información a través de canales no seguros y acompañar dicha información con su respectivo “hash origen”, el receptor al recibir la información podrá sacar el “hash” de la información recibida lo llamaremos “hash destino” y cotejarlo con el “hash origen” si ambos “hashes” son iguales podemos confirmar que la información no ha sido alterada en el canal que se envió, es solo un ejemplo en donde se usa actualmente este tipo de proceso de seguridad de información.

En la actualidad se tiene diferentes tipos de algoritmos o procesos tipo “hash” estos difieren en el nivel de seguridad. Los más utilizados o conocidos son: MD5, SHA256 y SHA512.

Ejemplo de SHA256:

Tenemos una cadena o frase como sigue: “Mini BlocklyChain es modular”

Si aplicamos un “hash” del tipo SHA256 a la cadena anterior nos dará el siguiente “hash”.

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2af827e8804db8

La anterior cadena alfanumérica es la firma que representa la frase del ejemplo anterior

Para más ejemplo podemos usar en internet el sitio:

<https://emn178.github.io/online-tools/sha256.html>

En el caso del algoritmo de “Prueba de Trabajo” (PoW) trabaja usando poder de cómputo para poder obtener un “hash” predefinido.

Imaginemos que tenemos el “hash” anterior que sacamos de la cadena “Mini BlocklyChain es modular”.

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2af827e8804db8

A este “hash” en su inicio ponemos el parámetro de dificultad que es simplemente poner ceros “0” en el inicio, es decir si decimos que la dificultad es de 4 tendrá “**0000**” + “hash” a este le llamaremos “hash semilla”

0000 f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2af827e8804db8

Ahora teniendo en cuenta que sabemos la información de entrada que es la cadena: “Mini BlocklyChain es modular” le agregamos al final de la cadena un numero empezando desde cero “0” y sacamos su hash a este lo llamaremos “hash nonce”:

f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2af827e8804db80

Sacamos “hash nonce”:

7529f3ad273fc8a9eff12183f8d6f886821900750bb6b59c1504924dfd85a7c8

Después realizamos una comparación del nuevo “hash nonce” con el “hash semilla” si son iguales el nodo que primero encuentre la igualdad ganara la ejecución de procesar la transacción en curso. Como podemos ver este proceso está basado en probabilidad y fuerza de computo que tenga el dispositivo lo que da a la prueba de trabajo “Proof of Work” una equidad de consenso para todos los nodos.

En caso de no coincidir el “hash semilla” con el “hash nonce”, se incrementa la dificultad en uno y se vuelve a sacar el “hash nonce”, el número que se esté incrementando le llamaremos número “nonce”, se compara con el “hash semilla” hasta que coincidan o sean iguales.

Como podemos ver el número “nonce” o incremento es el que ayudara a obtener el “hash” de igualdad.

Teniendo como base el algoritmo de “Prueba de Trabajo” (PoW), el algoritmo Prueba de Cuanto (PoQu) se basa en obtener el número “nonce” como lo hace PoW y usando una dificultad de nivel mínimo que va desde 1 a 5, este sirve únicamente para que el dispositivo móvil gane el derecho de ser un candidato para ganar el consenso.

La Prueba de Cuanto (PoQu), se activa cuando el teléfono móvil a terminado el PoW mínimo y gana el pase para obtener un numero de probabilidad en el sistema de QRNG.

El QRNG (Quantum Random Number Generator) es un Generador de números cuánticos aleatorios, este sistema está basado en generar verdaderos números aleatorios basados en

la mecánica cuántica es el sistema más seguro en la actualidad para generar este tipo de números. Para más detalles ver Anexo “Computación Cuántica con OpenQbit”.

Mini BlocklyChain puede implementar los dos tipos de conceso PoW mínimo y PoQu.

La Prueba de Cuanto (PoQu) se basa en obtener el número “nonce” este número en la Prueba de Cuanto (PoQu) es conocido como “Magic Number” con este el sistema “Peer to Peer” confirmara si es correcto el número y posteriormente se obtendrá un numero aleatorio con el pool de servidores QRNG. Este número aleatorio se registrará en todos los nodos, se creará una lista que contendrá $((\text{Sumatoria Nodos}) / 2)) + 1$ y de esta lista se escogerá el que tenga el mayor porcentaje de probabilidad para ser el candidato ganador del consenso (PoQu) y este ejecutará la cola de transacción en curso.

El algoritmo Prueba de Cuanto (PoQu) también usa prueba del **NIST** (National Institute of Standards and Technology) para asegúranos que los números aleatorios del QRNG son verdaderamente números aleatorios.

<https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf>

En Mini BlocklyChain tenemos implementado un bloque para la PoW y un bloque para el caso de la PoQu.

Estos bloques usan un tipo de hash: SHA256 para su uso libre, para uso comercial se tiene un SHA512 y otros tipos de hash según sea el requerimiento.

Para ver más detalles del concepto de HASH ver:

https://es.wikipedia.org/wiki/Funcion_hash

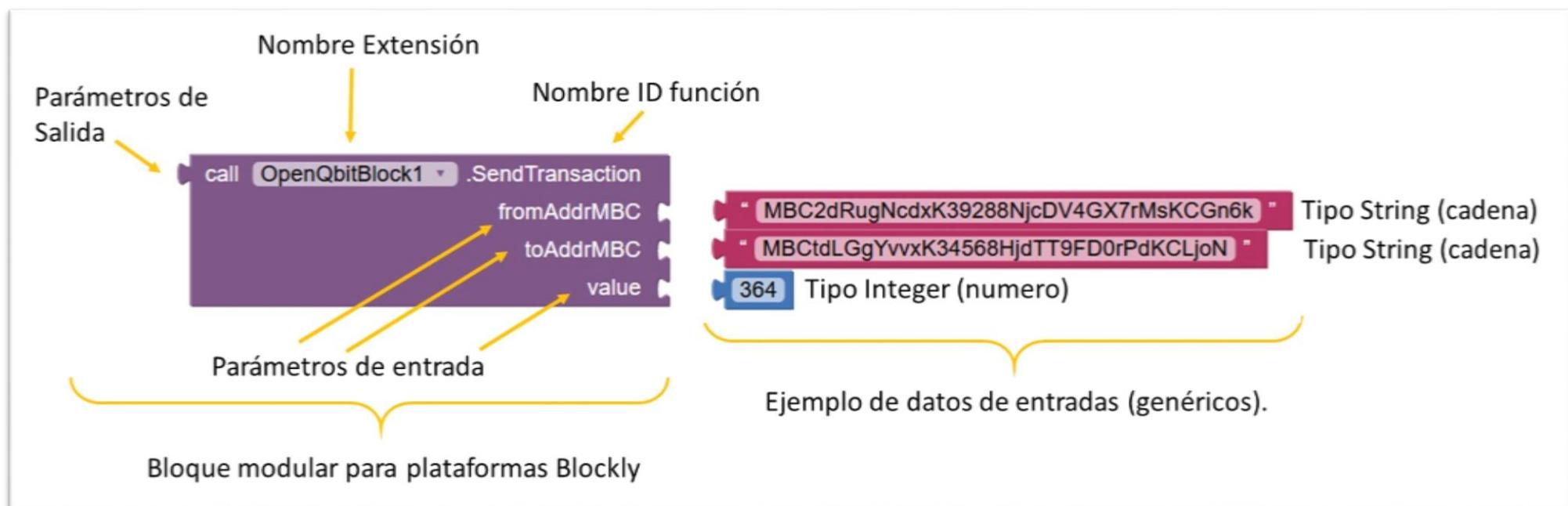
NOTA: La Prueba de Trabajo (PoW) usada en los teléfonos móviles solamente pueden usar una dificultad máxima de 5 ya que el procesamiento matemático de estos dispositivos no es dedicado como los servidores o PC. El algoritmo (PoW) únicamente lo usamos para obtener la oportunidad de obtener su pase o permiso para entrar al sistema del Generador Cuántico de Números Aleatorios (QRNG – Quantum Random Number Generator) y con este ejecutar el algoritmo de la Prueba de Cuanto (PoQu).

En teléfonos móviles no usar una dificultad máxima a 5 ya que el sistema podrá bloquearse y no responder debidamente.

19. Definición y uso de bloques en Mini BlocklyChain.

Empezaremos con explicar la distribución de los datos que tendrá todos los bloques, su sintaxis de uso y configuración.

En el siguiente ejemplo podemos ver un bloque modular y sus parámetros de entrada y salida, así como los tipos de datos de entrada, estos datos pueden ser de tipo String (cadena de caracteres) o Integer (numero entero o decimal). Mostramos como es su uso y configurarlo para su funcionamiento adecuado.



Cada bloque modular tendrá su descripción y se nombrara en caso de tener alguna(s) dependencia(s) obligatoria(s) u opcional(es) de otros bloques usados como parámetros de entrada se anunciará el proceso de integración.

Bloque para crear una lista de cadena de caracteres (String) temporal en un arreglo predefinido llamado internamente “chain” – (**AddHash**).

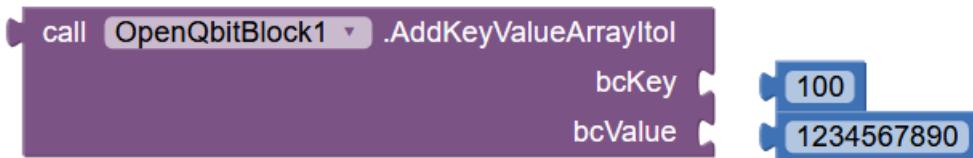


Parámetros de entrada: **block** <string>

Parámetros de salida: No aplica.

Descripción: Bloque para almacenar un arreglo temporal de hash o cadenas de caracteres en un arreglo predefinido llamado internamente “chain”.

Bloque para crear arreglos tipo key-value (**Integer-Integer**) – (**AddKeyValueArrayItol**)



Parámetros de entrada: **bcKey** <Integer>, **bcValue** <Integer>

Parámetros de salida: Retorna los valores introducidos en la entrada.

Descripción: Es un arreglo temporal tipo key-value, esta predefinido con nombre interno “**Itol_UTXO**” este es útil para procesar las transacciones UTXO.

Bloque para crear arreglos tipo key-value (**Integer-String**) – (**AddKeyValueArrayItoS**)

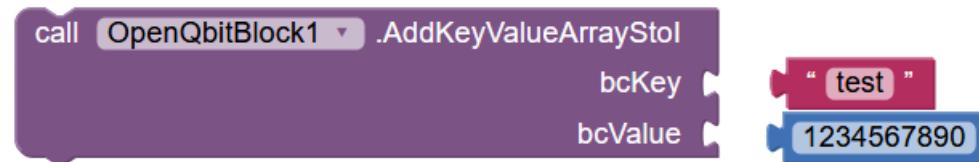


Parámetros de entrada: **bcKey** <Integer>, **bcValue** <String>

Parámetros de salida: Retorna los valores introducidos en la entrada.

Descripción: Es un arreglo temporal tipo key-value, esta predefinido con nombre interno “**ItoS_UTXO**” este es útil para procesar las transacciones UTXO.

Bloque para crear arreglos tipo key-value (String-Integer) – (AddKeyValueArrayItol)



Parámetros de entrada: **bcKey** <String>, **bcValue** <Integer>

Parámetros de salida: Retorna los valores introducidos en la entrada.

Descripción: Es un arreglo temporal tipo key-value, esta predefinido con nombre interno “Stol_UTXO” este es útil para procesar las transacciones UTXO.

Bloque para crear arreglos tipo key-value (String-String) – (AddKeyValueArrayStoS)

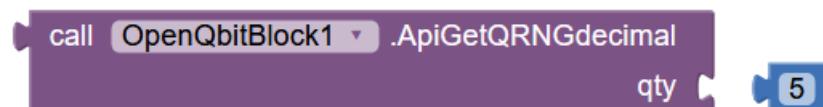


Parámetros de entrada: **bcKey** <String>, **bcValue** <String>

Parámetros de salida: Retorna los valores introducidos en la entrada.

Descripción: Es un arreglo temporal tipo key-value, esta predefinido con nombre interno “StoS_UTXO” este es útil para procesar las transacciones UTXO.

Bloque para generar números cuánticos aleatorios decimal – (ApiGetQRNGdecimal)



Parámetros de entrada: **qty** <Integer>

Parámetros de salida: Da la cantidad “qty” de números aleatorios cuánticos decimales introducidos en la entrada los números están dentro del rango de 0 y 1 en formato JSON.

Ejemplo:

```
qty = 5; output: {"result": [0.5843012986202495, 0.7746497687824652, 0.05951126805960929, 0.1986079055812694, 0.03689783439899279]}
```

Descripción: API de generador de números cuánticos aleatorios decimal (QRNG – Quantum Random Number Generator).

Bloque para generar números cuánticos aleatorios decimal – (**ApiGetQRNGdecimal**)



Parámetros de entrada: **qty** <Integer>, **min** <Integer>, **max** <max>

Parámetros de salida: Da la cantidad “qty” de números aleatorios cuánticos enteros introducidos en la entrada los números están dentro del rango de min y max en formato JSON.

Ejemplo:

`qty = 8, min = 1, max = 100; output: {"result": [3, 53, 11, 2, 66, 44, 9, 78]}`

Descripción: API de generador de números cuánticos aleatorios enteros (QRNG – Quantum Random Number Generator).

Bloque para sacar hash “SHA256” para cadenas de caracteres. (**ApplySha256**).



Parámetros de entrada: **input** <String>

Parámetros de salida: Calcula el hash “SHA256” de una cadena de caracteres.

Ejemplo:

Input = “Mini BlocklyChain es modular”

output: f41af7e61c3b02fdd5e5c612302b62a2dd52fcb38f9de97cb2af827e8804db8

Descripción: Función para sacar hash “SHA256”. Sha o SHA referirse a: Secure Hash Algorithm, un conjunto de funciones hash diseñado por la Agencia de Seguridad Nacional de los Estados Unidos. El SHA256 usa un algoritmo de 256 bites.

Bloque para limpiar el arreglo interno predefinido “chain” (**ClearBlockList**).

call OpenQbitBlock1 .ClearBlockList

Parámetros de entrada y salida: No aplica.

Descripción: Borrar todos los elementos que tiene el arreglo temporal interno “chain”.

Bloque para limpiar el arreglo interno predefinido (**Integer-Integer**) – (**ClearItol**).

call OpenQbitBlock1 .ClearItol

Parámetros de entrada y salida: No aplica.

Descripción: Borrar todos los elementos que tiene el arreglo temporal interno “Itol_UTXO”.

Bloque para limpiar el arreglo interno predefinido (**Integer-String**) – (**ClearItoS**).

call OpenQbitBlock1 .ClearItoS

Parámetros de entrada y salida: No aplica.

Descripción: Borrar todos los elementos que tiene el arreglo temporal interno “ItoS_UTXO”.

Bloque para limpiar el arreglo interno predefinido (**String-Integer**) – (**ClearStol**).

call OpenQbitBlock1 .ClearStol

Parámetros de entrada y salida: No aplica.

Descripción: Borrar todos los elementos que tiene el arreglo temporal interno “Stol_UTXO”.

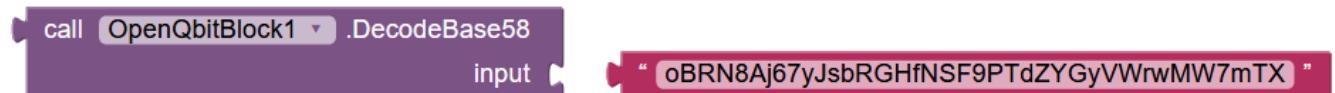
Bloque para limpiar el arreglo interno predefinido (**String-String**) – (**ClearStoS**).

call OpenQbitBlock1 .ClearStoS

Parámetros de entrada y salida: No aplica.

Descripción: Borrar todos los elementos que tiene el arreglo temporal interno “StoS_UTXO”.

Bloque para decodifica una cadena de caracteres (String) a Base10. (**DecodeBase58**)



Parámetros de entrada: **input<String>**

Parámetros de salida: Entrega la cadena original que se usó en el bloque (**EncodeBase58**).

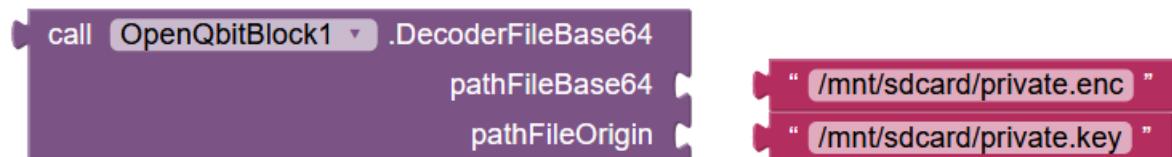
Ejemplo:

Input = oBRN8Aj67yJsbRGHfNSF9PTdZYGyVWrwMW7mTX

Output: "Mini BlocklyChain es modular".

Descripción: Convierte una cadena de Base58 a el texto original que se dio en el bloque (**EncodeBase58**)

Bloque para decodificar un archivo con algoritmo Base64 (**DecoderFileBase64**).



Parámetros de entrada: **pathFileBase64 <String>, pathFileOrigin <String>**

Parámetros de salida: Archivo origen que se introdujo en el bloque (**EncoderFileBase64**).

Descripción: Se convierte un archivo en formato Base64 al archivo original que fue introducido en el bloque (**EncoderFileBase64**).

Bloque para saber si está vacío el arreglo interno predefinido "chain". (**EmptyBlockList**)



Parámetros de entrada: No aplica.

Parámetros de salida: Regresa "True" si está vacío o regresa "Falso" si tiene datos.

Descripción: Bloque para preguntar si tiene elementos el arreglo predefinido interno temporal "chain".

Bloque para codificar una cadena de caracteres a Base58. (**EncodeBase58**).



Parámetros de entrada: **input<String>**

Parámetros de salida: Entrega la cadena original que se usó en el bloque (**EncodeBase58**).

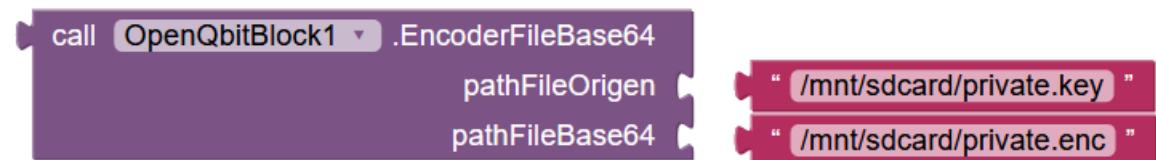
Ejemplo:

Input = “Mini BlocklyChain es modular”.

Output: oBRN8Aj67yJsbRGHfNSF9PTdZYGyVWrwMW7mTX

Descripción: Convierte una cadena de texto a una cadena en Base58. El algoritmo Base58 es un grupo de esquemas de codificación de binario a texto utilizados para representar enteros grandes como texto alfanumérico, introducido por Satoshi Nakamoto para su uso con Bitcoin.

Bloque para codificar un archivo con algoritmo Base64 (**EncoderFileBase64**).

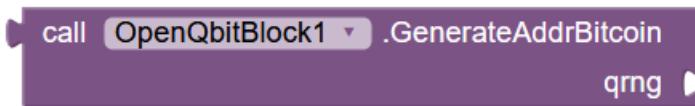


Parámetros de entrada: **pathFileOrigin <String>, pathFileBase64 <String>**

Parámetros de salida: Archivo codificado en Base64.

Descripción: Se convierte un archivo origen de cualquier formato a un archivo en formato Base64. Los nombres de los archivos pueden ser arbitrarios y escogidos por el usuario.

Bloque Generador de direcciones usuarios (**GenerateAddrBitcoin**).



Dependencia obligatoria: Bloque (**ApiGetQRNGinteger**),

Dependencias (opcionales): extensión **OpenQbitFileEncription**, extensión **OpenQbitFileDecryption** y extensión **OpenQbitSQLite**.

Parámetros de entrada: **qrng <dependencia obligatoria>**

Parámetros de salida: Dirección de transacciones de 34 caracteres alfanuméricos y dirección de uso en almacenamiento de keyStore.

Descripción: Bloque para crear una nueva dirección de transacciones genéricas Bitcoin para usuario y generador de clave privada (Firma digital para enviar transacciones) y clave pública (Dirección pública para hacer transacciones). Este generador de llaves, básicamente es la generadora de dirección para usarlas en una cartera digital o comúnmente llamada “Wallet”.

Este bloque es usado en conjunto con los bloques generador de números aleatorios cuánticos (QRNG – Quantum Random Number Generator) los dos parámetros de salida se deben insertar en el almacenamiento KeyStore.

KeyStore es una base de datos que almacena las claves privadas en formato hexadecimal:

Ejemplo de dirección hexadecimal que es almacenada en el KeyStore:

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

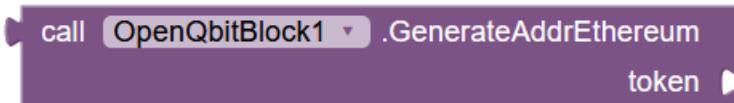
Esta base es usada únicamente por el usuario local y la información es cifrada, este proceso es a través del uso de la extensión **OpenQbitSQLite** y extensiones de cifrado de información **OpenQbitAESEncryption** y **OpenQbitAESDecryption**.

Para crear un KeyStore ver el Anexo de “Creación de KeyStore”. Las direcciones generadas utilizan el mismo algoritmo de direcciones Bitcoin, con identificador inicial de la dirección bitcoin que es “1”.

Las direcciones generadas por el bloque (**GenerateAddrBitcoin**) son de 34 caracteres alfanuméricos están compuestas por 33 caracteres alfanuméricos y 1 del identificador de Bitcoin de la siguiente forma:

12dRugNcdxK39288NjcDV4GX7rMsKCGn6k

Bloque Generador de direcciones usuarios (**GenerateAddrEthereum**).



Dependencia obligatoria: Obtener un token en: <https://accounts.blockcypher.com/signup>

Dependencias (opcionales): extensión **OpenQbitFileEncription**, extensión **OpenQbitFileDecryption** y extensión **OpenQbitSQLite**.

Parámetros de entrada: **token** <dependencia obligatoria - String>

Parámetros de salida: Dirección de transacciones de 40 caracteres alfanuméricicos no incluye el indicador de Ethereum inicial “0x” que nos daría los 42 caracteres de una dirección común. Retorna también la llave publica y la llave privada.

Ejemplo:

```
{
  "private": "227ac59f480131272003c2d723a7795ebd3580acaab62b5c537989e2ce4e08ef",
  "public":
    "04e2d55ebbebcc32a7384e096df559cc36b856c64a16e5b402e10585dc3ea055672aafa84d
    f8a859531570a650a8ab1e7a22949100efa1aa5f072c035551cac1ce",
  "address": "14e150399b0399f787b4d6fe30d8b251375f0d66"
}
```

Descripción: Bloque para crear una nueva dirección de transacciones para usuario y generador de clave privada (Firma digital para enviar transacciones) y clave pública (Dirección pública para hacer transacciones). Este generador de llaves es un API externo y se debe tener conexión Wifi o móvil, básicamente es la generadora de dirección para usarlas en una cartera digital o comúnmente llamada “Wallet”.

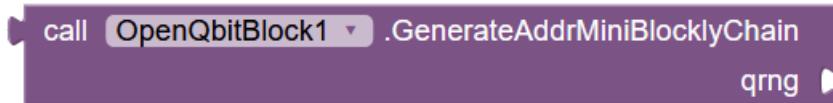
Se puede crear una KeyStore es una base de datos que almacena las claves privadas en formato hexadecimal, Ver Anexo de “Creación de KeyStore”.

Ejemplo de dirección hexadecimal que es almacenada en el KeyStore:

0x14e150399b0399f787b4d6fe30d8b251375f0d66

La llave privada es usada únicamente por el usuario local y la información es cifrada, este proceso es a través del uso de la extensión OpenQbitSQLite y extensiones de cifrado de información OpenQbitAESEncryption y OpenQbitAESDecryption.

Bloque Generador de direcciones usuarios (**GenerateAddrMiniBlocklyChain**).



Dependencia obligatoria: Bloque (**ApiGetQRNGinteger**),

Dependencias (opcionales): extensión **OpenQbitFileEncription**, extensión **OpenQbitFileDecryption** y extensión **OpenQbitSQLite**.

Parámetros de entrada: **qrng <dependencia obligatoria>**

Parámetros de salida: Dirección de transacciones de 36 caracteres alfanuméricos y dirección de uso en almacenamiento de keyStore. Revisar Bloque método (**PairKeysMBC**).

Descripción: Bloque para crear una nueva dirección de transacciones para usuario y generador de clave privada (Firma digital para enviar transacciones) y clave pública (Dirección pública para hacer transacciones). Este generador de llaves, básicamente es la generadora de dirección para usarlas en una cartera digital o comúnmente llamada “Wallet”.

Este bloque es usado en conjunto con los bloques generador de números aleatorios cuánticos (QRNG – Quantum Random Number Generator) los dos parámetros de salida se deben insertar en el almacenamiento KeyStore.

KeyStore es una base de datos que almacena las claves privadas en formato hexadecimal:

Ejemplo de dirección hexadecimal que es almacenada en el KeyStore:

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

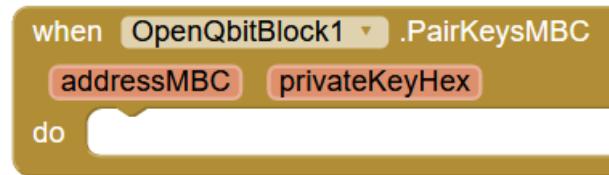
Esta base es usada únicamente por el usuario local y la información es cifrada, este proceso es a través del uso de la extensión OpenQbitSQLite y extensiones de cifrado de información OpenQbitAESEncryption y OpenQbitAESDecryption.

Para crear un KeyStore ver el Anexo de “Creación de KeyStore”. Las direcciones generadas utilizan el mismo algoritmo de direcciones Bitcoin, la única diferencia es que se cambia el identificador de la dirección bitcoin que es “1” o “3” por el identificador de Mini BlocklyChain “MBC”.

Las direcciones generadas por el bloque (**GenerateAddrMiniBlocklyChain**) son de 36 caracteres alfanuméricos están compuestas por 33 caracteres alfanuméricos y 3 del identificador de Mini BlocklyChain letras mayúsculas “MBC” quedando de la siguiente forma:

Mbc2dRugNcdxK39288NjcDV4GX7rMsKCGn6k

Bloque método (**PairKeysMBC**) es salida de bloque (**GenerateAddrMiniBlocklyChain**).



Parametros de entrada: No aplica.

Parametros de salida: **addressMBC <String>**, **privateKeyHex <String>**.

Descripción: Entrega dirección pública de usuario en formato Mini BlocklyChain y llave privada en formato hexadecimal.

ejemplo:

addressMBC: MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k

privateKeyHex:

024C8E05018319ARD4BB04E184C307BFF115976A05F974C7D945B5151E490ERD

Bloque Generador de direcciones usuarios (**GenerateKeyValuePair**).



Dependencia obligatoria: Bloque (**ApiGetQRNGinteger**),

Dependencias (opcionales): extensión **OpenQbitFileEncription**, extensión **OpenQbitFileDecryption** y extensión **OpenQbitSQLite**.

Parámetros de entrada: **qrng <dependencia obligatoria>**

Parámetros de salida: Proporciona la llave publica y llave primaria para usuario local.

Descripción: Genera llave publica y primaria usando algoritmo ECC (1) y con formato hexadecimal.

- (1) La Criptografía de Curva Elíptica (Elliptic curve cryptography, ECC) es una variante de la criptografía asimétrica o de clave pública basada en las matemáticas de las curvas elípticas.

Bloque para firmar transacciones (**GenerateSignature**).

```
call OpenQbitBlock1 .GenerateSignature
```

Dependencia obligatoria: Bloque (**GenerateKeyPair**), Bloque (**SenderLoadKeyPair**), Bloque (**RecipientLoadKeyPair**). Anteponer estos bloques antes de usarlo.

Parámetros de entrada: <dependencias de boques obligatorias>

Parámetros de salida: Ninguna salida.

Descripción: Genera firma digital desde el Sender (remitente) aplicado al activo enviado en la transacción al Recipient (destinatario), esta firma será confirmada cuando se esté procesando la transacción por algún nodo de la red, con esta firma se asegura el sistema Mini BlocklyChain de que el activo no se haya modificado en él envió y que cumpla con la relación de remitente-destinatario y no sea desviado o aplicado a otra dirección digital.

Bloque para obtener el balanceo total de activos de un usuario (**GetBalance**).

```
call OpenQbitBlock1 .GetBalance
```

```
fromAddrMBC
```

```
"MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k"
```

Dependencia obligatoria: Bloque (**ConektorTransactionTail**).

Parámetros de entrada: **fromTransTail** <Array String>, <dependencias de boques obligatorias>

Parámetros de salida: Verifica los gastos de activos de entrada y salida de cada transacción.

Descripción: Verifica el balance para aprobar si el usuario tiene activos para poder enviar o en su caso los activos que recibe se suman a su balance.

Bloque para obtener q22 de teléfono móvil. (**GetDeviceID**)

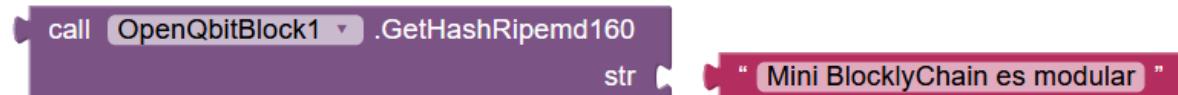
```
call OpenQbitBlock1 .GetDeviceID
```

Parámetros de entrada: **No aplica**

Parámetros de salida: Entrega el identificador interno del teléfono móvil.

Descripción: Proporciona el identificador IMEI del teléfono móvil único en cada dispositivo.

Bloque para sacar el hash RIPEMD-160 de una cadena de caracteres. (**Ripemd160**)



Parámetros de entrada: **str** <String>

Parámetros de salida: Calcula el hash “RIPEMD-160” de una cadena de caracteres.

Ejemplo:

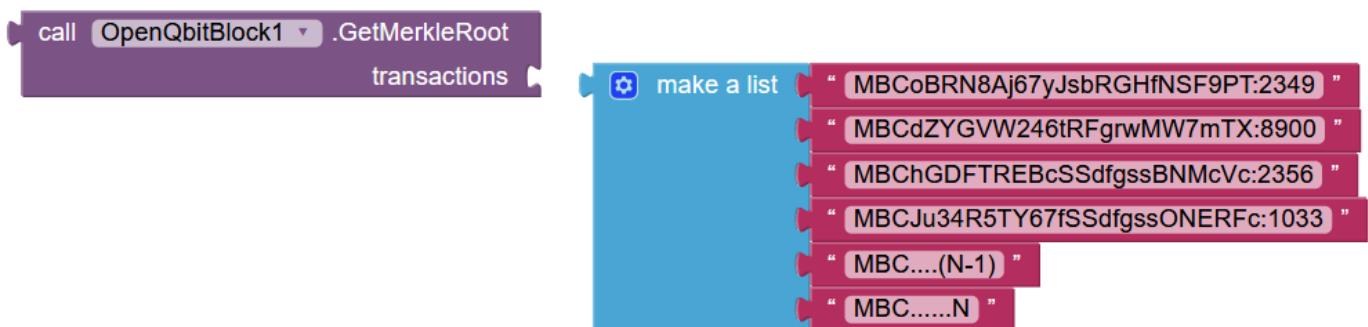
Input = “Mini BlocklyChain es modular”

output: ae29436e4b8ea8ed6143f3f92380dfa2f4f47336

Descripción: Obtener el hash “RIPEMD-160”. Este hash es utilizado en el proceso de generación de una dirección válida para Bitcoin y Mini BlocklyChain.

RIPEMD-160 (acrónimo de RACE Integrity Primitives Evaluation Message Digest, primitivas de integridad del resumen del mensaje) es un algoritmo del resumen del mensaje de 160 bits (y función criptográfica de hash).

Bloque para calcular el árbol de Merkler. (**GetMerkleRoot**)



Parámetros de entrada: **transactions** <Array String>

Parámetros de salida: Entrega un hash tipo “SHA256”.

Ejemplo:

Input = cola de transacciones, un arreglo de todas las transacciones que serán procesadas.

output: b4a44c42b6070825f763cd118d6ab49a8e80bbb7cdc0225064f8e042b94196bd

Descripción: Obtener el hash “SHA256” usando el algoritmo del árbol de Merkle.

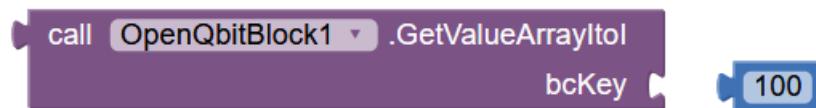
Un árbol hash de Merkle (Merkle Hash Tree) o árbol de merkle o árbol hash es una estructura de datos en árbol, binario o no, en el que cada nodo que no es una hoja está etiquetado con el hash de la concatenación de las etiquetas o valores (para nodos hoja) de sus nodos hijo. Son una generalización de las listas hash y las cadenas hash.

Permite que gran número de datos separados puedan ser ligados a un único valor de hash, el hash del nodo raíz del árbol. De esta forma proporciona un método de verificación segura y eficiente de los contenidos de grandes estructuras de datos. En sus aplicaciones prácticas normalmente el hash del nodo raíz va firmado para asegurar su integridad y que la verificación sea totalmente fiable. La demostración de que un nodo hoja es parte de un árbol hash dado requiere una cantidad de datos proporcional al logaritmo del número de nodos del árbol.

Actualmente el mayor uso de los árboles de Merkle es hacer seguros los bloques de datos recibidos de otros pares en las redes peer-to-peer, asegurar que estos son recibidos sin daños y sin ser alterados.

Es usada para verificar que una cola que tiene las transacciones a ser procesadas no haya sido modificada y poder asegurar su integridad para la dispersión en todos los nodos de la red de Mini BlocklyChain. Todos los nodos tienen que ejecutar este algoritmo para asegurar la integridad de cada transacción que será aplicada.

Bloque para obtener un valor del arreglo predefinido “Itol_UTXO” (**GetValueArrayItol**).

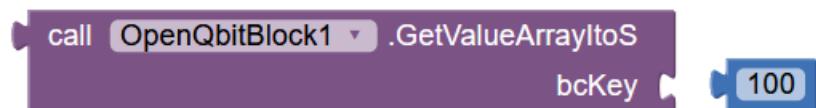


Parámetros de entrada: **bcKey** <Integer>

Parámetros de salida: Retorna el valor <Integer> almacenado en la etiqueta con el numero dado como entrada.

Descripción: Es una solicitud al arreglo temporal tipo key-value, que esta predefinido con nombre interno “Itol_UTXO” este es útil para procesar las transacciones UTXO.

Bloque para obtener un valor del arreglo predefinido “ItoS_UTXO” (**GetValueArrayItoS**).



Parámetros de entrada: **bcKey** <Integer>

Parámetros de salida: Retorna el valor <String> almacenado en la etiqueta con el numero dado como entrada.

Descripción: Es una solicitud al arreglo temporal tipo key-value, que esta predefinido con nombre interno “ItoS_UTXO” este es útil para procesar las transacciones UTXO.

Bloque para obtener un valor del arreglo predefinido “StoL_UTXO” (**GetValueArrayStoL**).



Parámetros de entrada: **bcKey** <String>

Parámetros de salida: Retorna el valor <Integer> almacenado en la etiqueta con el nombre dado como entrada.

Descripción: Es una solicitud al arreglo temporal tipo key-value, que esta predefinido con nombre interno “StoL_UTXO” este es útil para procesar las transacciones UTXO.

Bloque para obtener un valor del arreglo predefinido “StoS_UTXO” (**GetValueArrayStoS**).



Parámetros de entrada: **bcKey** <String>

Parámetros de salida: Retorna el valor <String> almacenado en la etiqueta con el nombre dado como entrada.

Descripción: Es una solicitud al arreglo temporal tipo key-value, que esta predefinido con nombre interno “StoS_UTXO” este es útil para procesar las transacciones UTXO.

Bloque validador de la cadena de bloques. (**IsChainValid**)

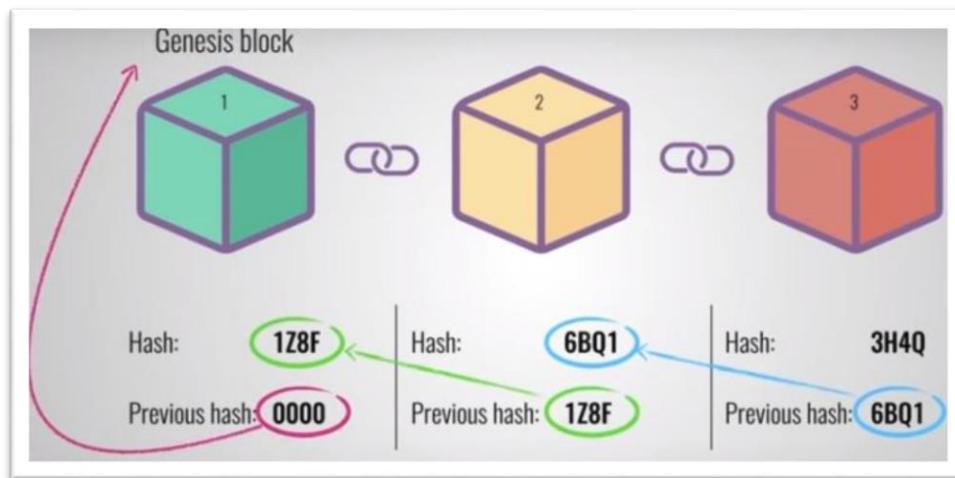


Dependencia obligatoria: Bloque (**GenerateKeyPairs**), Bloque (**SenderLoadKeyPair**), Bloque (**RecipientLoadKeyPair**), Bloque (**GenerateSignature**). Anteponer estos bloques antes de usarlo.

Parámetros de entrada: No aplica.

Parámetros de salida: Entrega “True” si la validación de la cadena de bloques es correcta o “False” en caso de que la validación falle.

Descripción: Nos proporciona la validación de los componentes que han sido insertados con anterioridad en el sistema de cadena de bloques, esta validación es la más importante de todo el sistema. Como trabaja la validación de cadena de bloques o como comúnmente se le conoce de forma genérica (BlockChain). Es la parte modular de todo sistema.



Como vemos en la imagen anterior cada bloque que se agrega en el sistema de almacenamiento está relacionada con el anterior bloque mediante los algoritmos de hash.

Por ejemplo, al crear un nuevo bloque para agregar, el hash que representa la nueva información esta obtenida sacando el hash de la información de nuevo bloque + el hash previo (previous hash). Con este tipo de enlace cualquier modificación mínima en el almacenamiento de cadenas de bloques será detectado lo que permite dar una seguridad en los datos muy alta y eficaz.

Enseguida se muestra un ejemplo de cómo se almacena una cadena de bloques en una base de datos SQLite, veremos como el hash previo está ligado al nuevo hash del último bloque (última cola de transacciones procesadas) que se insertó. Cada hash tanto en la columna “prevhash” y “newhash” representan la información de la cola de transacciones que en su momento fue procesada.

SQLite Expert Personal 5.3 (x64)

File View Database Object SQL Transaction Tools Help

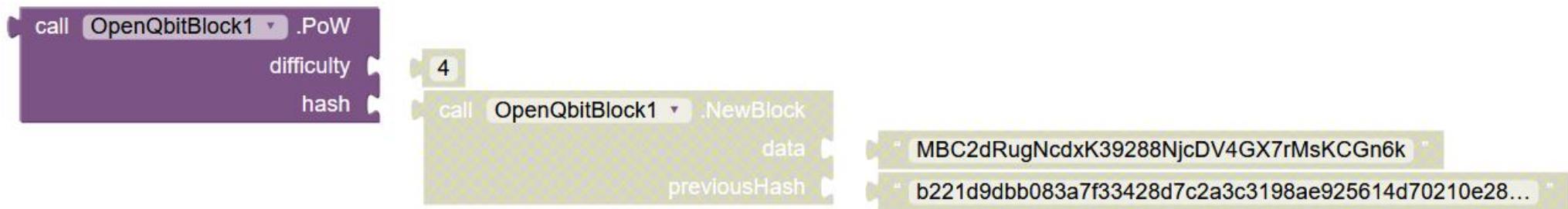
Database: miniblock Table: nblock File: C:\memo\thunkable\miniblock.db

miniblock

rowid	id	prevhash	newhash	opera	trans	balan
1	1	0767c864cef0334f27473902eb9868e7	bdc9065d20a4cd037bb1a7538486403e	2	0	0
2	2	bdc9065d20a4cd037bb1a7538486403e	6619f4809d73a267a4b9ac554bb4523a	1	5	5
3	3	6619f4809d73a267a4b9ac554bb4523a	4d186321c1a7f0f354b297e8914ab240	1	5	5

Previo Hash está relacionado al nuevo hash.

Bloque para realizar Prueba de Trabajo PoW y minar nuevos bloques. (**PoW**)



Dependencia obligatoria: Bloque (**NewBlock**). Anteponer este bloque antes de usarlo.

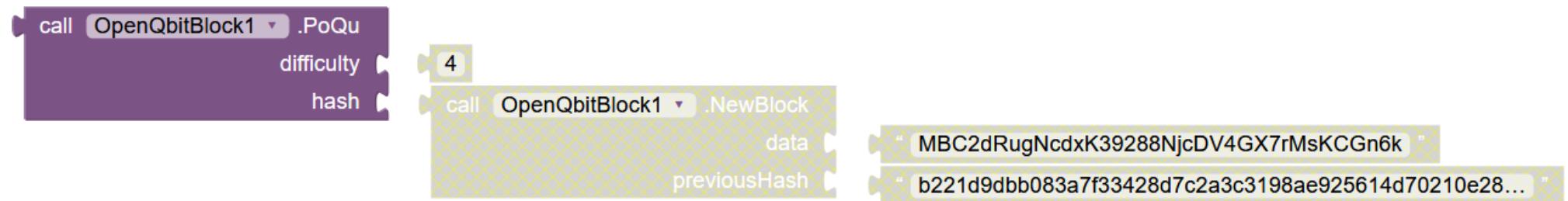
Parámetros de entrada: **difficulty** <Integer>, **hash** <Dependencia obligatoria>

Parámetros de salida: Da como resultado de salida un hash “SHA256” compuesto con el #Número de “ceros” dados en el parámetro de entrada difficulty (dificultad).

Descripción: Este bloque realiza una actividad llamada “minar” un nuevo bloque es lo que anteriormente hemos descrito como el proceso de PoW (Proof of Work) o Prueba de Trabajo, ver inciso ¿Qué es Prueba de Cuanto (PQu - Proof Quantum)?

El minar o ejecutar la PoW es un concepto donde los nodos se les da una tarea de resolver un acertijo matemático quien lo resuelve primero en el caso de Mini BlocklyChain obtiene la oportunidad de continuar con el proceso para poder ser elegido para ser el ganador de poder procesar la cola de transacciones que en ese momento está en espera de ser procesada.

Bloque para realizar Prueba de Trabajo PoW y minar nuevos bloques. (**PoQu**)



Dependencia obligatoria: Bloque (**NewBlock**). Anteponer este bloque antes de usarlo.

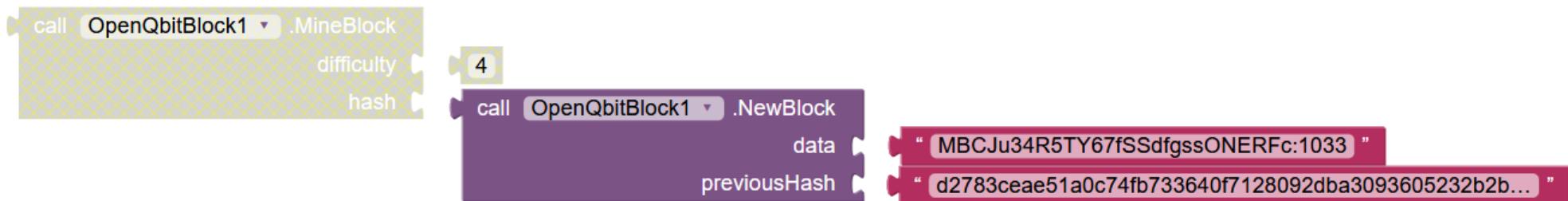
Parámetros de entrada: **difficulty** <Integer>, **hash** <Dependencia obligatoria>

Parámetros de salida: Da como resultado el número “Magic Number” y numero cuántico aleatorio que está dentro del rango de 0 y 1 tipo decimal de 16 dígitos, ejemplo (0. 5843012986202495) y un hash “SHA256” compuesto con el #Número de “ceros” dados en el parámetro de entrada difficulty (dificultad).

Descripción: Este bloque realiza el proceso de “minar” un nuevo bloque es lo que anteriormente hemos descrito como el proceso de PoW (Proof of Work) o Prueba de Trabajo, pero este proceso llama a la función de numero aleatorio cuántico generado después de haber calculado el número “nonce” adecuado para cumplir con el nivel de dificultad que puede ser en el rango de mínimo 1, máximo 5. Ver inciso ¿Qué es Prueba de Cuanto (PQu - Proof Quantum)?

El minar o ejecutar la PoW o PoQu es un concepto donde los nodos se les da una tarea de resolver un acertijo matemático el nodo quien lo resuelve primero en el caso de cualquier sistema de blockchain obtiene la oportunidad de continuar con el proceso para poder ser elegido para ser el ganador de poder procesar la cola de transacciones que en ese momento está en espera de ser procesada.

Bloque para la creación nuevos bloques (**NewBlock**).

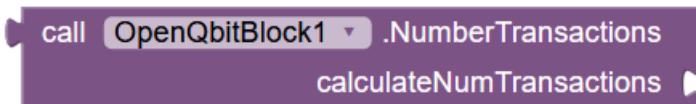


Parámetros de entrada: **data <String>**, **previousHash <String>**

Parámetros de salida: Entrega un hash calculado como: **SHA256 (data (parámetros de entrada) + previousHash (parámetro de entrada) + IMEI (parámetro interno) + MerkleRoot (parámetro interno))**.

Descripción: Este bloque realiza la creación de un nuevo bloque para ser procesado. Da como resultado de salida un hash “SHA256” compuesto de la cadena de los parámetros de entrada en el caso de los datos varían dependiendo de cada diseño de sistema y el previousHash está basado en el Hash de la anterior cadena de transacciones que ya fue procesada y se tiene almacenado en el sistema de cadena de bloque de Mini BlocklyChain, estos dos son parámetros variables, hay dos parámetros internos del sistema que son fijo y aseguran la integridad de la información y del sistema, estos dos parámetros internos son el ID único identificador del teléfono móvil y el hash del árbol de merklet de la cola de transacciones que está siendo procesada.

Bloque del total de transacciones locales del nodo (**NumberTransactions**)



Dependencia obligatoria: Bloque (**AddKeyValueToArrayStoS**). Anteponer este bloque antes de usarlo.

Parámetros de entrada: <Dependencia obligatoria>.

Parámetros de salida: Retorna el total de transacciones locales al nodo.

Descripción: Entrega el total de transacciones que serán aplicadas únicamente a las cuentas locales del nodo.

Bloque para leer dirección del destinario en archivo binario. (**RecipientLoadKeyValuePair**)

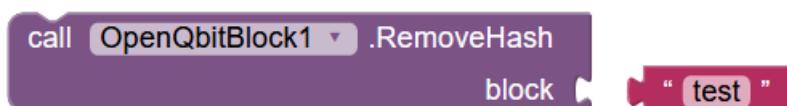


Parámetros de entrada: **No aplica**.

Parámetros de salida: Retorna llave privada y llave pública en un formato Base64.

Descripción: Obtiene la dirección privada y pública del destinatario desde un archivo binario como parámetro de entrada.

Bloque para borrar elemento en arreglo temporal interno “chain” (**RemoveHash**).

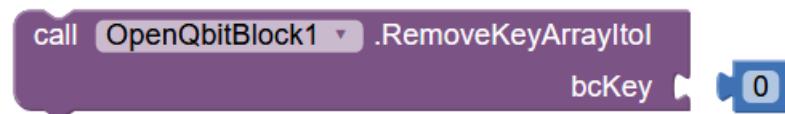


Parámetros de entrada: **block <String>**.

Parámetros de salida: No aplica.

Descripción: Obtiene la dirección privada y pública del destinatario desde un archivo binario como parámetro de entrada.

Bloque para borrar elemento en arreglo temporal interno “Itol_UTXO” (**RemoveKeyArrayItol**)

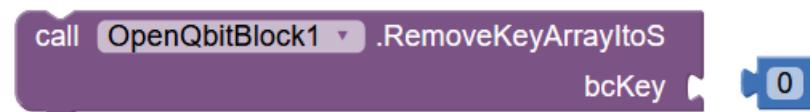


Parámetros de entrada: **bcKey <Integer>**.

Parámetros de salida: No aplica, tipo de elemento borrar <Integer>

Descripción: Se borrar el elemento asociado a la etiqueta numérica del arreglo temporal interno “Itol_UTXO”.

Bloque para borrar elemento en arreglo temporal interno “ItoS_UTXO” (**RemoveKeyArrayItoS**).

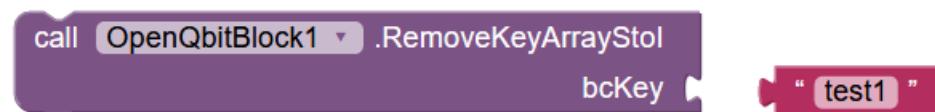


Parámetros de entrada: **bcKey <Integer>**.

Parámetros de salida: No aplica, tipo de elemento borrar <String>

Descripción: Se borrar el elemento asociado a la etiqueta numérica del arreglo temporal interno “ItoS_UTXO”.

Bloque para borrar elemento en arreglo temporal interno “Stol_UTXO” (**RemoveKeyArrayStol**).

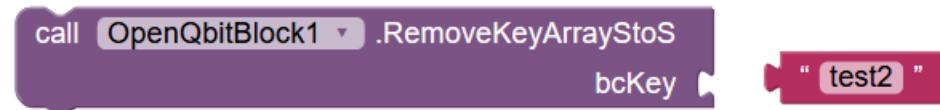


Parámetros de entrada: **bcKey <String>**.

Parámetros de salida: No aplica, tipo de elemento borrar <Integer>

Descripción: Se borrar el elemento asociado a la etiqueta numérica del arreglo temporal interno “Stol_UTXO”.

Bloque para borrar elemento en arreglo temporal interno “StoS_UTXO” (**RemoveKeyArrayStoS**).

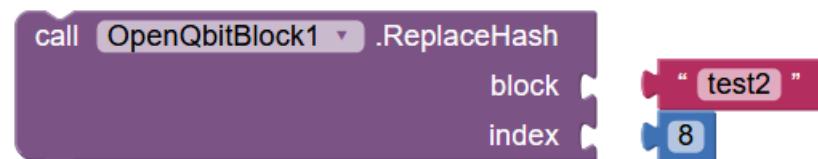


Parámetros de entrada: **bcKey** <String>.

Parámetros de salida: No aplica, tipo de elemento borrar <String>

Descripción: Borrar elemento de la etiqueta del arreglo temporal interno “StoS_UTXO”.

Bloque para remplazar un valor del arreglo temporal interno “chain” (**ReplaceHash**).

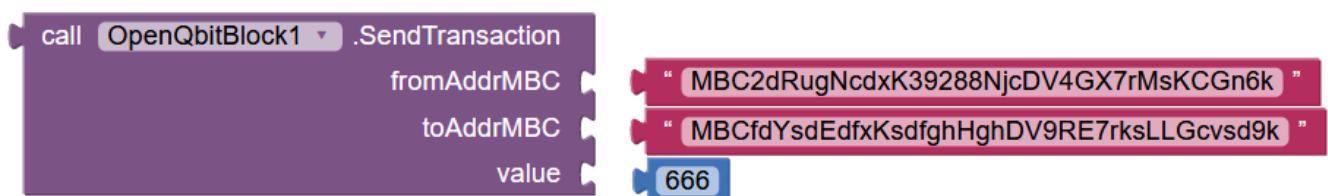


Parámetros de entrada: **block** <String>, **index** <Integer>

Parámetros de salida: No aplica.

Descripción: Se sustituye el elemento asociado al índice “index” con el valor de la etiqueta “block” en el arreglo interno temporal “chain”.

Bloque para iniciar (enviar) nueva transacción (**SendTrasaction**).



Parámetros de entrada: **fromAddrMBC** <String>, **toAddrMBC** <String>, **value** <Integer>

Parámetros de salida: Retorna valor “True” si fue enviada la transacción exitosamente o “False” si no fue exitoso él envió.

Descripción: Bloque que transforma la dirección de remitente y destinatario para un formato binario y se anexa a la cola de transacciones para ser procesada.

Bloque para leer dirección del remitente en archivo binario. (**SenderLoadKeyPair**)

call OpenQbitBlock1 .SenderLoadKeyPair

Parámetros de entrada: **No aplica.**

Parámetros de salida: Retorna llave privada y llave pública en un formato Base64.

Descripción: Obtiene la dirección privada y pública del remitente desde un archivo binario como parámetro de entrada.

Bloque para obtener número de elemento del arreglo temporal “chain” (**SizeBlockList**).

call OpenQbitBlock1 .SizeBlockList

Parámetros de entrada: **No aplica.**

Parámetros de salida: Retorna el número de elemento del arreglo interno “chain”.

Descripción: Obtiene el número de elemento del arreglo interno “chain”.

Bloque para obtener número de elemento del arreglo temporal “Itol_UTXO” (**Sizeltol**).

call OpenQbitBlock1 .Sizeltol

Parámetros de entrada: **No aplica.**

Parámetros de salida: Retorna el número de elemento del arreglo interno “Itol_UTXO”

Descripción: Obtiene el número de elemento del arreglo interno “Itol_UTXO”

Bloque para obtener número de elemento del arreglo temporal “ItoS_UTXO” (**SizeltoS**).

call OpenQbitBlock1 .SizeltoS

Parámetros de entrada: **No aplica.**

Parámetros de salida: Retorna el número de elemento del arreglo interno “ItoS_UTXO”.

Descripción: Obtiene el número de elemento del arreglo interno “ItoS_UTXO”.

Bloque para obtener número de elemento del arreglo temporal “StoL_UTXO” (**SizeStoL**).

call OpenQbitBlock1 .SizeStoL

Parámetros de entrada: **No aplica.**

Parámetros de salida: Retorna el número de elemento del arreglo interno “StoL_UTXO”.

Descripción: Obtiene el número de elemento del arreglo interno “StoL_UTXO”.

Bloque para obtener número de elemento del arreglo temporal “StoS_UTXO” (**SizeStoS**).

call OpenQbitBlock1 .SizeStoS

Parámetros de entrada: **No aplica.**

Parámetros de salida: Retorna el número de elemento del arreglo interno “StoS_UTXO”.

Descripción: Obtiene el número de elemento del arreglo interno “StoS_UTXO”.

Bloque creación de transacción con valores adicionales (**Transaction**).

The screenshot shows the Blockly interface with the Transaction block selected. The block has four input ports: activeValue (with value 666), fromAddrMBC (with value "MBC2dRugNcdxK39288NjcDV4GX7rMsKCGn6k"), toAddrMBC (with value "MBCfdYsdEdfxKsdfghHghDV9RE7rksLLGcvsd9k"), and inputsValue (with value "make a list auth: 3456 date: 04.06.2020").

Parámetros de entrada: **activeValue <Integer>, fromAddrMBC <String>, toAddrMBC <String>, inputsValue <Array String>**.

Parámetros de salida: Nos entrega, las direcciones en formato binario y el valor inpushValue convertido en una cadena <String>, y nos entrega el hash “SHA256” del valor ActiveValue.

Descripción: Prepara una nueva transacción para ser procesada por los nodos.

Bloque de validación de dirección Mini BlocklyChain. (**ValidateAddMiniBlocklyChain**)



Parámetros de entrada: Direcciones para usuario con formato Mini BlocklyChain.

Parámetros de salida: Retorna “True” si la dirección tiene el formato correcto o “Falso” si es no valida la dirección.

Descripción: Valida si es correcta la dirección Mini BlocklyChain introducida, este bloque aplica un algoritmo para verificar si a dirección fue creada usando el mecanismo de creación de dirección para ser usadas en el sistema Mini BlocklyChain.

Bloque de validación de dirección Bitcoin. (**ValidateAddBitcoin**)



Parámetros de entrada: **addr <String>**, Direcciones para usuario con formato Bitcoin (acepta direcciones Bitcoin con identificador inicial “1”, direcciones con identificador “3” no es aplicable).

Parámetros de salida: Retorna “True” si la dirección tiene el formato correcto o “Falso” si es no valida la dirección.

Descripción: Valida si es correcta la dirección Bitcoin introducida, este bloque aplica un algoritmo para verificar si a dirección fue creada usando el mecanismo de creación de dirección para ser usadas en el sistema Bitcoin.

Bloque de verificación de firma digital de la transacción en curso. (**VerifySignature**).



Parámetros de entrada: **No aplica**.

Parámetros de salida: Retorna “True” si la verificación es válida o “Falso” si es no valida la verificación.

Descripción: Obtiene la verificación de la firma digital que el remitente debió a ver realizado en el proceso de envío de la transacción que desea realizar. En este proceso de verificación se realiza para la comprobación del valor origen enviado no ha sido alterado en el canal donde se envió la transacción, así como se verifica el destinatario al que se debe aplicar la transacción

20. Uso de bloques para base de datos SQLite (versión MiniSQLite).

En esta sección veremos cómo usar los bloques para realizar dos principales operaciones que nos interesa para la funcionalidad del sistema Mini BlocklyChain que es “INSERTA” datos a la cadena de bloques y consultarlos.

NOTA: Las transacciones en la base de datos SQLite son diferentes a las transacciones enviadas por los nodos para ser aplicadas en el sistema Mini Blocklychain.

Las transacciones en la base de datos están basadas en un modelo tipo CRUD (Create, Read, Update y Delete) – (Crear, Leer, Actualizar y Borrar) y son los procesos que se pueden ejecutar con datos externos o internos únicamente en la base de datos SQLite. En los sistemas blockchain se utilizan principalmente los procesos de Crear y Leer, por seguridad están descartados los procesos de actualizar o borrar y más en donde se almacena la cadena de bloques que da la seguridad integral del sistema.

Por otra parte, las transacciones enviadas por los nodos se refieren a todos los procesos que involucra realizar la acción del envío de algún tipo de activo entre los integrantes (nodos) del sistema Mini BlocklyChain, este tipo de transacciones incluyen diversos procesos para su aplicación, estos pueden ser desde la creación de una dirección digital, una firma digital, una validación de firma, un proceso dentro de la base de datos SQLite, entre otros procesos.

Empezaremos con la definición y uso de los bloques de la base de datos SQLite versión MiniSQLite esta versión está integrada únicamente por 8 bloques. Se tiene una versión completa para manipular los datos de la base de datos SQLite, sin embargo, para fines prácticos del sistema Mini BlocklyChain con la versión MiniSQLite es suficiente.

En caso de desear revisar todos los componentes su uso y descripción ver Anexo “Bloques extendidos para base de datos SQLite”.

Bloques versión MiniSQLite:

Bloque para iniciar algún tipo de transacción en la base de datos SQLite (**BeginTransaction**)

call OpenQbitSQLite1 .BeginTransaction

Dependencia(s) obligatoria(s): Bloque (**ImportDatabase**), Bloque (**OpenDatabase**).

Parámetros de entrada: **Usar antes <Dependencia(s) obligatoria(s)>**

Parámetros de salida: No aplica, inicia una transacción en una base de datos SQLite.

Descripción: Bloque que inicia un proceso en la base de datos SQLite, debe antes haber ejecutado el bloque de importar base de datos (**ImportDatabase**) y el bloque de abrir la base de datos (**OpenDatabase**).

Bloque para hacer Commit en SQLite. (**CommitTrasaction**)

call **OpenQbitQSQLite1** .**CommitTransaction**

Dependencia(s) obligatoria(s): Bloque (**BeginTransaction**), Bloque (**ImportDatabase**), Bloque (**OpenDatabase**).

Parámetros de entrada: **Usar antes <Dependencia(s) obligatoria(s)>**

Parámetros de salida: No aplica, realiza un **commit** de una transacción en una base de datos SQLite.

Descripción: Bloque que inicia un proceso de **commit** en la base de datos SQLite, debe antes haber ejecutado el bloque de importar base de datos (**ImportDatabase**) y el bloque de abrir la base de datos (**OpenDatabase**).

Bloque para cerrar la base de datos SQLite que se importó o exportó (**CloseDatabase**)

call **OpenQbitQSQLite1** .**CloseDatabase**

Dependencia(s) obligatoria(s): Bloque (**ImportDatabase**), Bloque (**OpenDatabase**).

Parámetros de entrada: **Usar antes <Dependencia(s) obligatoria(s)>**

Parámetros de salida: No aplica, cierra una base de datos SQLite.

Descripción: Bloque que cierra la base de datos SQLite, debe antes haber ejecutado el bloque de importar base de datos (**ImportDatabase**) y el bloque de abrir la base de datos (**OpenDatabase**).

Bloque para exportar una base de datos SQLite (**ExportDatabase**).

call **OpenQbitQSQLite1** .**ExportDatabase**
 fileName **" mbcExport.sqlite "**

Dependencia(s) obligatoria(s): Bloque (**ImportDatabase**), Bloque (**OpenDatabase**).

Parámetros de entrada: **fileName <String>** introducir una ruta (path) en donde se encuentra una base de datos ya creada con formato SQLite.

Usar antes <Dependencia(s) obligatoria(s)>

Parámetros de salida: Exportar en una base de datos SQLite.

Descripción: Bloque que exporta una base de datos SQLite, debe antes haber ejecutado el bloque de importar base de datos (**ImportDatabase**) y el bloque de abrir la base de datos (**OpenDatabase**).

Bloque para importa base de datos SQLite. (**ImportDatabase**)

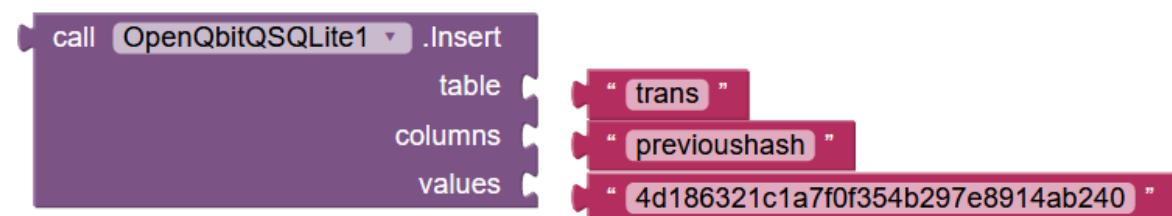


Parámetros de entrada: **fileName <String>** introducir una ruta (path) en donde se encuentra una base de datos ya creada con formato SQLite.

Parámetros de salida: Inicia una base de datos SQLite para ejecutar transacciones.

Descripción: Bloque que inicia un proceso en la base de datos SQLite, debe antes haber ejecutado el bloque de importar base de datos (**ImportDatabase**) y el bloque de abrir la base de datos (**OpenDatabase**).

Bloque para insertar datos en base de datos SQLite. (**Insert**)



Dependencia(s) obligatoria(s): Bloque (**ImportDatabase**), Bloque (**OpenDatabase**).

Parámetros de entrada: **table <String>**, **columns <String>**, **values <String>**, Usar antes <Dependencia(s) obligatoria(s)>

Parámetros de salida: Inserta una transacción en una base de datos SQLite.

Descripción: Bloque que inserta datos en la base de datos SQLite, debe antes haber ejecutado el bloque de importar base de datos (**ImportDatabase**) y el bloque de abrir la base de datos (**OpenDatabase**).

Bloque para abrir base de datos SQLite (**OpenDatabase**)



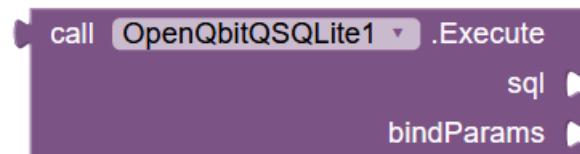
Dependencia(s) obligatoria(s): Bloque (**ImportDatabase**).

Parámetros de entrada: **Usar antes <Dependencia(s) obligatoria(s)>**

Parámetros de salida: No aplica, inicia o abrir una base de datos SQLite para realizar transacciones.

Descripción: Bloque que inicia una base de datos SQLite, debe antes haber.

Bloque para la consulta de datos en SQLite (**Execute**).



Dependencia(s) obligatoria(s): Bloque (**ImportDatabase**), Bloque (**OpenDatabase**).

Parámetros de entrada: **sql <String>, bindParams <String>, Usar antes <Dependencia(s) obligatoria(s)>**

Parámetros de salida: Se ejecuta la sentencia de SQL para crear una transacción en una base de datos SQLite.

Descripción: Bloque que ejecutar sentencias SQL en la base de datos SQLite, debe antes haber ejecutado el bloque de importar base de datos (**ImportDatabase**) y el bloque de abrir la base de datos (**OpenDatabase**).

21. Definición y uso de bloques de seguridad.

En esta sesión revisaremos el uso de los bloques que nos proporcionan niveles de seguridad para guardar, validar y transferir las transacciones de los nodos de la red de Mini BlocklyChain.

Los bloques de seguridad están basados en la siguiente extensión:

- I. Extensión OpenQbitAESEncryption.
- II. Extensión OpenQbitAESDecryption.
- III. Extensión OpenQbitAEToString.
- IV. Extensión OpenQbitEncDecData.
- V. Extensión OpenQbitFileHash.
- VI. Extensión OpenQbitRSA.
- VII. Extensión OpenQbitSSHClient (obligatoria)
- VIII. Extensión OpenQbitStringHash.
- IX. Extension OpenSQLite.

A excepción de la extensión OpenQbitSSHClient que es obligatoria, las anteriores extensiones son opcionales su uso en la creación de una red pública o privada de Mini BlocklyChain.

Sin embargo, para tener un sistema con seguridad en sus transacciones dependiendo de cada caso de negocio deberán ser aplicadas a discreción el uso de las extensiones que son opcionales.

Por ejemplo, en el caso del uso de hash podemos usar una serie de opciones de algoritmos como son: MD5, SHA1, SHA128, SHA256, SHA512 tanto para una cadena de caracteres como pudiendo ser aplicando también a cualquier tipo de archivo dependiendo el flujo de información de cada sistema creado con Mini BlocklyChain.

Extensión OpenQbitAESEncryption.

Bloque para cifrar archivo con seguridad AES. (**AESEncryption**).



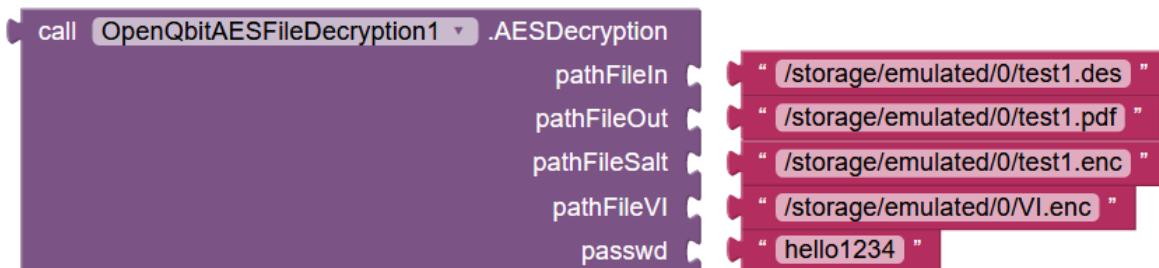
Parámetros de entrada: **pathFileIn <String>**, **pathFileOut <String>**, **pathFile <String>**, **pathFileVI <String>** y **passwd <String>**. Los nombres de los archivos son arbitrarios y a elección de cada diseño de sistema.

Parámetros de salida: Archivo cifrado con AES introducido en el parámetro de entrada **pathFileIn**.

Descripción: Bloque que nos entrega tres archivos de salida uno de estos es el archivo origen ya cifrado por el algoritmo AES con una llave de 256 bites, los otros dos archivos son usados para el control de la extensión para descifrar y recuperar el archivo original.

Extensión OpenQbitAESDecryption.

Bloque para descifrar archivo AES. (**AESDecryption**).



Parámetros de entrada: **pathFileIn <String>**, **pathFileOut <String>**, **pathFile <String>**, **pathFileVI <String>** y **passwd <String>**. Los nombres de los archivos son los mismos que se obtuvieron como resultado en el bloque (**AESEncryption**).

Parámetros de salida: Archivo original descifrado con AES introducido en el parámetro de entrada **pathFileIn**, en este caso para descifrar el archivo este es introducido en **pathFileIn** el archivo cifrado y en el **pathFileOut** nos entregara el archivo original (descifrado).

Descripción: Bloque que nos entrega un archivo en el parámetro **pathFileOut** que será la salida descifrada por el algoritmo AES con una llave de 256 bites.

Extensión OpenQbitAESToString.

Esta extensión es de un solo uso por sesión de dispositivo, es decir únicamente funciona cuando el dispositivo no es reiniciado (teléfono móvil), ya que el valor de cifrado VI es generado de forma temporal.

NOTA: En caso de reiniciar el dispositivo no se podrá recuperar la cadena cifrada.

Bloque para cifrado temporal de cadena de caracteres (**DecrypSecretText**)

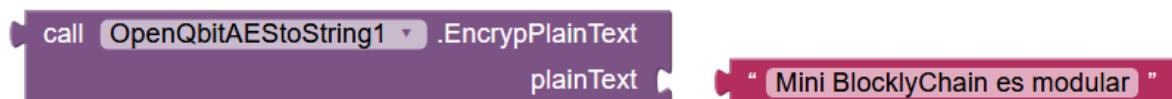


Parámetros de entrada: **secretText <String>**

Parámetros de salida: Cadena de caracteres original descifrado con AES con llave 256 bites.

Descripción: Bloque que nos entrega una cadena de caracteres, es el parámetro de entrada que se introdujo en el bloque (**EncrypPlainText**).

Bloque para descifrar una cadena de caracteres (**EncrypPlainText**)



Parámetros de entrada: **plainText <String>**

Parámetros de salida: Cadena de caracteres cifrado con AES usando llave de 256 bites.

Descripción: Bloque que nos entrega una cadena de caracteres alfanumérica cifrada con AES usando una llave digital de 256 bites.

Extensión OpenQbitEncDecData.

Bloque de cifrado especializado para bases de datos genérico. (**EncryptionData**)



Parámetros de entrada: **plainText <String>**

Parámetros de salida: Cadena de caracteres cifrado con AES usando llave de 256 bites.

Descripción: Bloque que nos entrega una cadena de caracteres alfanumérica cifrada con AES usando una llave digital de 256 bites.

Bloque de cifrado especializado para bases de datos genérico. (**DescriptData**)



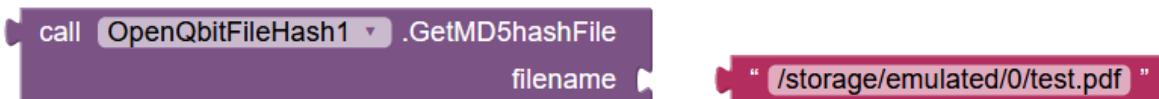
Parámetros de entrada: **plainText <String>**

Parámetros de salida: Cadena de caracteres cifrado con AES usando llave de 256 bites.

Descripción: Bloque que nos entrega una cadena de caracteres alfanumérica cifrada con AES usando una llave digital de 256 bites.

Extensión OpenQbitFileHash.

Bloque para generar MD5 de un archivo (**GetMD5hashFile**)

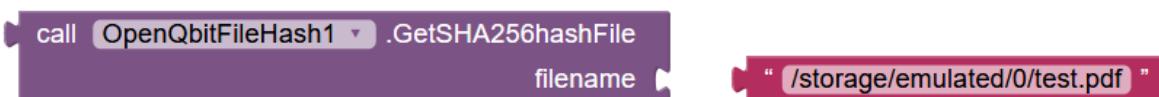


Parámetros de entrada: **filename <string>**

Parámetros de salida: Entrega el hash MD5 de archivo.

Descripción: Bloque para crear el hash MD5 del archivo dado en el parámetro de entrada.

Bloque para generar SHA256 de un archivo (**GetSHA256hashFile**)

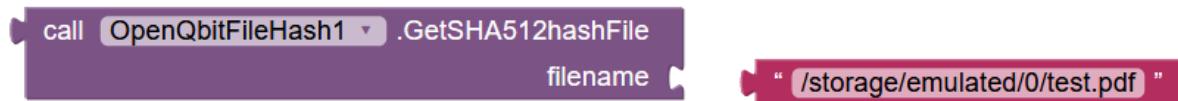


Parámetros de entrada: **filename <string>**

Parámetros de salida: Entrega el hash SHA256 de archivo.

Descripción: Bloque para crear el hash SHA256 del archivo dado en el parámetro de entrada.

Bloque para generar SHA512 de un archivo (**GetSHA512hashFile**)

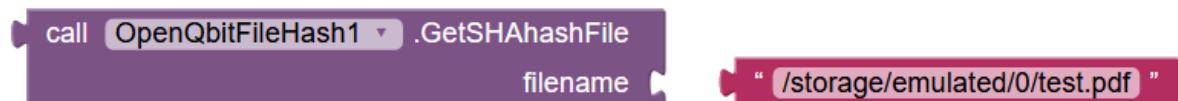


Parámetros de entrada: **filename** <string>

Parámetros de salida: Entrega el hash SHA256 de archivo.

Descripción: Bloque para crear el hash SHA256 del archivo dado en el parámetro de entrada.

Bloque para generar SHA1 de un archivo (**GetSHA1hashFile**)



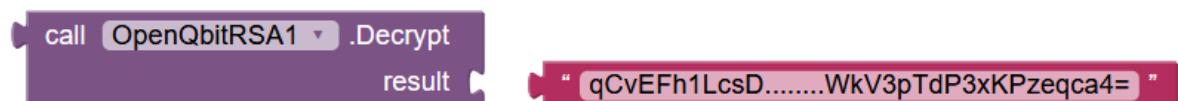
Parámetros de entrada: **filename** <string>

Parámetros de salida: Entrega el hash SHA1 de archivo.

Descripción: Bloque para crear el hash SHA1 del dado en el parámetro de entrada.

Extensión OpenQbitRSA.

Bloque para descifrar cadena de caracteres con RSA (**Decrypt**)



Dependencia(s) obligatoria(s): Bloque (**Encrypt**), Bloque (**OpenFromDiskPrivateKey**), Bloque (**OpenFromDiskPublicKey**).

Parámetros de entrada: **result** <String>

Parámetros de salida: Cadena de caracteres descifrada con RSA.

Descripción: Bloque que nos entrega una cadena de caracteres alfanumérica descifrada usando llave del tamaño que se usó en el bloque (**GenKeyPair**).

Bloque para cifrar cadena de caracteres con RSA (**Encrypt**)



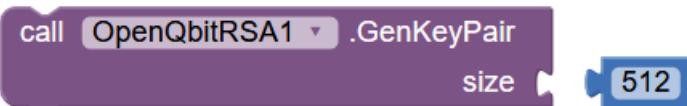
Dependencia(s) obligatoria(s): Bloque (**GenKeyValuePair**), Bloque (**SaveFromDiskPrivateKey**), Bloque (**SaveFromDiskPublicKey**).

Parámetros de entrada: **plain <String>**

Parámetros de salida: Cadena de caracteres cifrada con RSA.

Descripción: Bloque que nos entrega una cadena de caracteres alfanumérica descifrada usando llave del tamaño que se usó en el bloque (**GenKeyValuePair**).

Bloque para descifrar cadena de caracteres con RSA (**Decrypt**)

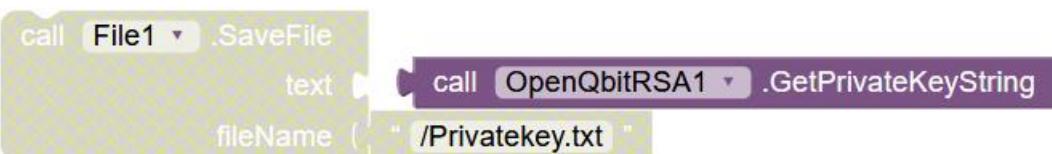


Parámetros de entrada: **size <Integer>**

Parámetros de salida: No aplica.

Descripción: Bloque para generar llave privada y llave publica basadas en el tamaño elegido.

Bloque para obtener la llave privada (**GetPrivateKeyString**)



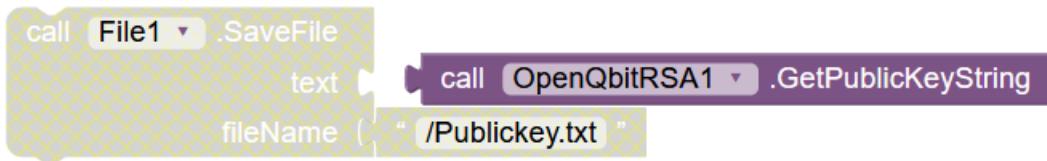
Dependencia(s) obligatoria(s): Bloque (**GenKeyValuePair**), Bloque (**GenKeyValuePair**), Bloque (**File**)

Parámetros de entrada: **No aplica.**

Parámetros de salida: Archivo con cadena de caracteres cifrada con RSA (llave privada)

Descripción: Bloque que nos entrega una cadena de caracteres alfanumérica que representa la llave privada cifrada usando llave del tamaño que se usó en el bloque (**GenKeyValuePair**).

Bloque para obtener la llave privada (**GetPublicKeyString**)



Dependencia(s) obligatoria(s): Bloque (**GenKeyPair**), Bloque (**GenKeyPair**), Bloque (**File**)

Parámetros de entrada: **No aplica.**

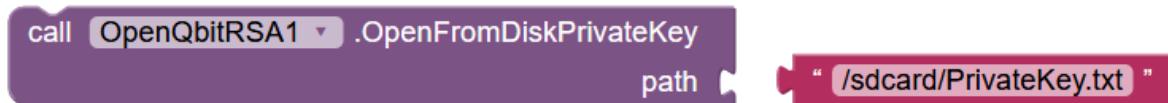
Parámetros de salida: Archivo con cadena de caracteres cifrada con RSA (llave publica)

Descripción: Bloque que nos entrega una cadena de caracteres alfanumérica que representa la llave publica cifrada usando llave del tamaño que se usó en el bloque (**GenKeyPair**).

NOTA: En los bloques anteriores (**GetPrivateKeyString**) y (**GetPublicKeyString**) en las dependencias usaremos el bloque genérico (**File**) de la aplicación App Inventor de la sesión de la paleta “Storage”.

Esta forma de almacenar la llave privada y pública por medio del bloque (**file**) nos puede ayudar a tener una mejor manipulación de la información.

Bloque para leer llave privada de un archivo (**OpenFromDiskPrivateKey**).



Dependencia(s) obligatoria(s): Bloque (**SaveFromDiskPrivateKey**) o Bloque (**GetPrivateKeyString**).

Parámetros de entrada: **path <String>**

Parámetros de salida: Carga en sistema cadena de caracteres cifrada de llave privada RSA.

Descripción: Bloque que nos entrega una cadena de caracteres alfanumérica cifrada de la llave privada almacenada en la ruta proporcionada del archivo.

Bloque para leer llave publica de un archivo (**OpenFromDiskPublicKey**)



Dependencia(s) obligatoria(s): Bloque (**SaveFromDiskPublicKey**) o Bloque (**GetPublicKeyString**).

Parámetros de entrada: **path <String>**

Parámetros de salida: Carga en sistema cadena de caracteres cifrada de llave publica RSA.

Descripción: Bloque que nos entrega una cadena de caracteres alfanumérica cifrada de la llave publica almacenada en la ruta proporcionada del archivo.

Bloque para guardar la llave privada en un archivo (**SaveToDiskPrivateKey**).



Dependencia(s) obligatoria(s): Bloque (**GenKeyValuePair**).

Parámetros de entrada: **plain <String>**

Parámetros de salida: Archivo con cadena de caracteres cifrada con RSA. (llave privada)

Descripción: Bloque que nos entrega un archivo con una cadena de caracteres alfanumérica cifrada usando llave privada del tamaño que se usó en el bloque (**GenKeyValuePair**).

Bloque para guardar la llave privada en un archivo (**SaveToDiskPublicKey**).



Dependencia(s) obligatoria(s): Bloque (**GenKeyValuePair**).

Parámetros de entrada: **plain <String>**

Parámetros de salida: Archivo con cadena de caracteres cifrada con RSA. (llave publica)

Descripción: Bloque que nos entrega un archivo con una cadena de caracteres alfanumérica cifrada usando llave publica del tamaño que se usó en el bloque (**GenKeyValuePair**).

Extension OpenQbitSSHClient.

Bloque Conector SSH Cliente (**ConnectorMiniBlocklyChain**)

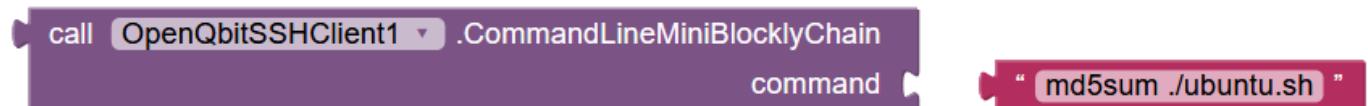


Parámetros de entrada: **username** <string>, **password** <string>, **host** <string>, **port**<integer>

Parámetros de salida: Si es exitosa la conexión con el servidor ssh de la terminal Termux nos entrega un mensaje; "**Connect SSH**", en caso de no ser exitosa nos entrega un mensaje **NULL**.

Descripción: Bloque de comunicación para conectar Mini BlocklyChain a la terminal Termux, vía protocolo de comunicación SSH (Secure Shell).

Bloque para Ejecutar comandos en terminal Termux Linux (**CommandLineMiniBlocklyChain**).



Parámetros de entrada: **command** <string>

Parámetros de salida: Dato variable, depende del comando ejecutado o programa.

Descripción: Bloque de ejecución de comandos en la terminal Termux pre-requisito hacer una conexión con el bloque (**ConnectorMiniBlocklyChain**) puede ejecutar todo tipo de comandos en línea y/o obtener datos específicos de ejecución de scripts o programas que tengan un CLI (Command-Line Interface) en línea.

Bloque para cerrar la sesión SSH (**DisconnectMiniBlocklyChainSSH**).

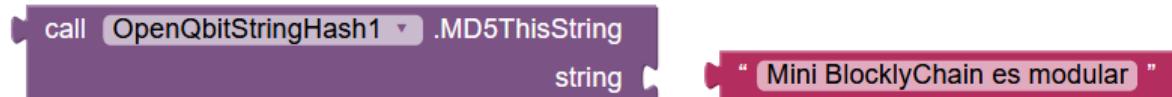


Parámetros de entrada y salida: No aplica (ninguno).

Descripción: Bloque para cerrar sesión de SSH.

Extensión OpenQbitStringHash.

Bloque para generar MD5 de cadena de caracteres (**MD5ThisString**)

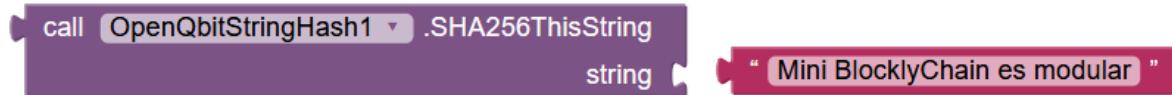


Parámetros de entrada: **string** <string>

Parámetros de salida: Entrega el hash MD5.

Descripción: Bloque para crear el hash MD5 de la cadena de caracteres dados en el parámetro de entrada.

Bloque para generar SHA256 de cadena de caracteres (**SHA256ThisString**)

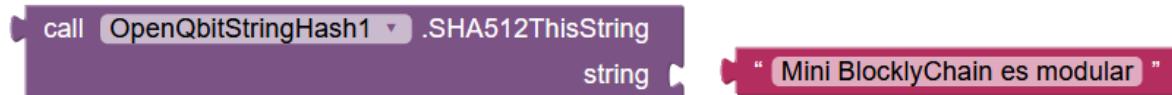


Parámetros de entrada: **string** <string>

Parámetros de salida: Entrega el hash SHA256.

Descripción: Bloque para crear el hash SHA256 de la cadena de caracteres dados en el parámetro de entrada.

Bloque para generar SHA512 de cadena de caracteres (**SHA512ThisString**)

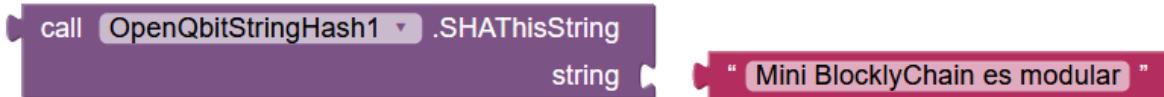


Parámetros de entrada: **string** <string>

Parámetros de salida: Entrega el hash SHA512.

Descripción: Bloque para crear el hash SHA512 de la cadena de caracteres dados en el parámetro de entrada.

Bloque para generar SHA1 de cadena de caracteres (**SHAThisString**)



Parámetros de entrada: **string** <string>

Parámetros de salida: Entrega el hash SHA1.

Descripción: Bloque para crear el hash SHA1 de la cadena de caracteres dados en el parámetro de entrada.

22. Configuración de parámetros de seguridad en Mini BlocklyChain.

Los parámetros de seguridad están divididos en tres componentes de cualquier sistema que se diseñe y se aplican en los siguientes componentes:

- a. Base de datos Redis (red de comunicación de respaldo)
- b. Sistema de sincronización “Peer to Peer” Syncthing
- c. Integrar seguridad para proteger la base de datos SQLite
- d. Ambiente para desarrolladores en la integración de módulos.

a. Base de datos Redis (red de comunicación de respaldo)

Renombrar comandos peligrosos, la característica adicional de seguridad incorporada en Redis implica renombrar o desactivar algunos comandos considerados peligrosos.

Cuando los ejecutan usuarios no autorizados, estos comandos pueden utilizarse para volver a configurar, destruir o borrar sus datos. Al igual que la contraseña de autenticación, el cambio de nombres o la inhabilitación de comandos se configura en la misma sección SECURITY del archivo /etc/redis/redis.conf.

Algunos de los comandos considerados peligrosos: FLUSHDB, FLUSHALL, KEYS, PEXPIRE, DEL, CONFIG, SHUTDOWN, BGREWRITEAOF, BGSAVE, SAVE, SPOP, SREM, RENAME, y DEBUG. No es una lista completa, pero renombrar o desactivar todos los comandos de esa lista es un buen comienzo para mejorar la seguridad de su servidor Redis.

Según sus necesidades específicas o las de su sitio, debe renombrar o desactivar un comando. Si sabe que nunca usará un comando que pueda someterse a manipulación, puede desactivarlo. Por otra parte, podría convenir renombrarlo.

Para habilitar o desactivar comandos de Redis, vuelva a abrir el archivo de configuración:

```
$ vi /etc/redis/redis.conf
```

Advertencia: Los siguientes pasos para desactivar y renombrar comandos son ejemplos. Sólo debe optar por desactivar o renombrar los comandos que le resulten pertinentes. Puede revisar la lista completa de comandos y determinar cómo pueden ser utilizados indebidamente en redis.io/commands.

Para desactivar un comando, simplemente renómbrelo de modo que pase a ser una cadena vacía (simbolizado por un par de comillas sin caracteres entre ellos), como se muestra a continuación:

```
/etc/redis/redis.conf
```

```
...
# It is also possible to completely kill a command by renaming it into
# an empty string:
#
rename-command FLUSHDB ""
rename-command FLUSHALL ""
rename-command DEBUG ""
```

Para renombrar un comando, asígnele otro nombre como se muestra en los ejemplos siguientes. Los comandos renombrados deben ser difíciles de adivinar para otros, pero fáciles de recordar para usted.

/etc/redis/redis.conf

```
    . . .
# rename-command CONFIG ""
rename-command SHUTDOWN SHUTDOWN_MENOT
rename-command CONFIG ASC12_CONFIG
    . . .
```

Guarde los cambios y cierre el archivo.

Después de renombrar un comando, aplique el cambio reiniciando Redis:

- sudo systemctl restart redis.service

Para probar el nuevo comando, ingrese la línea de comandos de Redis:

- redis-cli

Luego, realice la autenticación:

- auth your_redis_password

Output

OK

Como en el ejemplo anterior, supongamos que cambió el nombre del comando CONFIG por ASC12_CONFIG. Primero, intente utilizar el comando CONFIG original. Debido a que lo renombró, no debe funcionar:

- config get requirepass

Output

(error) ERR unknown command 'config'

Sin embargo, se podrá llamar al comando renombrado con éxito. No distingue entre mayúsculas y minúscula:

- asc12_config get requirepass

Output

1) "requirepass"
2) "your_redis_password"

Por último, podrá cerrar redis-cli:

- exit

Considere que, si ya utiliza la línea de comandos de Redis y reinicia Redis, deberá volver a realizar la autenticación. De lo contrario, si escribe un comando aparecerá este error:

Output
NOAUTH Authentication required.

b. Sistema de sincronización “Peer to Peer” Syncthing

En el uso del sistema “Peer to Peer” entre nodos el sistema cuenta con los siguientes tres elementos aplicados en la red de comunicación:

- Privada: no hay información guardada en ningún lugar que no sean tus computadoras. No hay un servidor central que pueda ser comprometido (legal o ilegalmente).
- Encriptada: Toda comunicación es asegurada a través del protocolo TLS (Seguridad en la Capa de Transporte); un protocolo criptográfico que incluye una secuencia perfecta para prevenir que nadie fuera de tu confianza pueda acceder a tu información.
- Autenticada: Cada nodo es identificado con un fuerte certificado criptográfico. Solo los nodos que hayas explícitamente permitido, pueden conectarse a tu información.

<https://docs.syncthing.net/users/security.html>

Uso del comando en línea SyncthingManager:

<https://github.com/classicsc/syncthingmanager>

c. Integrar seguridad para proteger la base de datos SQLite

Cuando se use la extensión (**OpenQbitSQLite**) o en su caso la extensión (**ConektorSSHClient**) para usar el CLI de SQLite se podrá combinar ambas extensiones con la extensión de seguridad AES (**OpenQbitEncDecData**).

Ver Anexo “Ejemplo de creación de sistema Mini BlocklyChain”.

d. Ambiente para desarrolladores en la integración de módulos.

Para implementar seguridad en el ambiente de desarrollo se podrá realizar con dos procesos:

- Restringir uso de librerías del OpenJDK.
- Uso restringido a nodos del sistema autorizados.

23. Anexo “Creación de bases de datos KeyStore & PublicKeys”.

Un KeyStore es un repositorio de certificados de seguridad, ya sea certificados de autorización o certificados de clave pública, password o claves genéricas de seguridad como las claves privadas correspondientes (direcciones) de un usuario, que se utiliza para crear o procesar transacciones.

Es una base de datos cifrada para que únicamente el dueño pueda hacer uso de esta. Normalmente es de tipo local, sin embargo, en el sistema de Mini BlocklyChain es una base de datos con seguridad sin embargo es compartida y distribuida en todos los nodos, esto se debe básicamente a una razón sencilla, todos los nodos deben de saber las direcciones de todos los usuarios (direcciones públicas), las direcciones privadas siempre son locales y son de uso único y exclusivo de cada usuario, sin embargo al realizar una transacción de entrada(deposito) o salida(gasto) toda transacción siempre tiene como mínimo tres componentes al ser creada: Dirección origen, Direcciones destino y activo o valor que se esté enviando.

Para crear nuestro KeyStore tendremos que seguir los siguientes pasos y requerimientos.

Un primer requisito es tener un tipo de almacén en donde se guardarán, en nuestro caso usaremos la base de datos SQLite y crearemos dos KeyStore una con las llaves privadas del usuario que será local y estará asegurada “cifrada” la información y otra base de datos que almacenará las llaves publicas esta base será compartida en todos los nodos de la red, la base que estará compartida en la red también estará cifrada, sin embargo esta contara con un password que únicamente los integrantes (nodos) de la red podrán compartir.

Segundo requisito fundamental es el proceso de cifrado de la información, este lo realizaremos con la extensión especializada para utilizar en base de datos genérica llamado (**OpenQbitEncDecData**) que usa un algoritmo AES.

La extensión (**OpenQbitEncDecData**) es funcional para la verificación de elementos unitarios de la cadena de bloques, sin embargo, en caso de tener que revisar la integridad total de la cadena de bloques no será eficiente por lo que también estaremos realizando un cifrado de una copia, pero en este caso será a través de las extensiones: (**OpenQbitAESEncryption**) y (**OpenQbitAESDecryption**) que trabajan sobre un archivo, no por dato referenciado.

NOTA: Debe estar funcionando el servidor de SSH en la terminal Termux para el correcto funcionamiento de la base de datos **KeyStore**. Ejecutar en la terminal el comando:

`$ sshd`

Las extensiones basadas en el algoritmo AES pueden ser usadas para cualquier tipo de datos o archivos y se eligió este algoritmo ya que es el único que ha sido demostrado que está a prueba de ataques basados en computación cuántica.

Criptosistema	Categoría	Tamaño de clave	Parámetro de seguridad	Algoritmo cuántico estimado que rompa el criptosistema	Nº de qubits lógicos necesarios	Nº de qubits físicos necesarios	Tiempo necesario para romper el sistema	Estrategias de reemplazo cuántico-resilientes
AES-GCM	Cifrado simétrico	128	128	Algoritmo de Grover	2.953	$4,61 \times 10^6$	$2,61 \times 10^{12}$ años	
		192	192		4.449	$1,68 \times 10^7$	$1,97 \times 10^{22}$ años	
		256	256		6.681	$3,36 \times 10^7$	$2,29 \times 10^{32}$ años	
RSA	Cifrado asimétrico	1024	80	Algoritmo de Shor	2.290	$2,56 \times 10^6$	3,58 horas	Migrar a un algoritmo PQC seleccionado por el NIST
RSA Problema del logaritmo discreto	Cifrado asimétrico	2048	112	Algoritmo de Shor	4.338	$6,2 \times 10^6$	28,63 horas	
		4096	128		8.434	$1,47 \times 10^7$	229 horas	
		256	128		2.330	$3,21 \times 10^6$	95 horas	
SHA256	Minado de Bitcoin	N/A	72	Algoritmo de Grover	2.403	$2,23 \times 10^6$	$1,8 \times 10^6$ años	
PBKDF2 con 10.000 iteraciones	Hashing de contraseñas	N/A	66	Algoritmo de Grover	2.403	$2,23 \times 10^6$	$2,3 \times 10^7$ años	Abandonar la autenticación basada en contraseñas

La anterior tabla referenciada a Academias Nacionales de Ciencias, Ingeniería y Medicina.

<https://www.nap.edu/catalog/25196/quantum-computing-progress-and-prospects>

Descripción

La mecánica cuántica, el subcampo de la física que describe el comportamiento de partículas muy pequeñas (cuánticas), proporciona la base para un nuevo paradigma de computación. Propuesto por primera vez en la década de 1980 como una forma de mejorar el modelado computacional de los sistemas cuánticos, el campo de la computación cuántica ha atraído recientemente una atención significativa debido al progreso en la construcción de dispositivos a pequeña escala. Sin embargo, se requerirán avances técnicos significativos antes de que se pueda lograr una computadora cuántica práctica a gran escala.

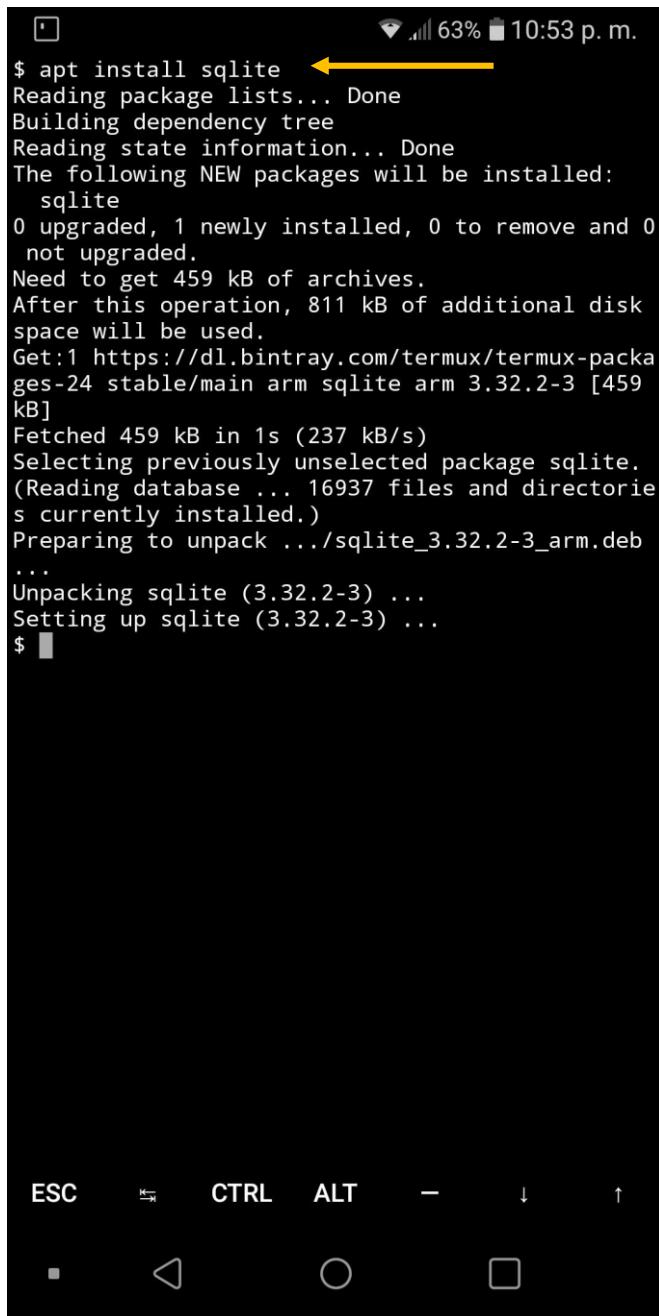
Quantum Computing: Progress and Prospects proporciona una introducción al campo, incluidas las características y limitaciones únicas de la tecnología, y evalúa la viabilidad y las implicaciones de crear una computadora cuántica funcional capaz de abordar problemas del mundo real. Este informe considera los requisitos de hardware y software, algoritmos cuánticos, impulsores de avances en computación cuántica y dispositivos cuánticos, puntos de referencia asociados con casos de uso relevantes, el tiempo y los recursos necesarios, y cómo evaluar la probabilidad de éxito.

Instalación de manejador de base de datos SQLite en terminal TERMUX y probamos que funcione correctamente el CLI (Command-Line) del comando `sqlite3` creando una base de datos llamada `keystore.db`

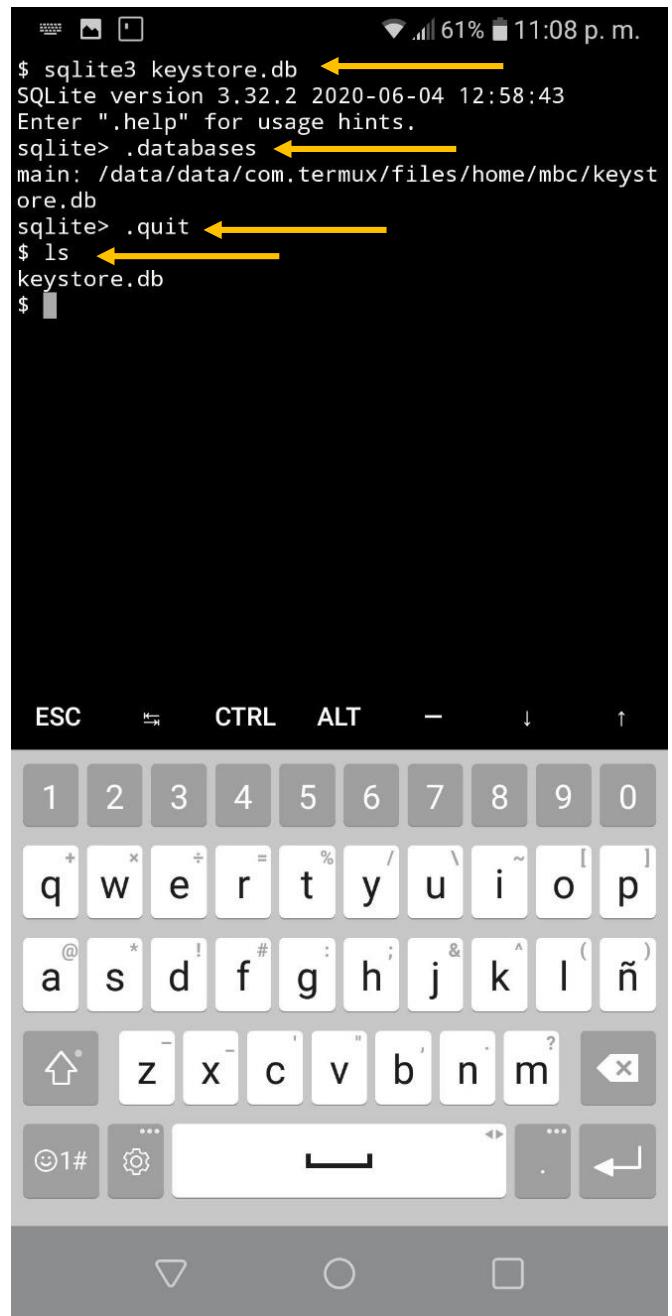
Usamos los comandos:

```
$ apt install sqlite
```

```
$ sqlite3 keystore.db
```



```
$ apt install sqlite
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  sqlite
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 459 kB of archives.
After this operation, 811 kB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-packages-24 stable/main arm sqlite arm 3.32.2-3 [459
kB]
Fetched 459 kB in 1s (237 kB/s)
Selecting previously unselected package sqlite.
(Reading database ... 16937 files and directorie
s currently installed.)
Preparing to unpack .../sqlite_3.32.2-3_arm.deb
...
Unpacking sqlite (3.32.2-3) ...
Setting up sqlite (3.32.2-3) ...
$
```



```
$ sqlite3 keystore.db
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
sqlite> .databases
main: /data/data/com.termux/files/home/mcb/keyst
ore.db
sqlite> .quit
$ ls
keystore.db
$
```

Después dentro del manejador **sqlite>** ejecutamos la sentencia .databases para revisar en que ruta (path) esta la base de datos que estamos creando, despues para salir y salvar la base de datos damos la sentencia de .quit.

NOTA: En ambas sentencias o comando dentro del manejador de **sqlite>** se tiene que primero poner un punto “.” en la sintaxis.

Por ultimo revisamos que ya haya sido creada la base de datos (vacía) dando el comando:

\$ ls

Procedemos a crear una tabla en donde se almacenarán las llaves primarias en tres diferentes formatos: hexadecimal, binario y la dirección de referencia del usuario de Mini BlocklyChain que es llave publica de su respectiva llave primaria.

El cifrado de datos AES se aplicará únicamente en los formatos hexadecimal y binario. En el caso de la dirección del usuario addrMBC y el alias no porque es la llave publica la cual se puede y deberá compartir a toda la red para poder recibir transacciones, así como el alias para realizar la búsqueda por este campo.

Creacion una tabla llamada “privatekey” en la base de datos en SQLite (keystore.db).

```
CREATE TABLE privatekey (
    id integer primary key
    , alias      VARCHAR(50) NOT NULL
    , addrHex   VARCHAR(65) NOT NULL
    , addrMBC   VARCHAR(65) NOT NULL
    ,addrBin    BLOB NOT NULL
);
```

Ejecutemos las sentencias en el CLI de sqlite para crear la base de datos keystore.db.

Volvamos a usar la línea de comando de sqlite3 con el siguiente comando:

\$ sqlite3

Este nos enviara dentro del manejador de bases de datos de **sqlite>** en este primero abriremos la base de datos ya creada para poder trabajar en ella con la siguiente sentencia dentro del manejador:

Sqlite> .open keystore.db

Después ingresaremos la sentencia de SQL “CREATE TABLE” para crear la tabla **privatekey**.

Enseguida se muestra dos opciones para crear la misma tabla, una es a través de una sola línea donde se incluye toda la sentencia SQL de los elementos que la integran. El segundo ejemplo es a través de introducir cada elemento que integra la estructura de forma segmentada.

```
$ sqlite3
```

```
SQLite version 3.32.2 2020-06-20 15:25:24
```

```
Enter ".help" for usage hints.
```

```
Connected to a transient in-memory database.
```

```
Use ".open FILENAME" to reopen on a persistent database.
```

```
sqlite> .open keystore.db
```

```
sqlite> CREATE TABLE privatekey (id integer primary key AUTOINCREMENT NOT NULL, alias VARCHAR(50) NOT NULL, addrHex VARCHAR(65) NOT NULL, addrMBC VARCHAR(65) NOT NULL, addrBin BLOB NOT NULL);
```

```
sqlite> .quit
```

Enseguida se muestra la creación de la misma tabla **privatekey** pero es mediante la introducción de la sentencia SQL en forma segmentada:

```
sqlite> CREATE TABLE privatekey ( ...> id integer primary key AUTOINCREMENT, ...> alias VARCHAR(50) NOT NULL, ...> addrHex VARCHAR(65) NOT NULL, ...> addrMBC VARCHAR(65) NOT NULL, ...> addrBin BLOB NOT NULL ...> );
```

```
sqlite> .tables
```

```
privatekey
```

```
sqlite> .quit
```

Los dos anteriores ejemplos dan el mismo resultado. Posteriormente ejecutamos la sentencia **.tables** para revisar que la tabla **privatekey** haya sido creada, al final daremos la sentencia de **.quit** con este proceso ya hemos creado la tabla **privatekey** dentro de la base de datos SQLite **keystore.db**

Hasta este momento ya tenemos la estructura de la base de datos **keystore.db** y su tabla **privatekey** donde será almacenadas las llaves privadas de una forma cifrada.

Lo anterior deberá mostrar algo similar en el nodo (teléfono móvil) con la terminal TERMUX.

```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open keystore.db
sqlite> CREATE TABLE privatekey (
...>     id integer primary key AUTOINCREMENT,
...>     alias VARCHAR(50) NOT NULL,
...>     addrHex VARCHAR(65) NOT NULL,
...>     addrMBC VARCHAR(65) NOT NULL,
...>     addrBin BLOB NOT NULL
...> );
sqlite> .tables
privatekey
sqlite> .quit
$
```

Sentencias SQL para crear nueva tabla “privatekey”.

Ahora utilizaremos algunas extensiones de seguridad para insertar datos “llaves privadas” y posteriormente consultar estos datos.

Usaremos el bloque para generar una nueva dirección de usuario (**GenerateAddrMiniBlocklyChain**), este bloque nos entregara la dirección privada en dos formatos (hexadecimal y binario), así como la dirección publica en formato de dirección genérica de MBC. También usaremos la extensión (**OpenQbitSSHClient**) y la extensión (**OpenQbitEncDecData**) para ver más detalle de estas revisar la sección de “Definición y uso de bloques de seguridad”. En la terminal Termux debe estar corriendo el servicio de SSH.

INSERTAR datos cifrados en base de datos keystore.db

```

when SveKeyStore .Click
do evaluate but ignore result
call OpenQbitBlock1 .GenerateAddrMiniBlocklyChain
    qrng " 11, 2, 45, 89, 23, 5, 8, 66, 3, 1, 99 "
    pathFilePrivateKey "/mnt/sdcard/dcim/MiniPrivate.key"

```

```

call OpenQbitBlock1 .ApiGetQRNGinteger
    qty
    min
    max

```

Serie de números aleatorios cuánticos generada por el bloque (ApiGetQRNGInteger) revisar cómo dar formato de resultado JSON, ya que la entrada del bloque (GenerateAddrMiniBlocklyChain) necesita una serie de números únicamente separados por comas ",".

```

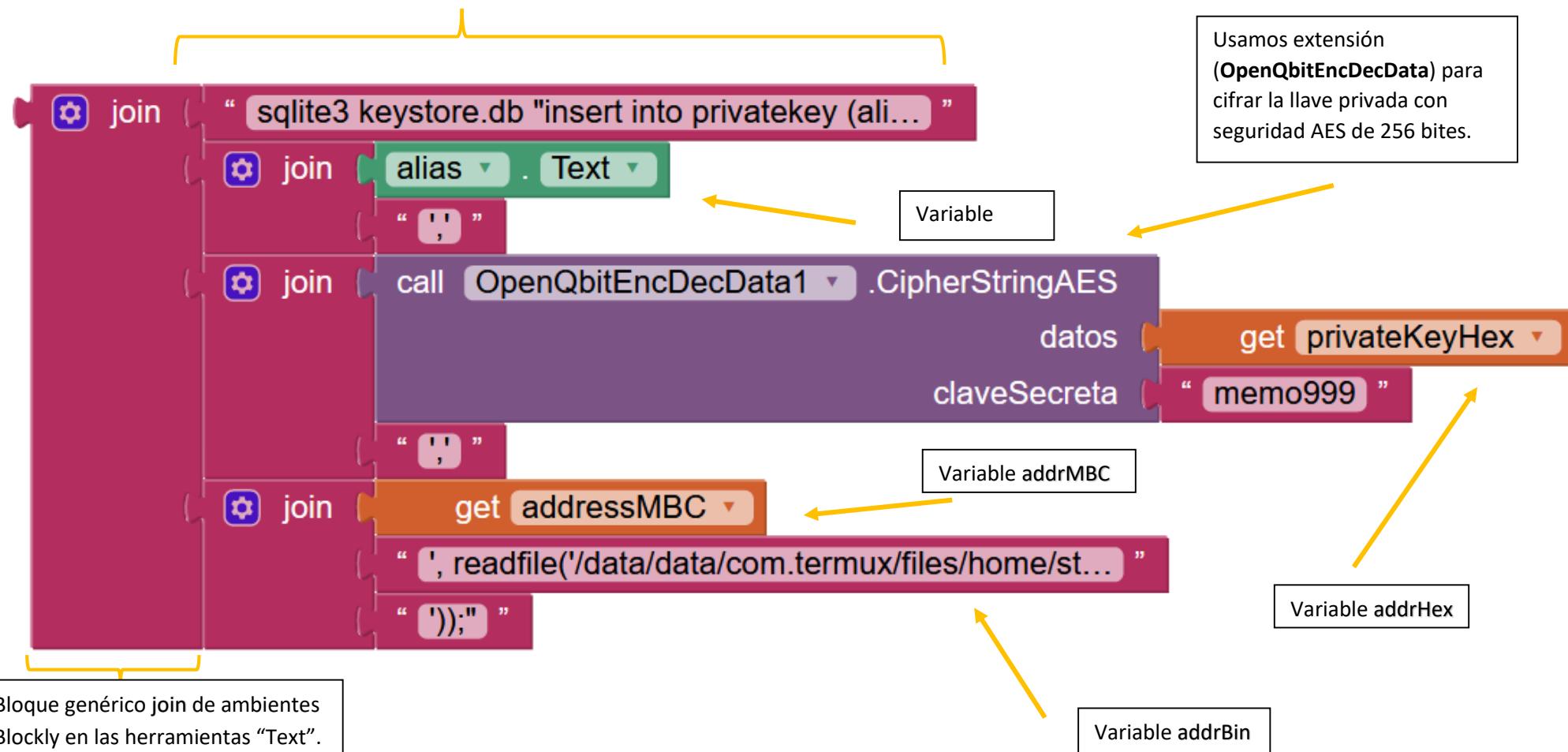
when OpenQbitBlock1 .PairKeysMBC
    addressMBC
    privateKeyHex
    pathFilePrivateKey
do evaluate but ignore result
call OpenQbitSSHClient1 .ConnectorMiniBlocklyChain
    username " u0_a251 "
    password " memo1234 "
    host " localhost "
    port 8022
set global SelectSQLenc to
call OpenQbitSSHClient1 .CommandLineMiniBlocklyChain
    command join " sqlite3 keystore.db \"insert into privatekey (ali...\" "
        join alias Text
            " "
        join call OpenQbitEncDecData1 .CipherStringAES
            datos get privateKeyHex
            claveSecreta " memo999 "
        join " "
    join get addressMBC
        " , readfile('/data/data/com.termux/files/home/st...\" "
        " ') ); \" "
call OpenQbitSSHClient1 .DisconnectMiniBlocklyChainSSH

```

La sintaxis del comando es muy importante, debemos introducir el siguiente comando en el formato adecuado en el ambiente Blockly.

Los valores de values siempre serán las variables, ejemplo:

```
sqlite3 keystore.db "insert into privatekey (alias, addrHex, addrMBC, addrBin) values ('memo', 'QWERTY', 'MBC12345',  
readfile('/data/data/com.termux/files/home/storage/dcim/MiniPrivate.key'));"
```



Después de ejecutar los bloques tendremos un resultado en la tabla privatekey de la base de datos keystore.db similar al siguiente:

Entramos al manejador de sqlite en la terminal de Termux, abrimos la base keystore.db y ejecutamos la sentencia:

```
$ sqlite3
```

SQLite version 3.32.2 2020-06-20 15:25:24

Enter ".help" for usage hints.

Connected to a transient in-memory database.

Use ".open FILENAME" to reopen on a persistent database.

```
sqlite> .open keystore.db
```

```
sqlite> select * from privatekey;
```

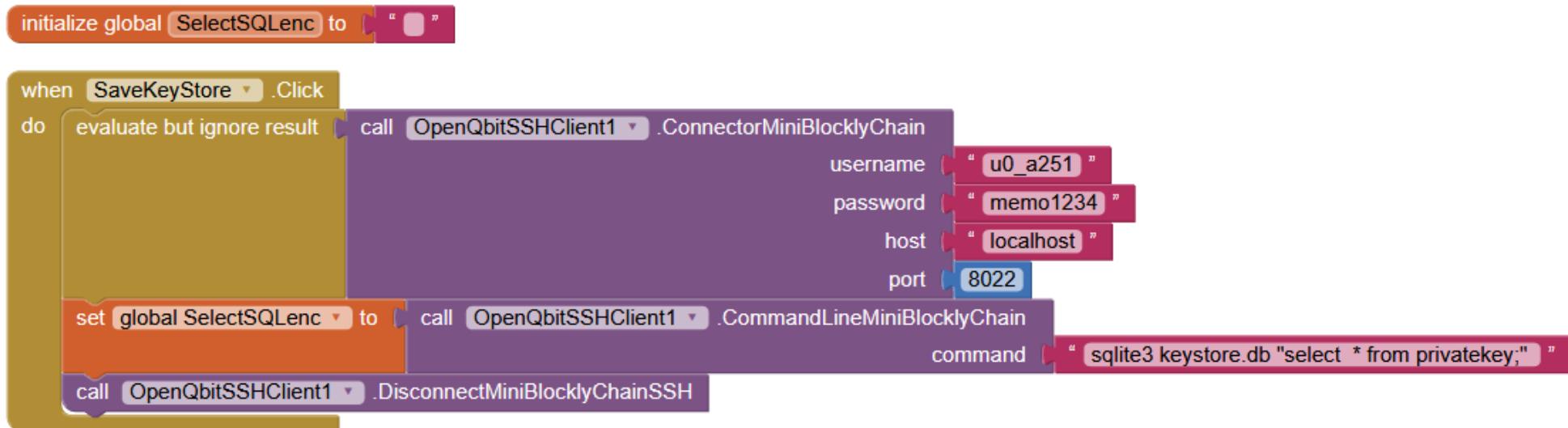
The screenshot shows a Termux terminal window with the following text output:

```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open keystore.db
sqlite> select * from privatekey;
1|mexico|07JBBizTcC0Ce6a8Pwe5aTV41Ql1DKiUQZyPSfJuRbVUZnvIwQJcry4LilBMs5jHavQeMo1iN8rCO87V5Xka2YaKR4fswopeTQmI/Q+ipzI=|2eENpFt2HSuGtXNXoeiwfrp5d6e87Z7y5|0@@
sqlite>
```

Annotations with yellow arrows point to specific parts of the output:

- An arrow points from a box labeled "Alias: mexico" to the alias "mexico" in the output.
- An arrow points from a box labeled "addrBin (llave privada en binario)" to the binary key "1|mexico|07JBBizTcC0Ce6a8Pwe5aTV41Ql1DKiUQZyPSfJuRbVUZnvIwQJcry4LilBMs5jHavQeMo1iN8rCO87V5Xka2YaKR4fswopeTQmI/Q+ipzI=|2eENpFt2HSuGtXNXoeiwfrp5d6e87Z7y5|0@@".
- An arrow points from a box labeled "addrHex cifrado." to the hex string "1|mexico|07JBBizTcC0Ce6a8Pwe5aTV41Ql1DKiUQZyPSfJuRbVUZnvIwQJcry4LilBMs5jHavQeMo1iN8rCO87V5Xka2YaKR4fswopeTQmI/Q+ipzI=|2eENpFt2HSuGtXNXoeiwfrp5d6e87Z7y5|0@@".
- An arrow points from a box labeled "Dirección addrMBC" to the public key "1|mexico|07JBBizTcC0Ce6a8Pwe5aTV41Ql1DKiUQZyPSfJuRbVUZnvIwQJcry4LilBMs5jHavQeMo1iN8rCO87V5Xka2YaKR4fswopeTQmI/Q+ipzI=|2eENpFt2HSuGtXNXoeiwfrp5d6e87Z7y5|0@@".

CONSULTAR todos los datos en la tabla **privatekey** de la base de datos SQLite **keystore.db**



CONSULTAR los alias de los datos en la tabla **privatekey** de la base de datos SQLite **keystore.db**

sqlite3 keystore.db "select alias from privatekey;"

CONSULTAR todos los datos de la columna addrHex cifrados en la tabla **privatekey** de la base de datos SQLite **keystore.db**

sqlite3 keystore.db "select addrHex from privatekey;"

CONSULTAR para recuperar la llave privada addrBin en la tabla **privatekey** de la base de datos SQLite **keystore.db**

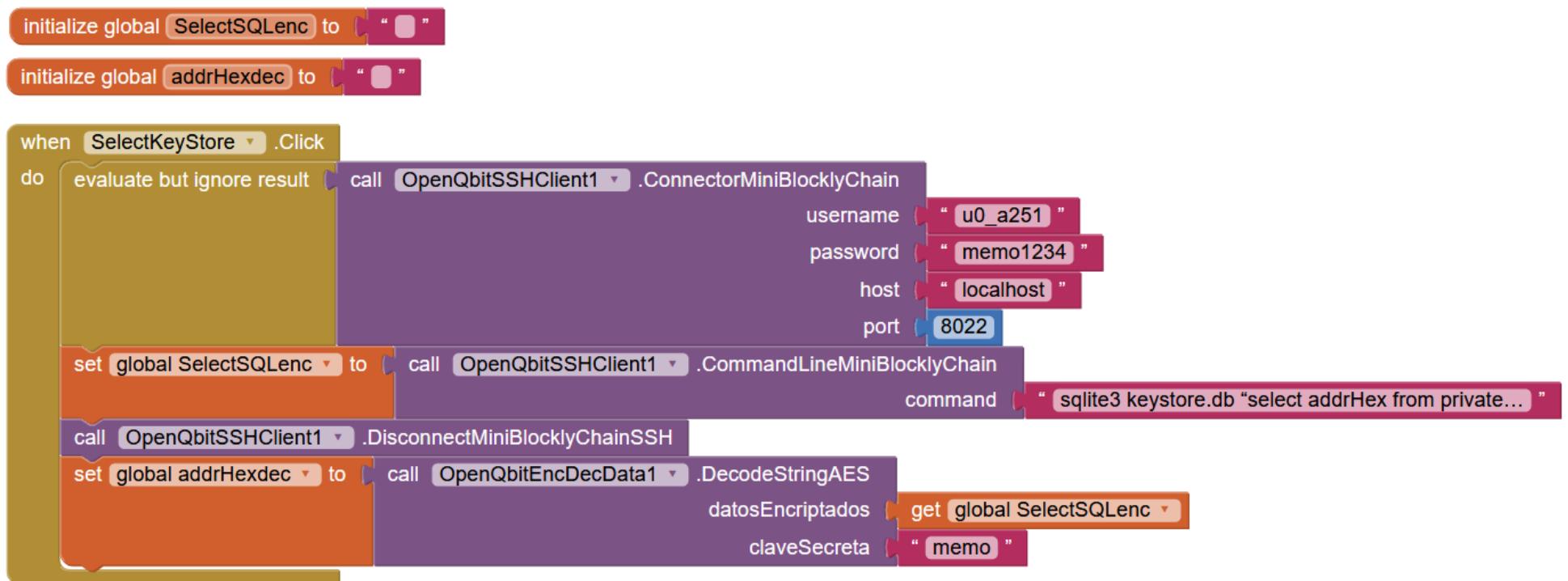
sqlite3 **writefile('PrivateKey.key', addrBin)** FROM privatekey WHERE alias='mexico'; (2)

(1) Este tipo de consulta será la principal para consultar la llave privada para realizar el proceso de la firma de las transacciones.

CONSULTAR PARA DESCIFRAR DATOS por alias de la columna addrHex en la tabla **privatekey** de la base de datos SQLite **keystore.db**

En este caso usaremos el bloque (**DecodeStringAES**) para descifrar los datos almacenados en la columna “addrHex”.

sqlite3 keystore.db “select addrHex from privatekey where='mexico';”



Esta consulta nos entrega la llave privada en formato hexadecimal, esta es la parte fundamental de cualquier recepción o envío de transacciones. Se recomienda de manera reiterativa que se guarde un respaldo de esta base de datos keystore.db

Diseño de base de datos **publickeys.db** con tabla **publicaddr**:

```
$ sqlite3
```

SQLite version 3.32.2 2020-06-20 15:25:24

Enter ".help" for usage hints.

Connected to a transient in-memory database.

Use ".open FILENAME" to reopen on a persistent database.

```
sqlite> .open publickeys.db
```

```
sqlite> CREATE TABLE publicaddr (id integer primary key AUTOINCREMENT NOT NULL,
    pubMBC VARCHAR NOT NULL, pubHex VARCHAR NOT NULL, pubBin BLOB NOT NULL);
```

```
sqlite> .quit
```

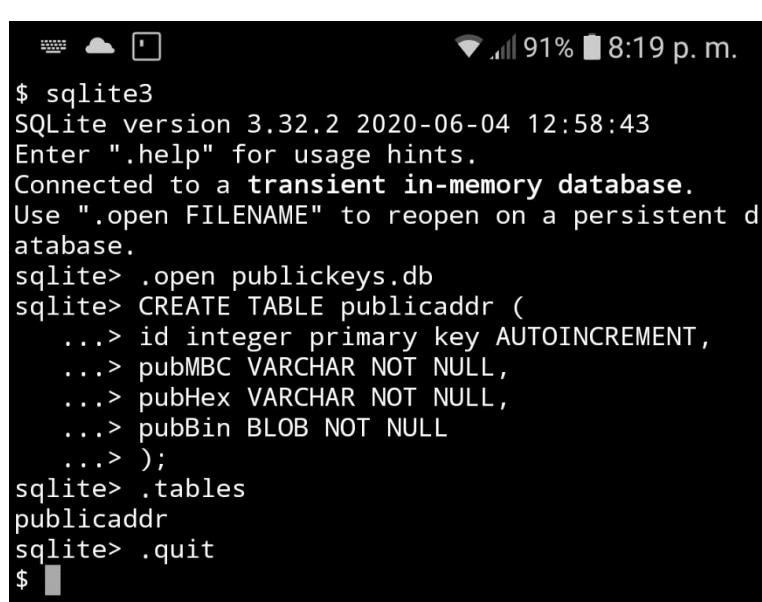
Enseguida se muestra la creación de la tabla **publicaddr** con la siguiente la sentencia SQL en forma segmentada:

```
sqlite> CREATE TABLE publicaddr (
...>   id integer primary key AUTOINCREMENT,
...>   pubMBC VARCHAR NOT NULL,
...>   pubHex VARCHAR NOT NULL,
...>   pubBin BLOB NOT NULL
...> );
```

```
sqlite> .tables
```

publicaddr

```
sqlite> .quit
```



The screenshot shows a terminal window on a mobile device. The status bar at the top indicates signal strength, battery level (91%), and the time (8:19 p.m.). The terminal prompt is '\$'. The user enters the command '\$ sqlite3' followed by the SQLite version information and help instructions. Then, the user connects to a transient in-memory database and creates the 'publicaddr' table with four columns: 'id' (integer primary key AUTOINCREMENT), 'pubMBC' (VARCHAR NOT NULL), 'pubHex' (VARCHAR NOT NULL), and 'pubBin' (BLOB NOT NULL). Finally, the user lists the tables in the database and exits the SQLite shell.

```
$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open publickeys.db
sqlite> CREATE TABLE publicaddr (
...>   id integer primary key AUTOINCREMENT,
...>   pubMBC VARCHAR NOT NULL,
...>   pubHex VARCHAR NOT NULL,
...>   pubBin BLOB NOT NULL
...> );
sqlite> .tables
publicaddr
sqlite> .quit
$
```

24. Anexo “RESTful SQLite comandos GET/POST”.

En seguida se muestran los diferentes comandos que podemos usar en el Restful SQLite de la red de respaldo. Estas se han consultado del desarrollo original de GITHUB (<https://github.com/olsonpm/sqlite-to-rest>).

Operaciones CRUD (Create, Read, Update y Delete) RESTful.

La siguiente es una lista de las operaciones disponibles puestas a disposición por la API RESTful en forma de pseudo ejemplos. Asumiremos una base “op.sqlite” y dos tablas asociadas “trans” para transacciones y otra “sign” para la dirección origen, dirección destino y valor del activo con dos columnas id INTEGER PRIMARY KEY.

Como se señaló en las limitaciones, tenga en cuenta que los métodos (DELETE y POST) solo pueden afectar una fila a la vez.

OBTENER Esto permite la mayor variación. Más adelante veremos todos los operadores de consulta disponibles.

Los encabezados se pueden especificar justo debajo de las URL.

- **/trans**

Requests for all rows

- **/trans?id=1**

Where id = 1

- **/trans**

range: rows=0-2

First three rows

- **/trans**

range: rows=-5

Last five rows

- **/trans**

range: rows=0-

Tantas filas como pueda proporcionar el servidor, que en la práctica será el menor de maxRange y el recuento total de filas.

/trans

range: rows=1-

Tantas filas como pueda proporcionar el servidor, comenzando desde la fila 1.

- **/trans**

order: name

Ordered by name ascending

- **/trans**

order: name desc

Ordered by name descending

- **/trans**

order: name desc,id

Contribuido, pero ordena primero por nombre descendente, y en el caso de un empate por id ascendente.

- **/trans?id>1**

Donde id > 1

- **/trans?id>=2&id<5**

Donde id >= 2 and id < 5

- **/trans?name_NOTNULL**

Donde name es no null

- **/trans?name_ISNULL**

Donde name es null

- **/trans?id!=5&name_LIKE'Spotted%'**

Donde id != 5 y name es LIKE "Spotted%" (ignorar citas)

- **/trans?id>=1&id<10&name_LIKE'Avery%'**

order: name desc,id range: rows=2-4

Contribuido por el bien del ejemplo.

Obtenga transacción con identificadores entre 1 y 9 inclusive, con un nombre como "Avery%", ordenado primero por nombre descendente y luego por identificador ascendente, obteniendo la fila tercera a quinta del resultado. O en SQL:

DELETE Requiere una cadena de consulta con todas las claves primarias establecidas iguales a un valor. Esto exige una eliminación máxima de una sola fila.

/trans?id=1

Deletes trans with id=1

Si la transacción en su lugar tuviera una clave principal compuesta de id y nombre.

/trans?id=1&name='Avery IPA'

POST crear

No debe pasar una cadena de consulta. Si se pasa una cadena de consulta, se supone la actualización POST. Todas las solicitudes POST deben pasar el tipo de contenido del encabezado: application / json.

Tenga en cuenta que el cuerpo debe contener todas las columnas CLAVE PRIMARIA no anulables y no INTEGER. En caso contrario, se enviará una respuesta 400 indicando qué campos se perdieron. Las columnas anulables serán nulas y las columnas INTEGER PRIMARY KEY se incrementarán automáticamente según las especificaciones sqlite3.

Los datos de JSON se especificarán justo debajo de las URL.

- **/trans**

```
{"id":1,"name":"Serendipity"}
```

Creates a trans with id = 1 and name = 'Serendipity'

- **/trans**

```
{"id":1}
```

Creates a trans with id = 1 and name = NULL

- **/trans**

```
{"name":"Serendipity"}
```

Crea una transacción con id establecida en el siguiente valor incrementado por sqlite3 Especificaciones INTEGER PRIMARY KEY.

/trans

```
{ }
```

Crea una transacción con id incrementado y el nombre establecido en NULL.

Actualización POST

Debe contener una cadena de consulta. Sin una cadena de consulta, se supone la creación POST. Al igual que con POST create, el tipo de contenido del encabezado: application / json es obligatorio.

La cadena de consulta debe contener todas las claves primarias para garantizar que solo se actualice una fila. Si se pasan valores incorrectos, se devolverá un 400 con las claves infractoras.

El cuerpo de la solicitud debe contener un objeto no vacío y debe contener claves válidas correspondientes a los nombres de columna.

Los datos de JSON se especificarán justo debajo de las URL.

/trans?id=1

{ "id":2 }

Actualiza la transacción con una identificación de 1 configurándola en dos.

/trans?id=1

{ "name": "MCBza45Rt56cvbgfdR2Swd788kj" }

Actualiza la transacción con la identificación de 1 estableciendo su nombre o valor en "MCBza45Rt56cvbgfdR2Swd788kj".

Si la mesa de transacción en su lugar tuviera una clave principal compuesta de id y nombre.

/trans?id=1&name=MCBza45Rt56cvbgfdR2Swd788kj

{ "name": "MCB3deFG5Hj678MNb09vLdfaasx" }

Actualiza la transacción donde id es uno y la dirección es MCBza45Rt56cvbgfdR2Swd788kj, estableciendo la MCB3deFG5Hj678MNb09vLdfaasx.

Referencia

`isSqliteFile`

Simplemente verifica los primeros 16 bytes del archivo para ver si es igual a 'sqlite format 3' seguido de un byte nulo.

`isDirectory`

Devuelve el resultado de `fs.statsSync` seguido de `.isDirectory`

`isFile`

Devuelve el resultado de `fs.statsSync` seguido de `.isFile`

GET operadores de consulta

Las condiciones de consulta deben estar delimitadas por símbolos de unión p. Ej. `id> 5 & name!= MCBAza45Rt56cvbgfdR2Swd788kj`

Operadores binarios (requieren un valor después) Man.

=

!=

>=

<=

>

<

LIKE

_LIKE es especial porque debe tener comillas simples de apertura y cierre. De lo contrario, se generará un error 400 que muestra dónde no se pudo completar el análisis y qué se esperaba. Ver Operaciones CRUD RESTful para ejemplos.

Operadores unarios (deben seguir el nombre de una columna)

_ES NULO

_NO NULO

Objeto de configuración del enrutador

isLadenPlainObject

El propósito de este objeto es proporcionar una configuración genérica para el enrutador sqlite. Se permiten las siguientes propiedades:

prefijo: isLadenString La cadena pasada a la opción de constructor de prefijo de koa-router. Por ejemplo, el servidor de esqueleto no especifica un prefijo, lo que permite que la API de cerveza sea golpeada directamente desde la raíz del dominio http://localhost: 8085 / trans. Si establece el prefijo a '/ api', entonces debería enviar solicitudes a http://localhost: 8085 / api / trans.

allTablesAndViews: un objeto de configuración tabular

Las configuraciones especificadas en este objeto se aplicarán a todas las tablas y vistas, opcionalmente anuladas por la propiedad tablesAndViews.

tablesAndViews: isLadenPlainObject El objeto pasado debe tener claves que coincidan con la columna de la base de datos o ver los nombres. De lo contrario, se lanzará un mensaje de error amistoso. Los valores para cada tabla y vista deben ser un objeto de configuración tabular.

Objeto de configuración tabular

isLadenPlainObject Este objeto representa configuraciones que se pueden establecer para vistas o tablas. Permite las siguientes propiedades:

maxRange: isPositiveNumber

Aplicación predeterminada: 1000

Este es el rango máximo que su servidor permitirá solicitudes. Si llega una solicitud GET sin encabezado de rango, la especificación supone que desean todo el recurso. Si el número de filas que dan como resultado que GET es mayor que maxRange, se devuelve un estado 416 con el encabezado personalizado max-range. El valor predeterminado de la aplicación es deliberadamente conservador con la esperanza de que los autores establezcan maxRange de acuerdo con sus necesidades.

Tenga en cuenta que 'Infinito' es un número positivo válido.

banderas: isLadenArray

Actualmente, el único indicador aceptado es la cadena 'sendContentRangeInHEAD'. Cuando se establece, las solicitudes HEAD devolverán el rango de contenido disponible en la forma contenido-rango: * / <max-range>. La razón por la que es configurable es que calcular el rango máximo puede ser más trabajo de lo que vale, dependiendo de la carga del servidor y el tamaño de sus tablas.

Encabezados personalizados

Solicitud

orden: este encabezado solo se define para GET, y puede considerarse como el equivalente de sql ORDER BY. Debe contener un nombre de columna delimitado por comas, cada uno opcionalmente seguido de un espacio y las cadenas 'asc' o 'desc'. Si se envían valores de pedido incorrectos, una respuesta 400 indicará cuáles.

Respuesta

No todos son necesariamente personalizados, pero todo su uso está fuera de lo definido en la especificación y, por lo tanto, necesita aclaración.

OBTENER

max-range: este encabezado se devuelve cuando el número de filas solicitado supera el maxRange configurado. Tenga en cuenta que la solicitud puede no especificar el encabezado del rango, pero el número de filas que dan como resultado ese recurso aún se verificará.

rango de contenido: estados rfc7233

solo los códigos de estado 206 (Contenido parcial) y 416 (Rango no satisfactorio) describen un significado para Content-Range.

Cuando sqlite-to-rest responde con un código de estado 200, el encabezado del rango de contenido se envía con el formato 206 de <row start> - <row end> / <row count>.

Cuando se envía una solicitud sin un encabezado de rango y el número de filas resultantes supera maxRange, se devuelve un 400 con el rango de contenido establecido en el formato 416 de * / <conteo de filas>

Tenga en cuenta que este encabezado puede devolverse en una respuesta HEAD.

accept-order: se devolverá si el orden del encabezado de la solicitud tiene una sintaxis incorrecta o nombres de columna incorrectos especificados. Para más detalles, consulte HEAD -> aceptar-orden a continuación.

25. Anexo “Conector Código Java SQLite-Redis”.

En seguida se muestra un código del enlace entre la comunicación de ambas bases de datos SQLite-Redis. Básicamente como podemos observar el conector cambia el formato de SQLite a una cadena genérica de los datos (transacciones) para que la reciba Redis.

El anterior proceso funge como un disparador de la cola de transacciones hacia todos los nodos que estén conectados y que sean posibles candidatos para procesar la cola de transacciones en curso.

Al recibir la base de datos Redis la cola de transacciones por la configuración de Master-Slave que se tiene, distribuirá la información en un tiempo real e igualdad para todos los nodos.

Otro punto fundamental es que todos los nodos deberán estar sincronizados en su reloj, esto lo realizaremos como ya lo vimos en anteriormente con la funcionalidad de la consulta a un pool de servidor NTP (Network Time Protocol).

Este conector también realiza el primer nivel de revisión de seguridad de los datos mediante el cálculo del árbol de Merkle Root de todas las transacciones.

Código base del Conector SQLite-Redis.

```
import java.sql.*;
import java.util.*;
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;
import java.security.*;
import java.security.MessageDigest;
import redis.clients.jedis.Jedis;
class RediSqlite{
    public String previousHash;
    public String merkleRoot;
    public static ArrayList<String> transactions = new ArrayList<String>();
    public static ArrayList<String> treeLayer = new ArrayList<String>();
    public static String getMerkleRoot() {
        int count = transactions.size();
        int item = 1;
        int impar = 0;
        ArrayList<String> previousTreeLayer = transactions;
        while(count > 1) {
```

```

treeLayer = new ArrayList<String>();
for(int i=1; i <= count ; i=i+2) {

    if (!esPar(count) && i == count) impar = 1;
        treeLayer.add(applySha256(previousTreeLayer.get(i-item)
previousTreeLayer.get(i-impar)));
    }

    item = 1;
    impar = 0;
    count = treeLayer.size();
    previousTreeLayer = treeLayer;
}

String merkleRoot = (treeLayer.size() == 1) ? treeLayer.get(0) : "";
return merkleRoot;
}

//Define if Merkle Tree is even or odd
static boolean esPar(int numero){
    if (numero%2==0) return true; else return false;
}

public static String applySha256(String input){
try {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    //Applies sha256 to our input,
    byte[] hash = digest.digest(input.getBytes("UTF-8"));
    StringBuffer hexString = new StringBuffer(); // This will contain hash as hexadecimal
    for (int i = 0; i < hash.length; i++) {
        String hex = Integer.toHexString(0xff & hash[i]);
        if(hex.length() == 1) hexString.append('0');
        hexString.append(hex);
    }
    return hexString.toString();
}
catch(Exception e) {
    throw new RuntimeException(e);
}
}

public static void main(String args[]){
try{

```

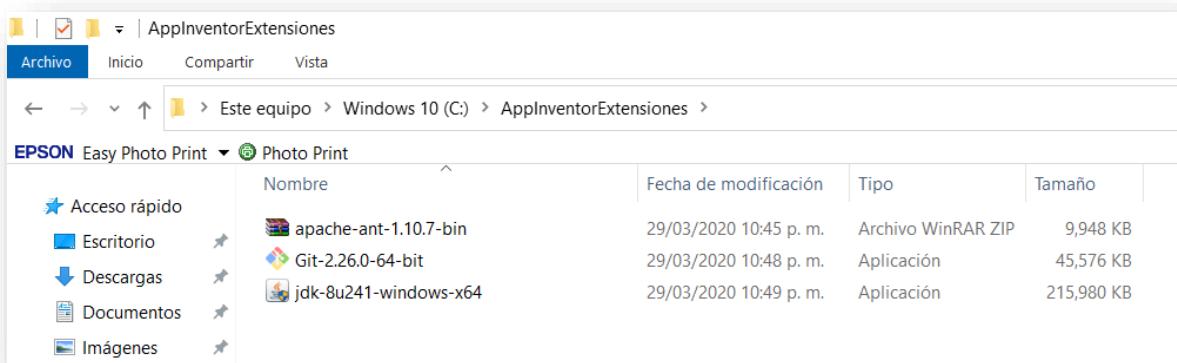
```
Connection      con=DriverManager.getConnection("jdbc:sqlite:C:/memo/sqlite-tools-
win32-x86-3310100/trans.sqlite3");
//Connecting to Redis server on localhost
Jedis = new Jedis("localhost");
jedis.auth("memo1234");
Statement stmt=con.createStatement();
String sql = "SELECT * FROM brewery";
ResultSet rs=stmt.executeQuery(sql);
while(rs.next()) {
    transactions.add(rs.getString(2));
    jedis.set("LATAM:"+String.valueOf(rs.getInt(1)), "["+"\""+rs.getString(2)+"\""+","+"\""+rs.get
String(3)+"\""+","+"\""+rs.getString(4)+"\"]");
}
jedis.set("LATAM:merkleroot", getMerkleRoot());
System.out.println("Numero de elementos ArrayList:: "+treeLayer.size());
con.close();
}catch(Exception e){ System.out.println(e);}

}
}
```

26. Anexo “Mini BlocklyChain para desarrolladores”.

En este anexo revisaremos la configuración, instalación y uso básico de cómo generar módulos externos basados en el lenguaje de programación Java para ambiente Blockly y podremos crear módulos especializados para cada caso de negocio e insertar funcionalidades al Sistema Mini BlocklyChain o agregar otras funcionalidades a nuestra aplicación móvil.

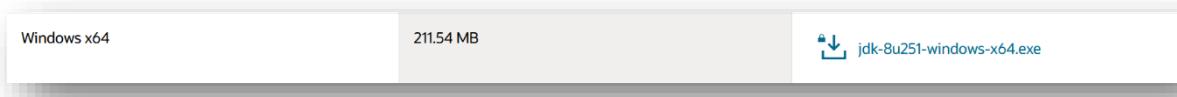
Creamos un directorio en nuestro sistema Windows llamado “ApplInventorExtensiones” y este bajaremos los siguientes paquetes (software) publico.



1.- Vamos a bajar la última versión del **JDK (Java Development Kit)**

ejemplo: jdk-8u251-windows-x64.exe (211 MB)

<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>



2.- **Apache Ant** librería que utiliza JAVA para poder construir aplicaciones, <http://ant.apache.org/bindownload.cgi>, en mi caso he bajado la Ant 1.10.8 (Binary Distributions) (apache-ant-1.10.8-bin.zip). Posiblemente haya versiones más avanzadas.

1.9.15 release - requires minimum of Java 5 at runtime

- 1.9.15 .zip archive: [apache-ant-1.9.15-bin.zip \[PGP\] \[SHA512\]](#)
- 1.9.15 .tar.gz archive: [apache-ant-1.9.15-bin.tar.gz \[PGP\] \[SHA512\]](#)
- 1.9.15 .tar.bz2 archive: [apache-ant-1.9.15-bin.tar.bz2 \[PGP\] \[SHA512\]](#)

1.10.8 release - requires minimum of Java 8 at runtime

- 1.10.8 .zip archive: [apache-ant-1.10.8-bin.zip \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.gz archive: [apache-ant-1.10.8-bin.tar.gz \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.bz2 archive: [apache-ant-1.10.8-bin.tar.bz2 \[PGP\] \[SHA512\]](#)
- 1.10.8 .tar.xz archive: [apache-ant-1.10.8-bin.tar.xz \[PGP\] \[SHA512\]](#)

Old Ant Releases

Older releases of Ant can be found [here](#). We highly recommend to not use those releases but upgrade to Ant's [latest](#) release.

3.- Instalamos el Git Bash desde si sitio <https://git-scm.com/download/win>

Downloading Git



You are downloading the latest (**2.27.0**) **64-bit** version of **Git for Windows**. This is the most recent [maintained build](#). It was released **9 days ago**, on **2020-06-01**.

[Click here to download manually](#)

Other Git for Windows downloads

[Git for Windows Setup](#)

[32-bit Git for Windows Setup.](#)

[64-bit Git for Windows Setup.](#)

[Git for Windows Portable \("thumbdrive edition"\)](#)

[32-bit Git for Windows Portable.](#)

[64-bit Git for Windows Portable.](#)

The current source code release is version **2.27.0**. If you want the newer version, you can build it from [the source code](#).

4.- Descomprimimos “Apache Ant.” Al terminar de descomprime lo hace en doble carpeta, por ejemplo:

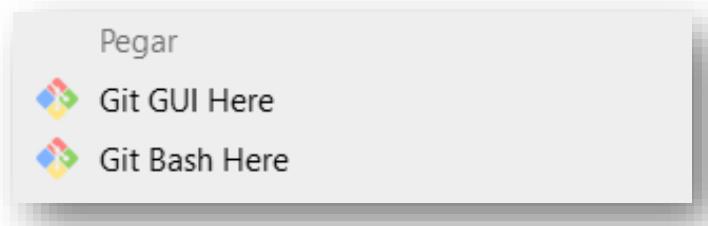
C:\AppInventorExtensiones\apache-ant-1.10.8-bin\apache-ant-1.10.8-bin\

- Cámbialo para que esté en C:\AppInventorExtensiones\apache-ant-1.10.8-bin\

Nombre	Fecha de modificación	Tipo	Tamaño
bin	01/09/2019 11:43 a. m.	Carpetas de archivos	
etc	01/09/2019 11:43 a. m.	Carpetas de archivos	
lib	01/09/2019 11:43 a. m.	Carpetas de archivos	
manual	01/09/2019 11:43 a. m.	Carpetas de archivos	
CONTRIBUTORS	01/09/2019 11:43 a. m.	Archivo	7 KB
contributors	01/09/2019 11:43 a. m.	Documento XML	33 KB
fetch	01/09/2019 11:43 a. m.	Documento XML	14 KB
get-m2	01/09/2019 11:43 a. m.	Documento XML	5 KB
INSTALL	01/09/2019 11:43 a. m.	Archivo	1 KB
KEYS	01/09/2019 11:43 a. m.	Archivo	94 KB
LICENSE	01/09/2019 11:43 a. m.	Archivo	15 KB
NOTICE	01/09/2019 11:43 a. m.	Archivo	1 KB
patch	01/09/2019 11:43 a. m.	Documento XML	2 KB
README	01/09/2019 11:43 a. m.	Archivo	5 KB
WHATSNEW	01/09/2019 11:43 a. m.	Archivo	250 KB

5.- Instalamos Git Bash. Dejamos todo por defecto en la instalación.

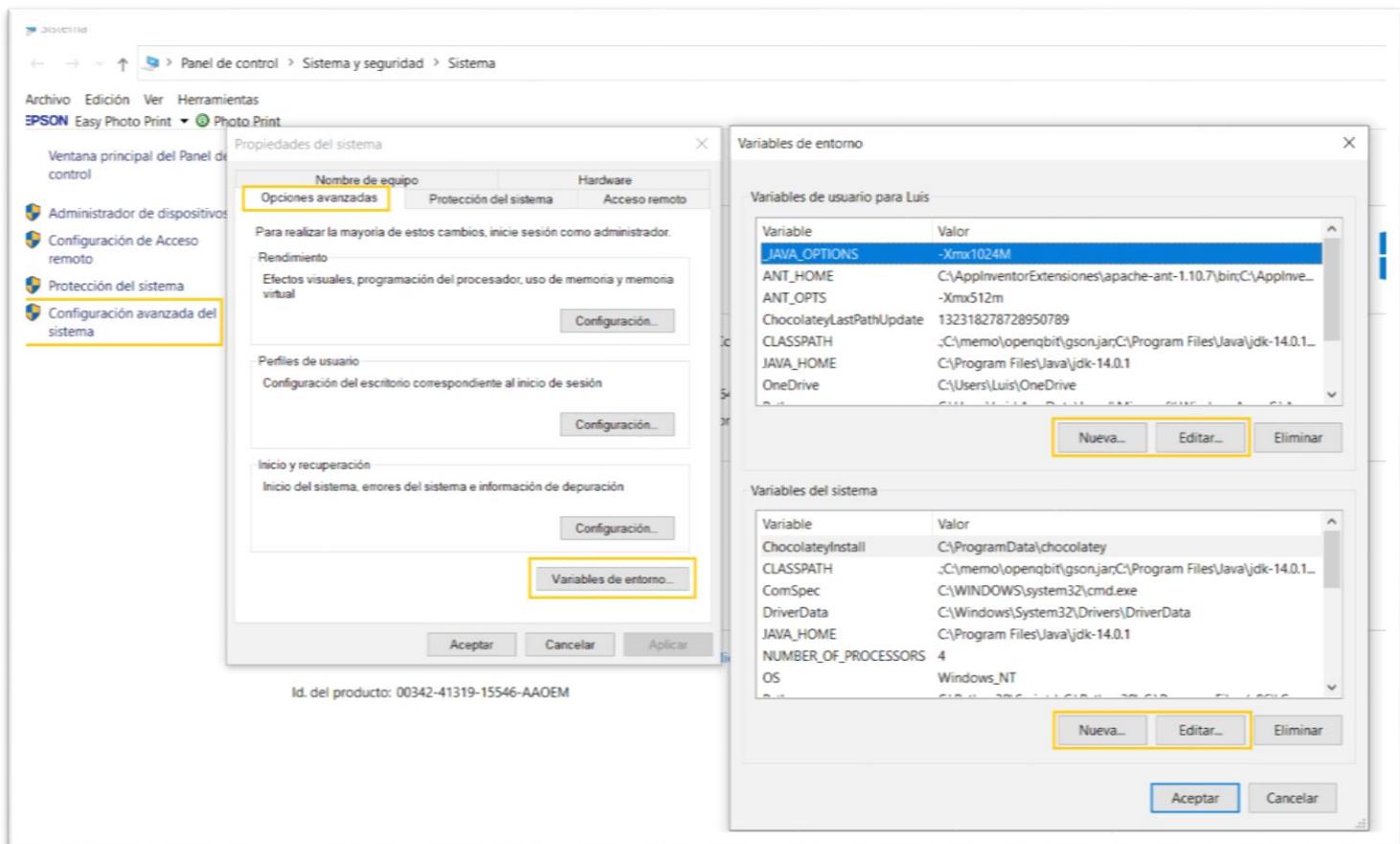
- Posteriormente podremos ver que tiene un enlace en el Botón de inicio de Windows, haciendo click en el botón izquierdo en el explorador de archivos.



6.- Instalamos el Java SE Development Kit. Dependiendo de la versión de Windows necesitaras probablemente reiniciar el equipo.

Variables de entorno del sistema Windows. Crearemos las variables de entorno. Para ello vamos a:

Panel de Control -> Sistema -> Configuración avanzada del sistema -> Opciones avanzadas -> Variables de entorno.



Según queramos poner una Nueva Variable o Editar una ya existen, pulsamos el botón de "Nueva..." o "Editar.."

Para añadir direcciones a las ya establecidas, las separamos por punto y coma;

En la parte de Variables de usuario para juan, establecemos estas Nuevas...

_JAVA_OPTIONS le ponemos de Valor -Xmx1024m

ANT_HOME le ponemos de Valor C:\ApplInventorExtensiones\apache-ant-1.10.8-bin [es decir, la carpeta donde hemos descomprimido apache-ant]

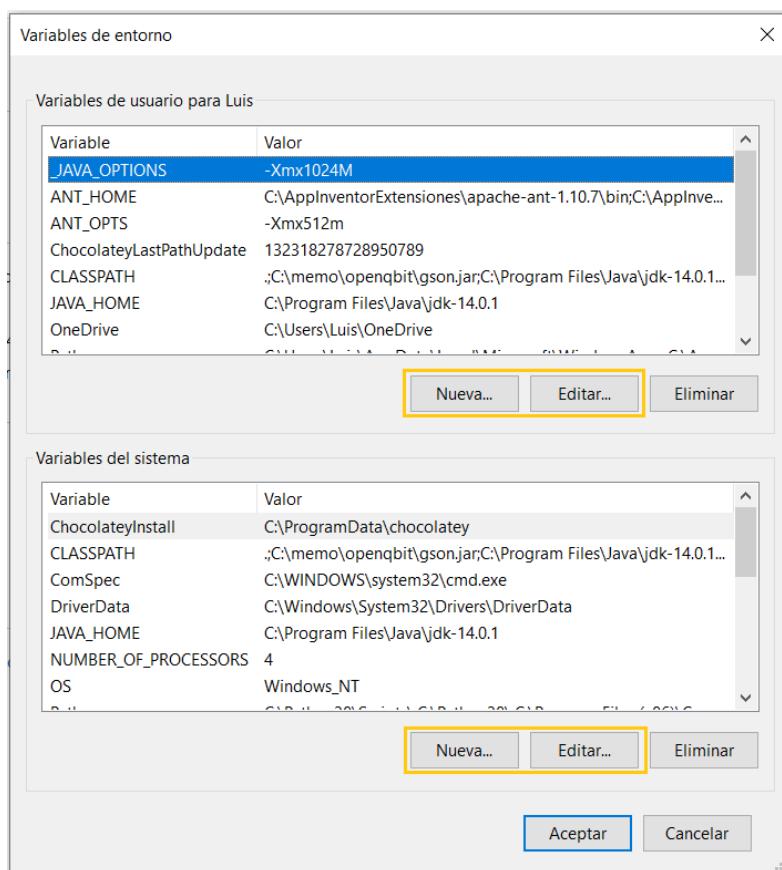
ANT_OPTS le ponemos de Valor -Xmx256M

JAVA_HOME le ponemos de Valor C:\Program Files\Java\jdk1.8.0_131 [Si tuviera otro Valor anterior, modifícalo. Observa que es jdk NO ES jdr]

CLASSPATH le ponemos de Valor %ANT_HOME%\lib;%JAVA_HOME%\lib

En PATH agregamos ;%ANT_HOME%\bin;%JAVA_HOME%\bin [Observa que ; comienza por punto y coma; para agregar a las que ya estaban.]

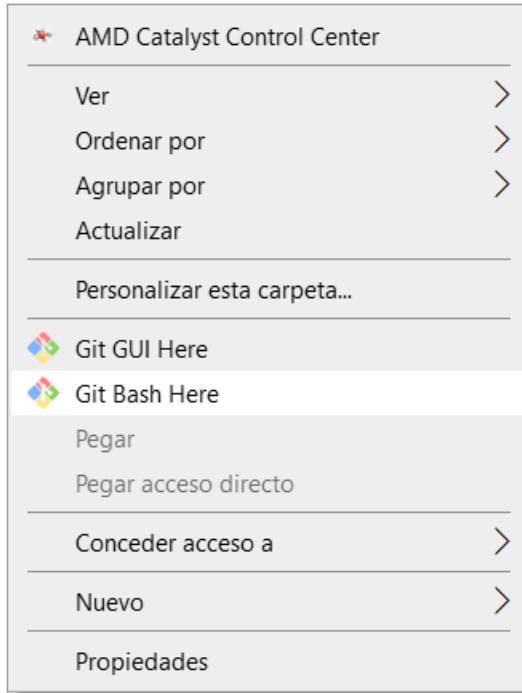
NOTA: Las variables se separan unas de otras mediante punto y coma:
variableuno;variabledos;variable-n;variable-n+1



Creación del clon de App Inventor en nuestro ordenador.

Vamos a crear en nuestro servidor (PC) una copia “clon” de App Inventor, lo bajaremos directamente de Internet y se creará esa copia.

Para esto nos apoyaremos de la aplicación **Git Bash** damos click para abrir una terminal.



Ejecutamos el comando en la terminal de Git Bash:

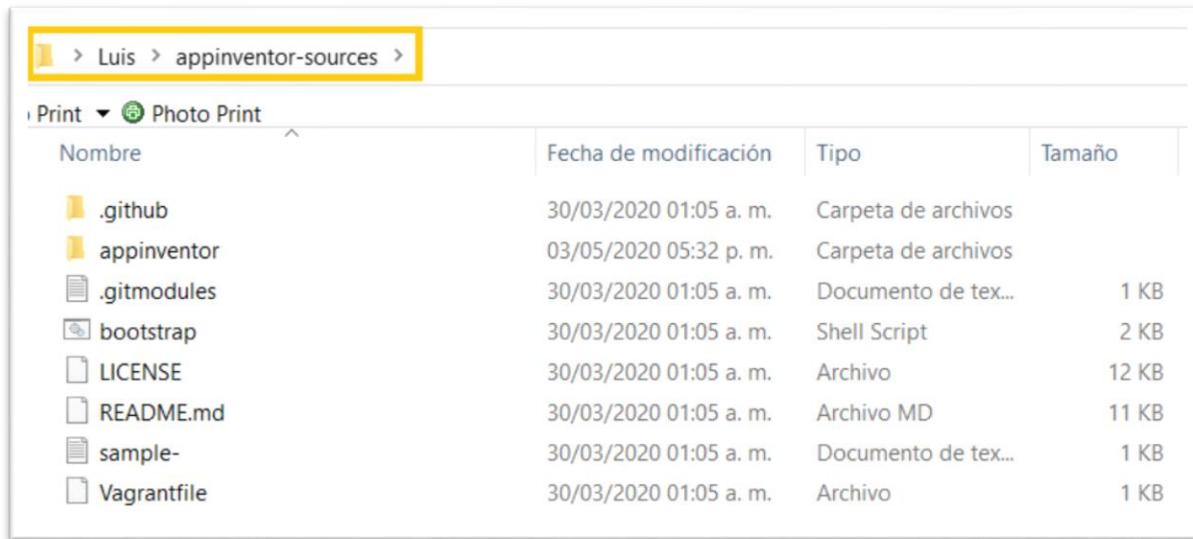
```
$ git clone https://github.com/mit-cml/appinventor-sources.git
```

```
uis@DESKTOP-LLGPLR6 MINGW64 ~
git clone https://github.com/mit-cml/appinventor-sources.git
Cloning into 'appinventor-sources'...
remote: Counting objects: 41191, done.
remote: Total 41191 (delta 0), reused 0 (delta 0), pack-reused 41190
receiving objects: 100% (41191/41191), 553.30 MiB | 494.00 KiB/s, done.
resolving deltas: 100% (21999/21999), done.
Checking out files: 100% (1758/1758), done.
uis@DESKTOP-LLGPLR6 MINGW64 ~
```

El sitio donde está el repositorio:

<https://github.com/mit-cml/appinventor-sources/>

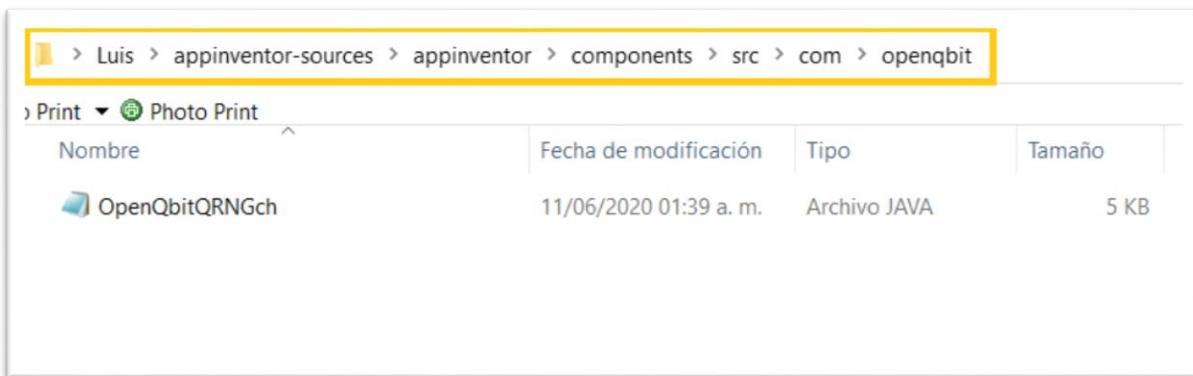
Creará una carpeta llamada appinventor-sources con la fuente de App Inventor.



Nombre	Fecha de modificación	Tipo	Tamaño
.github	30/03/2020 01:05 a. m.	Carpeta de archivos	
appinventor	03/05/2020 05:32 p. m.	Carpeta de archivos	
.gitmodules	30/03/2020 01:05 a. m.	Documento de tex...	1 KB
bootstrap	30/03/2020 01:05 a. m.	Shell Script	2 KB
LICENSE	30/03/2020 01:05 a. m.	Archivo	12 KB
README.md	30/03/2020 01:05 a. m.	Archivo MD	11 KB
sample-	30/03/2020 01:05 a. m.	Documento de tex...	1 KB
Vagrantfile	30/03/2020 01:05 a. m.	Archivo	1 KB

Creamos el siguiente directorio en la ruta (path) C:\Users\Luis\appinventor-sources\backup\appinventor\components\src\com\openqbit

Adentro copiamos nuestro programa de prueba siguiente llamado OpenQbitQRNGch.java



Nombre	Fecha de modificación	Tipo	Tamaño
OpenQbitQRNGch	11/06/2020 01:39 a. m.	Archivo JAVA	5 KB

Creación de la extensión.

RESPETA LAS MAYÚSCULAS Y MINÚSCULAS, no es lo mismo Hola que hola.

No pongas acentos. Ya estamos preparado para realizar nuestra primera extensión, será el un generador de números aleatorios cuánticos.

Usamos a un Editor de Texto, Notepad++, creamos un archivo llamado OpenQbitQRNGch.java que nos genera números cuánticos aleatorios con el siguiente código:

```
// Esta extensión se utiliza para sacar Quantum Random Number Generator
QRNG Switzerland API.

package com.openqbit.OpenQbitQRNGch;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.runtime.util.MediaUtil;
import com.google.appinventor.components.runtime.*;
import java.io.*;

@DesignerComponent(version = OpenQbitQRNGch.VERSION,
    description = "API Quantum Random Number Generator. Quantum random
numbers as a service, QRNGaaS, web API for the quantum random number
generator of Quantis developed by the Swiss company - " + "Guillermo
Vidal - OpenQbit.com",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "http://www.pinntar.com/logoQbit.png")
@SimpleObject(external = true)

public class OpenQbitQRNGch extends AndroidNonvisibleComponent implements
Component {

    public static final int VERSION = 1;
    public static final String DEFAULT_TEXTO_1 = "";
    private ComponentContainer container;
    private String texto_1 = "";

    public OpenQbitQRNGch(ComponentContainer container) {
        super(container.$form());
        this.container = container;
    }

    // Establecimiento de las Propiedades.
    //@DesignerProperty(editorType =
    PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue =
    OpenQbitQRNGch.DEFAULT_TEXTO_1 + "")
    //@SimpleProperty(description = "API Get Quantum Random Number Generator,
    random number between 0 and 1. Enter variable *quantity* of numbers to
    generate")

    @SimpleFunction(description = "API Get Quantum Random Number Generator,
    random number between 0 and 1. Enter variable integer number *quantity*
    of numbers to generate")
    public String APIGetQRNGdecimal(String qty) throws Exception {

```

```
String url = "http://random.openqu.org/api/rand?size=" + qty;
String[] command = { "curl", url };

ProcessBuilder process = new ProcessBuilder(command);
Process p;
p = process.start();
BufferedReader reader = new BufferedReader(new
InputStreamReader(p.getInputStream()));
StringBuilder builder = new StringBuilder();
String line = null;
while ((line = reader.readLine()) != null) {
    builder.append(line);

    builder.append(System.getProperty("line.separator"));
}

String result = builder.toString();
//System.out.print(result);
return result;

}

@SimpleFunction(description = "API Get Quantum Random Number Generator,
random number between a range min to max numbers. Enter variable integer
number *quantity* of numbers to generate a range of minimum *min* number
and maximum *max* number")
public String APIGetQRNGinteger(String qty, String min, String max)
throws Exception {
    String url = "http://random.openqu.org/api/randint?size=" + qty +
"&min=" + min + "&max=" + max;
    String[] command = { "curl", url };

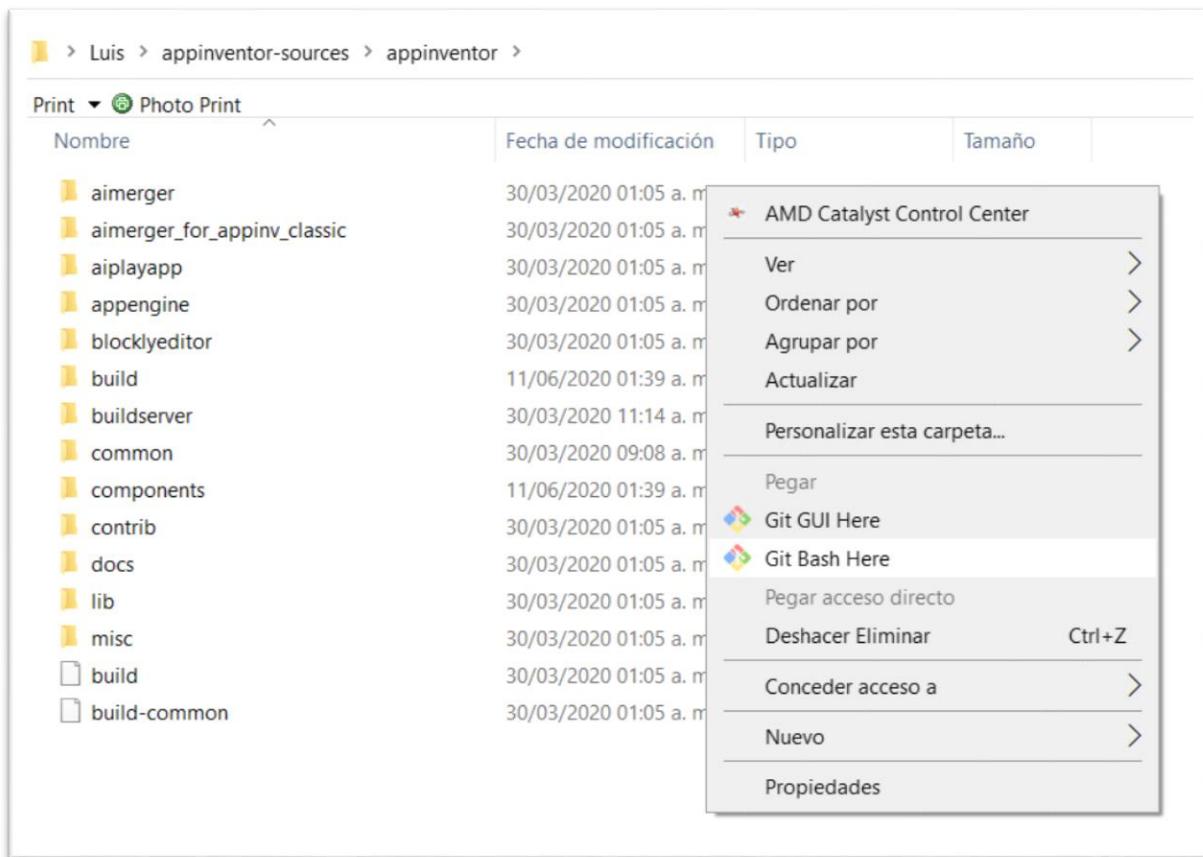
    ProcessBuilder process = new ProcessBuilder(command);
    Process p;
    p = process.start();
    BufferedReader reader = new BufferedReader(new
InputStreamReader(p.getInputStream()));
    StringBuilder builder = new StringBuilder();
    String line = null;
    while ((line = reader.readLine()) != null) {
        builder.append(line);

        builder.append(System.getProperty("line.separator"));
    }

    String result = builder.toString();
    //System.out.print(result);
    return result;

}
}
```

Ejecutamos el **Git Bash** posicionándonos en la siguiente ruta (path):
C:\Users\Luis\appinventor-sources\appinventor. En este directorio damos click botón izquierdo del ratón y seleccionamos la terminal de **Git Bash**.



La terminal de Git bash se posicionará en el siguiente directorio:

C:\Users\Luis\appinventor-sources\appinventor (master)

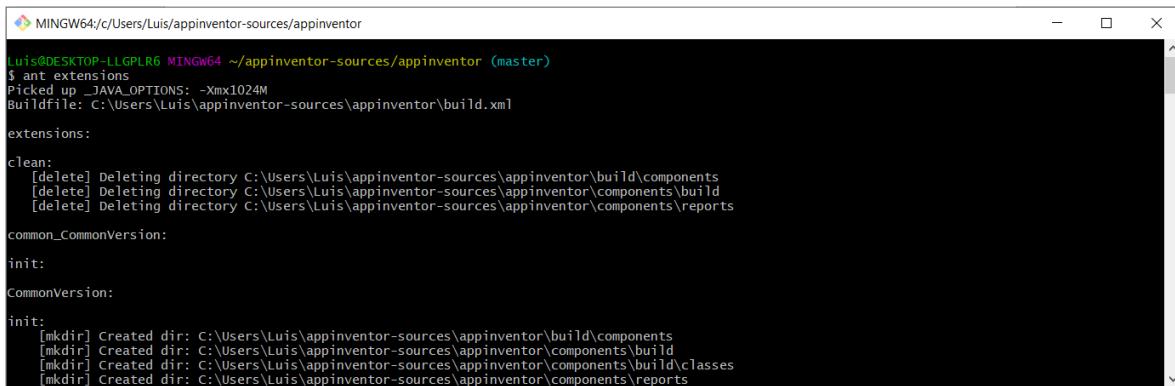
```

MINGW64:/c/Users/Luis/appinventor-sources/appinventor
Luis@DESKTOP-LLGPLR6 MINGW64 ~ /appinventor-sources/appinventor (master)
$ |

```

En la terminal de Git Bash y ejecutamos el siguiente comando:

\$ ant extensions



```
MINGW64:/c/Users/Luis/appinventor-sources/appinventor
Luis@DESKTOP-LLGPLR6 MINGW64 ~/appinventor-sources/appinventor (master)
$ ant extensions
Picked up _JAVA_OPTIONS: -Xmx1024M
Buildfile: C:\Users\Luis\appinventor-sources\appinventor\build.xml

extensions:
clean:
[delete] Deleting directory C:\Users\Luis\appinventor-sources\appinventor\build\components
[delete] Deleting directory C:\Users\Luis\appinventor-sources\appinventor\components\build
[delete] Deleting directory C:\Users\Luis\appinventor-sources\appinventor\components\reports

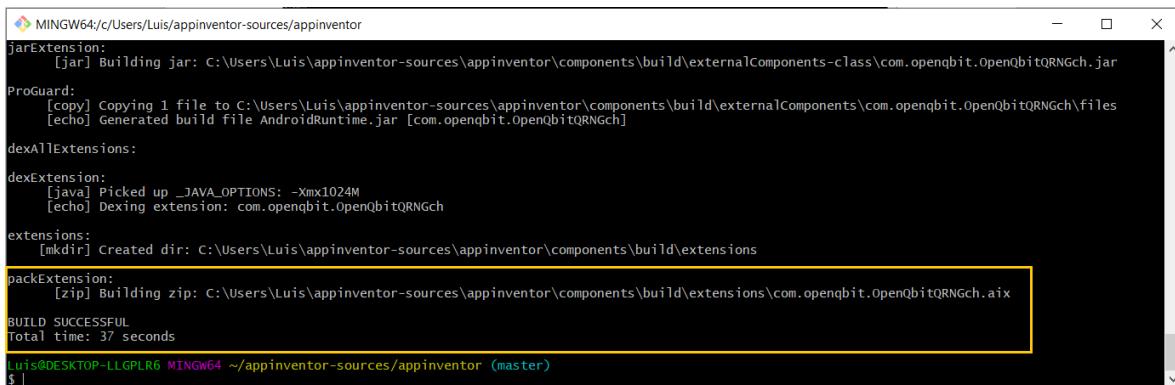
common_CommonVersion:
init:
CommonVersion:
init:
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\build\components
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\build
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\build\classes
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\reports
```

Si todo ha ido bien obtendremos: BUILD SUCESSFUL.

Se ha creado nuestra extensión en...

C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions

NOTA: el contenido de la carpeta extensions, se borra y se vuelve a crear cada vez que hacemos un ant extensions.



```
MINGW64:/c/Users/Luis/appinventor-sources/appinventor
jarExtension:
[jar] Building jar: C:\Users\Luis\appinventor-sources\appinventor\components\build\externalComponents-class\com.openqbit.OpenQbitQRNGch.jar

ProGuard:
[copy] Copying 1 file to C:\Users\Luis\appinventor-sources\appinventor\components\build\externalComponents\com.openqbit.OpenQbitQRNGch\files
[echo] Generated build file AndroidRuntime.jar [com.openqbit.OpenQbitQRNGch]

dexAllExtensions:

dexExtension:
[java] Picked up _JAVA_OPTIONS: -Xmx1024M
[echo] Dexing extension: com.openqbit.OpenQbitQRNGch

extensions:
[mkdir] Created dir: C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions

packExtension:
[zip] Building zip: C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions\com.openqbit.OpenQbitQRNGch.aix

BUILD SUCCESSFUL
Total time: 37 seconds
Luis@DESKTOP-LLGPLR6 MINGW64 ~/appinventor-sources/appinventor (master)
$ |
```

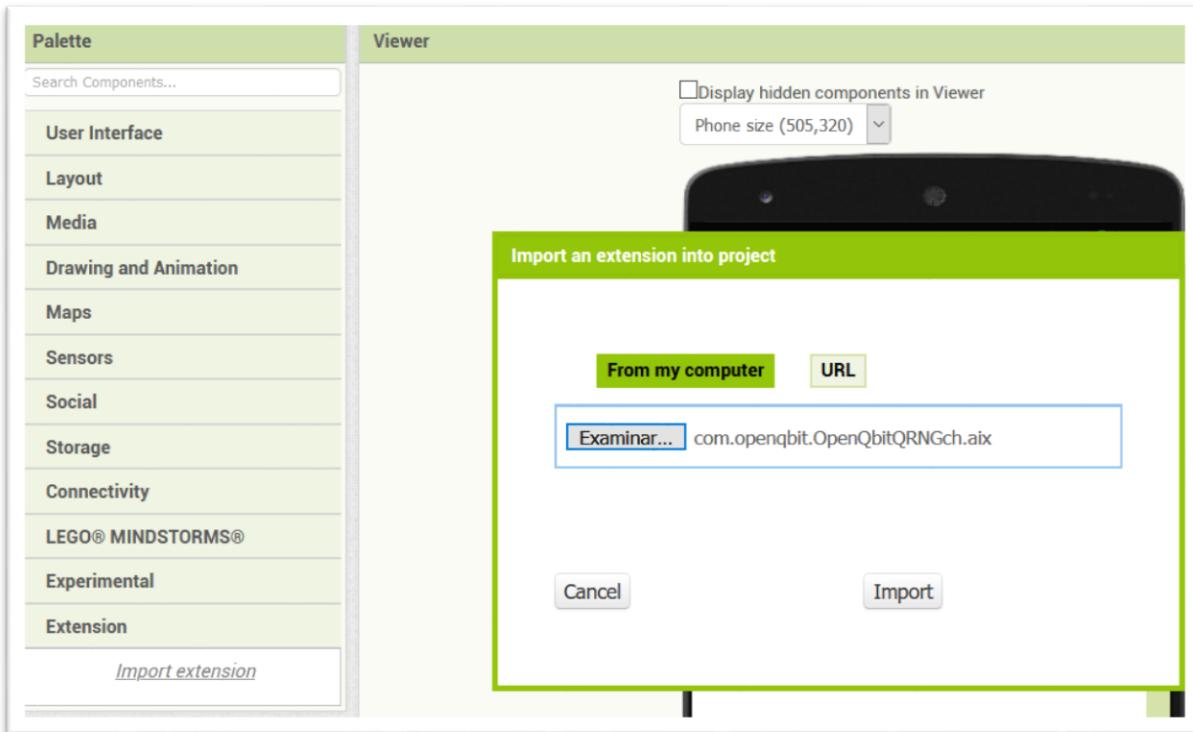
Vamos a esa carpeta:

C:\Users\Luis\appinventor-sources\appinventor\components\build\extensions

y copiamos el archivo com.openqbit.OpenQbitQRNGch.aix, este archivo es nuestra extensión.

Teniendo ya una cuenta en App Inventor u otro sistema de Blockly entramos para agregar la nueva extensión y probarla.

Vamos a la parte inferior en donde está la opción de Extensión y damos click en “Import extensión”:



Damos en el botón de “Import”. Ahora tendremos ya los bloques (**APIGetQRNGdecimal**) y (**APIGetQRNGinteger**) para ser usados:



Ya tenemos nuestra primer extensión creada y funcional.

27. Anexo “BlocklyCode Smart Contracts”.

Los BlocklyCode son programas hechos en lenguaje java. La extensión para crear este tipo de programas comúnmente llamados “Smart Contracts” o contratos inteligentes son una forma de procesar convenios entre usuarios (empresas o personas).

Esta el bloque (**BlocklyCode**), se implementa en un programa que ya tiene parámetros establecidos y que dependiendo del tipo de acuerdos que podrían ser ejecutados por el sistema de Mini BlocklyChain de forma automática al cumplirse las premisas que rigen el “Smart Contract”.

Los parámetros a considerar sugeridos o parámetros de entrada <inputsValue> son:

Fecha de vencimiento

Fecha referenciada

Tiempo de vencimiento

Tiempo referenciado

Activo referenciado

Activo total fijo

Activo variable

Miembros del contrato

Datos confirmados (String)

Datos confirmados (Filename)

Evento variable

Evento fijo

Validación de documento(s)

Validación de cadena (String)

Firma valida

Intervalo definido (Integer)

Intervalo decimal

Mínimo establecido

Máximo establecido

Antes de iniciar con el uso del bloque (**BlocklyCode**) primero tenemos que instalar el sistema que realizara la ejecución de los “Smart Contracts”, esto lo hacemos instalando en la terminal de Termux el OpenJDK y el OpenJRE.

OpenJDK (Open Java Development Kit). - Es la herramienta para desarrollar programas en java, contiene las librerías, compilador y la JVM (Java Virtual Machine).

OpenJRE (Open Java Runtime Environment). - Es la herramienta únicamente para ejecutar programas en java. Contiene la JVM (Java Virtual Machine).

Procedemos con la instalación del OpenJRE y OpenJDK en la terminal Termux de los nodos que forman el sistema Mini BlocklyChain.

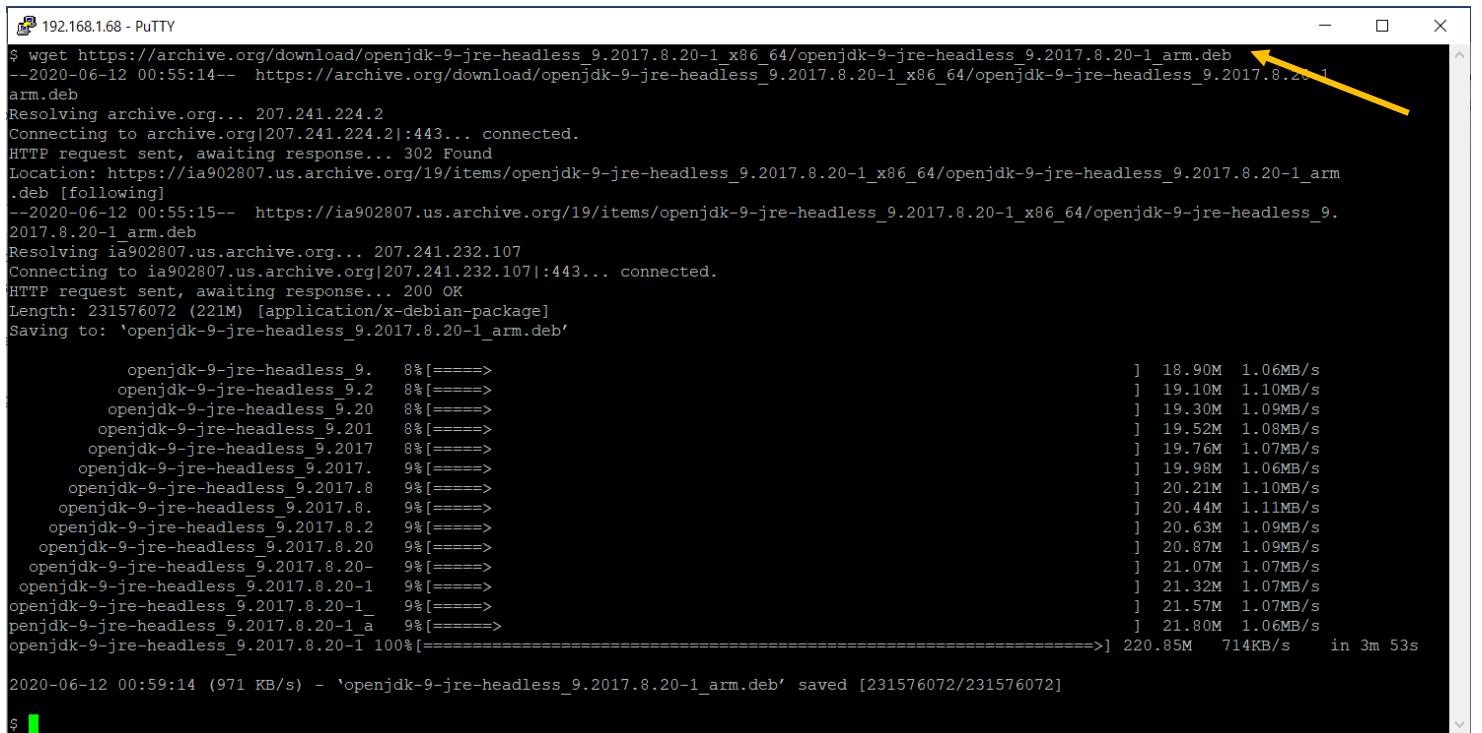
Instalamos las dependencias

\$ apt install -y wget



Bajamos el OpenJRE

\$ wget https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb



```
192.168.1.68 - PuTTY
$ wget https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
--2020-06-12 00:55:14-- https://archive.org/download/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Resolving archive.org... 207.241.224.2
Connecting to archive.org[207.241.224.2]:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://ia902807.us.archive.org/19/items/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb [following]
--2020-06-12 00:55:15-- https://ia902807.us.archive.org/19/items/openjdk-9-jre-headless_9.2017.8.20-1_x86_64/openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Resolving ia902807.us.archive.org... 207.241.232.107
Connecting to ia902807.us.archive.org[207.241.232.107]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 231576072 (221M) [application/x-debian-package]
Saving to: 'openjdk-9-jre-headless_9.2017.8.20-1_arm.deb'

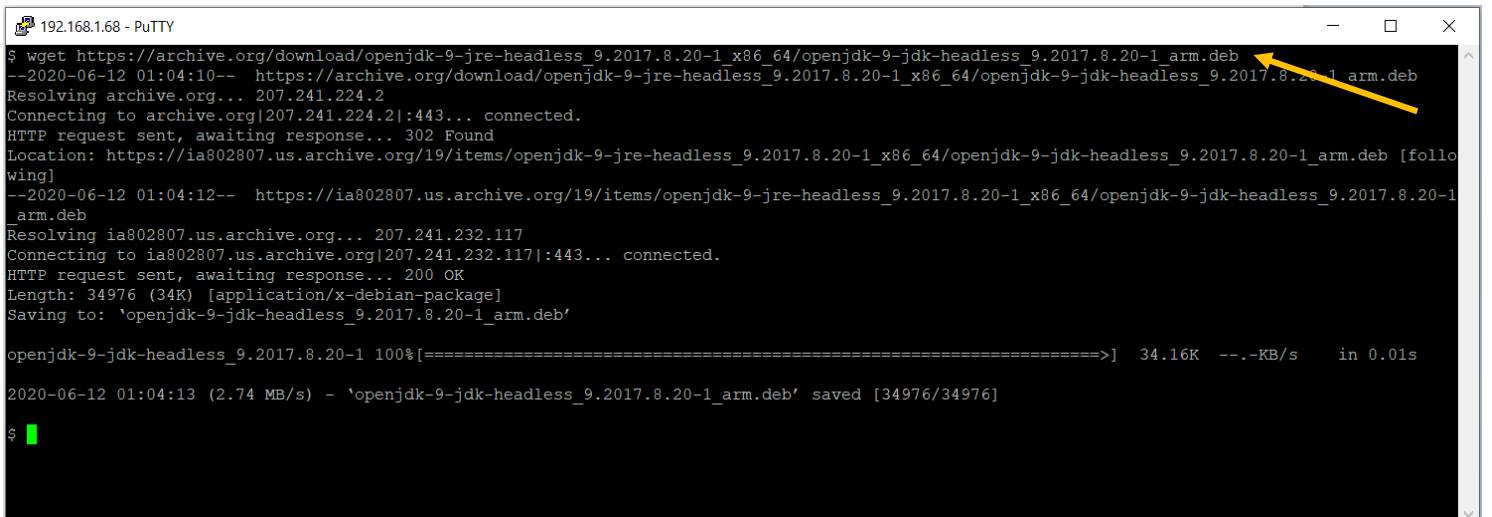
openjdk-9-jre-headless_9. 8%[=====] 18.90M 1.06MB/s
openjdk-9-jre-headless_9.2 8%[=====] 19.10M 1.10MB/s
openjdk-9-jre-headless_9.20 8%[=====] 19.30M 1.09MB/s
openjdk-9-jre-headless_9.201 8%[=====] 19.52M 1.08MB/s
openjdk-9-jre-headless_9.2017 8%[=====] 19.76M 1.07MB/s
openjdk-9-jre-headless_9.2017. 9%[=====] 19.98M 1.06MB/s
openjdk-9-jre-headless_9.2017.8 9%[=====] 20.21M 1.10MB/s
openjdk-9-jre-headless_9.2017.8. 9%[=====] 20.44M 1.11MB/s
openjdk-9-jre-headless_9.2017.8.2 9%[=====] 20.63M 1.09MB/s
openjdk-9-jre-headless_9.2017.8.20 9%[=====] 20.87M 1.09MB/s
openjdk-9-jre-headless_9.2017.8.20- 9%[=====] 21.07M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1 9%[=====] 21.32M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1_ 9%[=====] 21.57M 1.07MB/s
openjdk-9-jre-headless_9.2017.8.20-1_a 9%[=====] 21.80M 1.06MB/s
openjdk-9-jre-headless_9.2017.8.20-1 100%[=====] 220.85M 714KB/s in 3m 53s

2020-06-12 00:59:14 (971 KB/s) - 'openjdk-9-jre-headless_9.2017.8.20-1_arm.deb' saved [231576072/231576072]

$
```

Bajamos el OpenJDK

\$ wget https://archive.org/download/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb



```
192.168.1.68 - PuTTY
$ wget https://archive.org/download/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
--2020-06-12 01:04:10-- https://archive.org/download/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
Resolving archive.org... 207.241.224.2
Connecting to archive.org[207.241.224.2]:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://ia802807.us.archive.org/19/items/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb [following]
--2020-06-12 01:04:12-- https://ia802807.us.archive.org/19/items/openjdk-9-jdk-headless_9.2017.8.20-1_x86_64/openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
Resolving ia802807.us.archive.org... 207.241.232.117
Connecting to ia802807.us.archive.org[207.241.232.117]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 34976 (34K) [application/x-debian-package]
Saving to: 'openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb'

openjdk-9-jdk-headless_9.2017.8.20-1 100%[=====] 34.16K --.-KB/s in 0.01s

2020-06-12 01:04:13 (2.74 MB/s) - 'openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb' saved [34976/34976]

$
```

Realizamos la instalación desde la terminal Termux de OpenJDK y OpenJRE:

```
$ apt install -y ./openjdk-9-jre-headless_9.2017.8.20-1_arm.deb ./openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
```



```
$ ls
openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb
openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
$ apt install -y ./openjdk-9-jdk-headless_9.2017.8.20-1_arm.deb ./openjdk-9-jre-headless_9.2017.8.20-1_arm.deb
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'openjdk-9-jdk-headless' instead
of './openjdk-9-jdk-headless_9.2017.8.20-1_arm.
deb'
Note, selecting 'openjdk-9-jre-headless' instead
of './openjdk-9-jre-headless_9.2017.8.20-1_arm.
deb'
The following additional packages will be instal
led:
  ca-certificates-java freetype libpng
The following NEW packages will be installed:
  ca-certificates-java freetype libpng
  openjdk-9-jdk-headless
  openjdk-9-jre-headless
0 upgraded, 5 newly installed, 0 to remove and 0
not upgraded.
Need to get 668 kB/232 MB of archives.
After this operation, 376 MB of additional disk
space will be used.
Get:1 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm ca-certificates-java all
20200101 [110 kB]
Get:2 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm libpng arm 1.6.37-2 [190
kB]
Get:3 https://dl.bintray.com/termux/termux-pac
ges-24 stable/main arm freetype arm 2.10.2 [368
kB]
Get:4 /data/data/com.termux/files/home/openjdk/o
penjdk-9-jre-headless_9.2017.8.20-1_arm.deb open
jdk-9-jre-headless arm 9.2017.8.20-1 [232 MB]
Get:5 /data/data/com.termux/files/home/openjdk/o
penjdk-9-jdk-headless_9.2017.8.20-1_arm.deb open
jdk-9-jdk-headless arm 9.2017.8.20-1 [35.0 kB]

ESC   ⏪   CTRL   ALT   -   ↓   ↑
      □   ◀   ○   □
```



```
Get:5 /data/data/com.termux/files/home/openjdk/o
penjdk-9-jdk-headless_9.2017.8.20-1_arm.deb open
jdk-9-jdk-headless arm 9.2017.8.20-1 [35.0 kB]
Fetched 668 kB in 23s (28.0 kB/s)
Selecting previously unselected package ca-certifi
cates-java.
(Reading database ... 16939 files and directo
ries currently installed.)
Preparing to unpack .../ca-certificates-java_202
00101_all.deb ...
Unpacking ca-certificates-java (20200101) ...
Selecting previously unselected package libpng.
Preparing to unpack .../libpng_1.6.37-2_arm.deb
...
Unpacking libpng (1.6.37-2) ...
Selecting previously unselected package freetype
.
Preparing to unpack .../freetype_2.10.2_arm.deb
...
Unpacking freetype (2.10.2) ...
Selecting previously unselected package openjdk-
9-jre-headless.
Preparing to unpack .../openjdk-9-jre-headless_9
.2017.8.20-1_arm.deb ...
Unpacking openjdk-9-jre-headless (9.2017.8.20-1)
...
Selecting previously unselected package openjdk-
9-jdk-headless.
Preparing to unpack .../openjdk-9-jdk-headless_9
.2017.8.20-1_arm.deb ...
Unpacking openjdk-9-jdk-headless (9.2017.8.20-1)
...
Setting up libpng (1.6.37-2) ...
Setting up freetype (2.10.2) ...
Setting up ca-certificates-java (20200101) ...
Setting up openjdk-9-jre-headless (9.2017.8.20-1
) ...
Setting up openjdk-9-jdk-headless (9.2017.8.20-1
) ...
$ █

ESC   ⏪   CTRL   ALT   -   ↓   ↑
      □   ◀   ○   □
```

Ya que hemos finalizado la instalación del ambiente OpenJDK y OpenJRE. Estas instalaciones contienen la JVM (Java Virtual Machine) que será el ambiente que ejecutara el BlocklyCode.

Ahora instalaremos un editor para crear archivo en línea, usaremos el editor llamado **vi** ejecutamos el siguiente comando:

```
$ apt install vim
```

Ahora probemos compilar un archivo de prueba y ejecutarlo. Creamos en la terminal de Termux con el editor **vi** y escribimos el código en java de un “Hello World”, y lo salvamos en el archivo HelloWorld.java

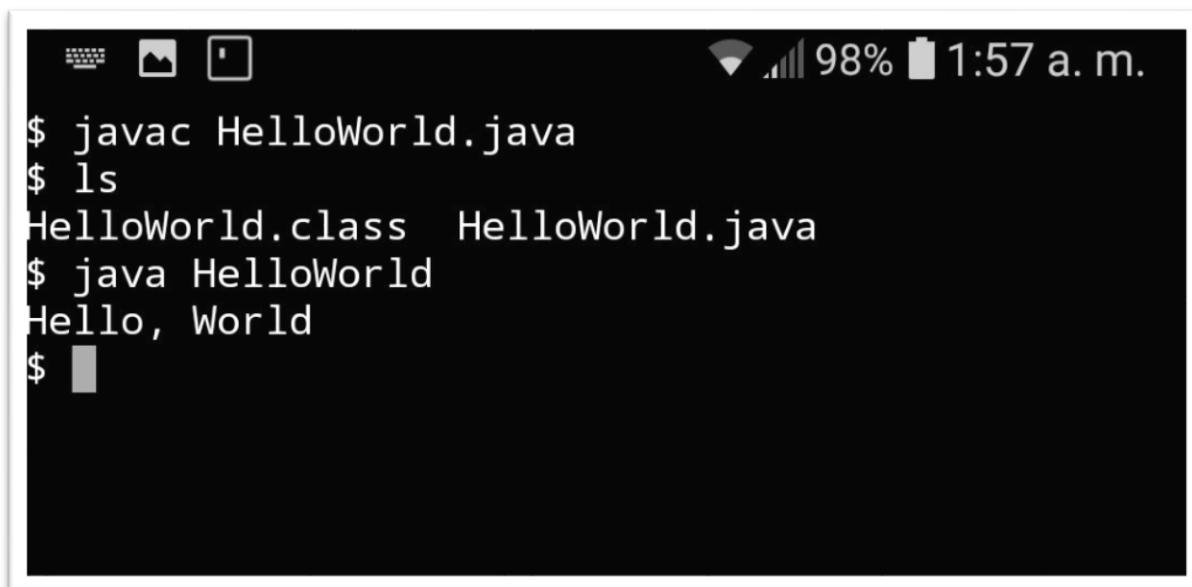
```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        // Prints "Hello, World" to the terminal window.  
        System.out.println("Hello, World");  
    }  
  
}
```

Después ejecutamos en la terminal de Termux el siguiente comando para la compilación y después la ejecución del programa:

```
$ javac HelloWorld.java
```

 (Después de ejecutar se crea un archivo bytecode HelloWorld.class)

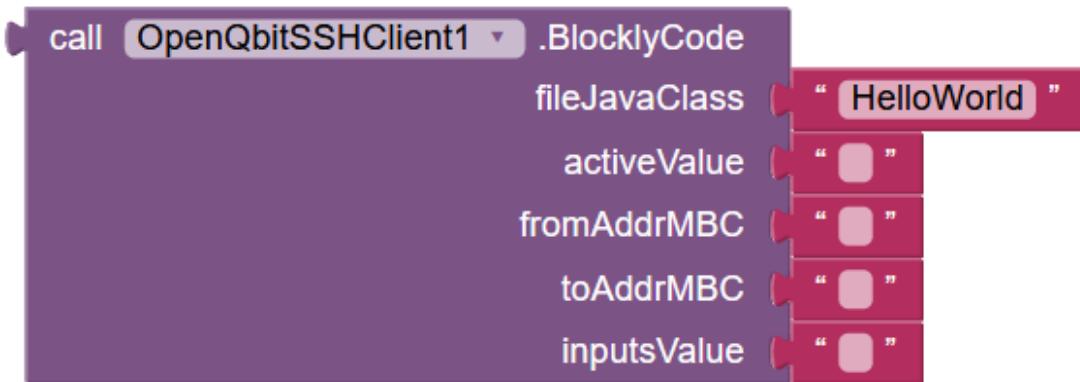
```
$ java HelloWorld
```

 (Después de ejecutar e imprime el anuncio, Hello World)

The screenshot shows a Termux terminal window on a mobile device. The status bar at the top indicates signal strength, battery level (98%), and time (1:57 a.m.). The terminal window displays the following command-line session:

```
$ javac HelloWorld.java  
$ ls  
HelloWorld.class  HelloWorld.java  
$ java HelloWorld  
Hello, World  
$
```

Uso del bloque (**BlocklyCode**) este bloque se encuentra en la extensión (**ConectorSSHClient**).



En el anterior bloque se puede ver como se ejecutará el archivo HelloWorld.class que ya ha sido compilado y posicionado en el directorio base de usuario.

Es un bloque en el cual se puede realizar la ejecución de un programa ya compilado en java, lo que comúnmente en el ambiente de desarrollo se conoce como el archivo bytecode en su forma binario.

Este tipo de archivo ya esta listo para ser ejecutado por la Java Virtual Machine (JVM).

Hay dos formas de crear un archivo bytecode o con extensión .CLASS, el usuario puede crearlo en su PC por comodida o su gusto también se puede realizar en el telefono móvil recordemos que ya hemos instalado el ambiente para compilar JDK (Java Development Kit) con anterioridad, aunque sera menos eficiente ya que la limitación del teclado contara, también en caso de elegir el ambiente de Android se tendrá que tener en cuenta que las librerías están limitadas a que tiene OpenJDK que se haya instalado.

Los parámetros de entrada están abiertos en <inputsValue> y los otros fijos son los de activeValue, fromaddrMBC y toAddrMBC.

En el caso de realizar pruebas de ejecución podrán dejarse en blanco sin contenido y únicamente se ejecutará el archivo bytecode sin parámetros.

El anterior bloque es como si se ejecutara la siguiente línea de comando:

```
$ java HelloWorld
```

Los programas desarrollados para BlocklyCode estarán sujetos a cada entorno particular de diseño y se podrán contralar y ejecutar mediante rutina con el agente “cron” y compartir con syncthingmanager.

28. Anexo “Computación Cuántica con OpenQbit”.

¿Cómo funciona la computación cuántica? ⁽²⁾

La transformación digital está generando cambios en el mundo más rápido que nunca. ¿Creerías que la era digital está por acabarse? Ya se ha señalado la **alfabetización digital** como un área donde el conocimiento abierto y las oportunidades accesibles para aprender sobre la tecnología son urgentes para abordar las brechas en el desarrollo social y económico. Aprender de los conceptos claves de la era digital se volverá aún más crítico ante la inminente llegada de otra nueva ola tecnológica capaz de transformar los modelos existentes con una velocidad y potencia asombrosa: las **tecnologías cuánticas**.

En este artículo, comparamos los conceptos básicos de la computación tradicional y la computación cuántica; y también comenzamos a explorar su aplicación en otras áreas relacionadas.

¿Qué son las tecnologías cuánticas?

A lo largo de la historia, el ser humano ha ido desarrollando tecnología a medida que ha ido entendiendo el funcionamiento de la naturaleza a través de la ciencia. Entre los años 1900 y 1930, el estudio de algunos fenómenos físicos que aún no estaban bien entendidos dio lugar a una nueva teoría física, la **Mecánica Cuántica**. Esta teoría describe y explica el funcionamiento del mundo microscópico, hábitat natural de moléculas, átomos o electrones. Gracias a ella no solo se ha conseguido explicar esos fenómenos, sino que ha sido posible entender que la realidad subatómica funciona de forma completamente contra intuitiva, casi mágica, y que en el mundo microscópico tienen lugar sucesos que no ocurren en el mundo macroscópico.

Entre estas **propiedades cuánticas** se incluyen la superposición cuántica, el entrelazamiento cuántico y el teletransporte cuántico.

- La **superposición cuántica** describe cómo una partícula puede estar en diferentes estados a la vez.
- El **entrelazamiento cuántico** describe cómo dos partículas tan separadas como se desee pueden estar correlacionadas de forma que, al interactuar con una, la otra se entera.
- El **teletransporte cuántico** utiliza el entrelazamiento cuántico para enviar información de un lugar a otro del espacio sin necesidad de viajar a través de él.

Las tecnologías cuánticas son basadas en estas propiedades cuánticas de la naturaleza subatómica.

En este caso, hoy en día el entendimiento del mundo microscópico a través de la Mecánica Cuántica nos permite inventar y diseñar tecnologías capaces de mejorar la vida de las OpenQbit.com

personas. Hay muchas y muy diferentes tecnologías que utilizan fenómenos cuánticos y, algunas de ellas, como el láser o las imágenes por resonancia magnética (IRM), llevan ya entre nosotros más de medio siglo. Sin embargo, actualmente estamos presenciando una revolución tecnológica en áreas como la computación cuántica, la información cuántica, la simulación cuántica, la óptica cuántica, la metrología cuántica, los relojes cuánticos o los sensores cuánticos.

¿Qué es la computación cuántica? Primero, hay que entender la computación clásica.



Carácter	Bits
?	111
A	01000001
\$	00100100
:)	0011101000101001

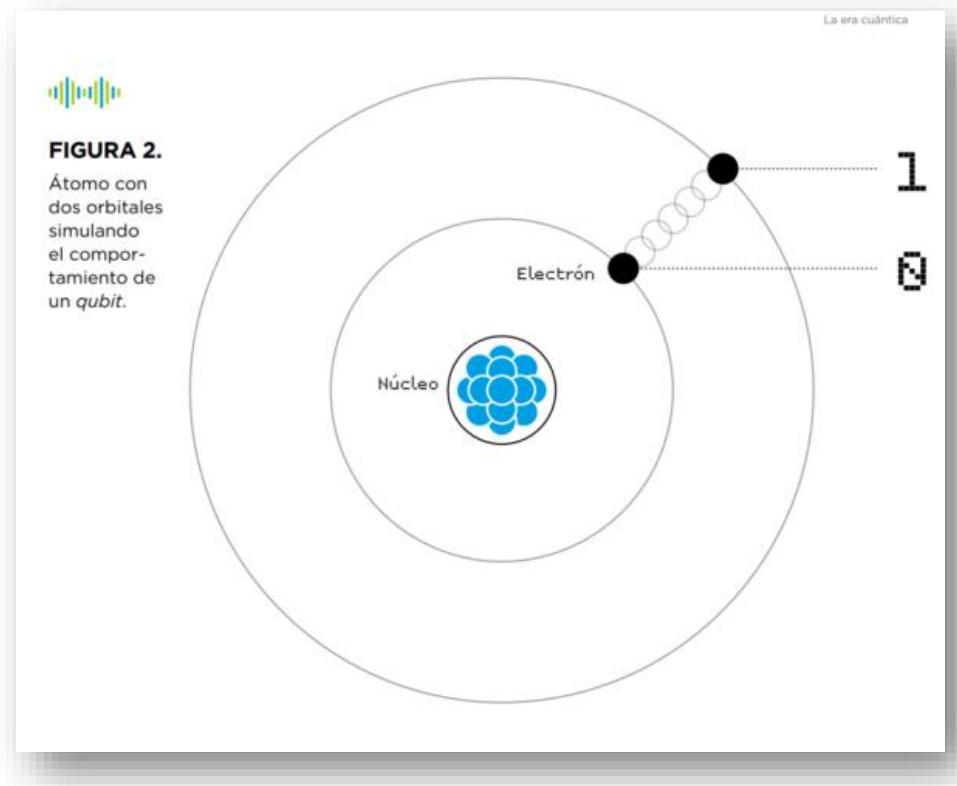
Para entender cómo funcionan los computadores cuánticos es conveniente explicar primero cómo funcionan los computadores que utilizamos a diario, a los que nos referiremos en este documento como computadores digitales o clásicos. Estos, al igual que el resto de los dispositivos electrónicos como tabletas o teléfonos móviles, utilizan bits como unidades fundamentales de memoria. Esto significa que los programas y aplicaciones están codificados en bits, es decir, en lenguaje binario de ceros y unos. Cada vez que interactuamos con cualquiera de estos dispositivos, por ejemplo, pulsando una tecla del teclado, se crean, destruyen y/o modifican cadenas de ceros y unos dentro de la computadora.

La pregunta interesante es, ¿qué son físicamente estos ceros y unos dentro de la computadora? Los estados cero y uno de los bits se corresponden con corriente eléctrica que circula, o no, a través de unas piezas microscópicas denominadas transistores, que actúan como interruptores. Cuando no circula corriente, el transistor está “apagado” y se corresponde con un bit 0, y cuando circula está “encendido” y se corresponde con un bit 1.

De forma más simplificada, es como si los bits 0 y 1 se correspondiesen con huecos, de manera que un hueco vacío es un bit 0 y un hueco ocupado por un electrón es un bit 1. Es por este motivo que estos dispositivos se llaman electrónicos. A modo de ejemplo, en la figura 1 se muestra la escritura en lenguaje binario de algunos caracteres. Ahora que tenemos una idea de cómo funcionan los computadores actuales, trataremos de entender cómo funcionan los cuánticos.

De los bits a qubits

La unidad fundamental de información en computación cuántica es el quantum bit o qubit. Los qubits son, por definición, sistemas cuánticos de dos niveles -ahora veremos ejemplos- que al igual que los bits pueden estar en el nivel bajo, que se corresponde con un estado de baja excitación o energía definido como 0, o en el nivel alto, que se corresponde con un estado de mayor excitación o definido como 1. Sin embargo, y aquí radica la diferencia fundamental con la computación clásica, los qubits también pueden estar en cualquiera de los infinitos estados intermedios entre el 0 y el 1, como por ejemplo un estado que sea mitad 0 y mitad 1, o tres cuartos de 0 y un cuarto de 1. Este fenómeno se conoce como superposición cuántica y es natural en sistemas cuánticos.



Algoritmos cuánticos, computación exponencialmente más poderosa y eficiente

El propósito de los computadores cuánticos es aprovechar estas propiedades cuánticas de los *qubits*, como sistemas cuánticos que son, para poder correr algoritmos cuánticos que utilizan la superposición y el entrelazamiento para ofrecer una capacidad de procesamiento mucho mayor que los clásicos. Es importante indicar que el verdadero cambio de paradigma no consiste en hacer lo mismo que hacen las computadoras digitales o clásicas -las actuales-, pero más rápido, como de forma errónea se puede leer en muchos artículos, sino que los algoritmos cuánticos permiten realizar ciertas operaciones de una manera totalmente

diferente que en muchos casos resulta ser más eficiente -es decir, en mucho menos tiempo o utilizando muchos menos recursos computacionales-.

Veamos un ejemplo concreto de lo que esto implica. Imaginemos que estamos en Bogotá y queremos saber cuál es la mejor ruta para llegar a Lima de entre un millón de opciones para llegar ($N=1.000.000$). De cara a poder utilizar computadoras para encontrar el camino óptimo necesitamos digitalizar 1.000.000 opciones, lo que implica traducirlas a lenguaje de bits para el computador clásico y a *qubits* para el computador cuántico. Mientras que una computadora clásica necesitaría ir uno por uno analizando todos los caminos hasta encontrar el deseado, una computadora cuántica se aprovecha del proceso conocido como paralelismo cuántico que le permite considerar todos los caminos a la vez. Esto implica que, si bien la computadora clásica necesita del orden de $N/2$ pasos o iteraciones, es decir, 500.000 intentos, la computadora cuántica encontrará la ruta óptima tras solo \sqrt{N} operaciones sobre el registro, es decir, 1.000 intentos.

En el caso anterior la ventaja es cuadrática, pero en otros casos es incluso exponencial, lo que significa que con n *qubits* podemos obtener una capacidad computacional equivalente a 2^n bits. Para exemplificar esto, es frecuente contar que con unos 270 qubits se podrían tener más estados base en un computador cuántico -más cadenas de caracteres diferentes y simultáneas- que el número de átomos en el universo, que se estima en torno a 280. Otro ejemplo, es que se estima que con un computador cuántico de entre 2000 y 2500 *qubits* se podría romper prácticamente toda la criptografía utilizada hoy en día (la conocida como criptografía de clave pública).

¿Por qué es importante saber sobre la tecnología cuántica?

Estamos en un momento de transformación digital en el que distintas tecnologías emergentes como blockchain, inteligencia artificial, drones, Internet de las cosas, realidad virtual, 5G, impresoras 3D, robots o vehículos autónomos tienen cada vez más presencia en múltiples ámbitos y sectores. Estas tecnologías, llamadas a mejorar la calidad de vida del ser humano acelerando el desarrollo y generando impacto social, avanzan hoy en día de manera paralela. Solo en contadas ocasiones vemos compañías desarrollando productos que exploten combinaciones de dos o más de estas tecnologías, como blockchain y IoT o drones e inteligencia artificial. Si bien están destinadas a converger generando así un impacto exponencialmente mayor, la etapa inicial de desarrollo en que se encuentran y la escasez de desarrolladores y personas con perfiles técnicos hacen que las convergencias sean aún una tarea pendiente.

De las tecnologías cuánticas, por su potencial disruptivo, se espera que no solo converjan con todas estas nuevas tecnologías, sino que tengan una influencia transversal en prácticamente la totalidad de ellas. La computación cuántica amenazará la autenticación, intercambio y almacenamiento seguro de datos, teniendo un impacto mayor en aquellas tecnologías en las que la criptografía tiene un rol más relevante, como ciberseguridad o

blockchain, y un impacto negativo menor pero también a considerar en tecnologías como 5G, IoT o drones.

¿Quieres practicar la computación cuántica?

En la red hay ya disponibles decenas de simuladores de computadores cuánticos con diferentes lenguajes de programación ya existentes como C, C++, Java, Matlab, Máxima, Python o Octave. También, lenguajes nuevos como Q#, lanzado por Microsoft. Se puede explorar y jugar con una maquina cuántica virtual a través de plataformas como la de IBM y la de Rigetti.

Mini BlocklyChain es creado por OpenQbit.com compañía que se enfoca en desarrollar tecnología basada en computación cuántica para diferentes tipos de sectores tanto privado como público.

Por qué Mini BlocklyChain es diferente a los demás blockchain, simplemente porque el sistema fue creado para ser modular e incluir computación cuántica.

- (2) <https://blogs.iadb.org/conocimiento-abierto/es/como-funciona-la-computacion-cuantica/>

29. Anexo “Bloques extendidos para base de datos SQLite”.

Para ver más detalle del uso adecuado y puntual de cada bloque que se componen la extensión OpenQbitSQLite revisar el autor original de la extensión en la liga siguiente.

OpenQbitSQLite es un clone del diseño original con pequeñas modificaciones para ser usado con un nivel de seguridad AES.

<https://github.com/frdfsnlght/aix-SQLite>

30. Anexo “Ejemplo de creación de sistema Mini BlocklyChain”.

Realizaremos un sistema de prueba en donde podremos verificar las funcionalidades, ventajas y facilidad de crear un sistema Mini BlocklyChain.

Empezaremos por el diseño y desarrollo del almacenamiento en donde residirá la información lo que ya hemos definido como cadena de bloques. El diseño se basará en la base de datos SQLite y dependiendo de cada organización o diseño esta puede ser cambiada fácilmente por otra base de datos como pudiera ser Microsoft SQL Server, MySQL, Maria DB, Oracle, DB2, PostgreSQL entre otras. Aunque nuevamente debido al proceso de transacciones y concurrencia a la base SQLite se puede usar a cualquier nivel ya sea para empresa o aplicación personal.

Creacion de base de datos llamada **minibc** en esta tendrá una tabla que estará almacenando la cadena de bloques. Esta base estará embebida dentro del código del programa final que se instalara en los nodos, a este lugar de almacenamiento se denomina “assets packaged” es un especio interno en los ambientes Blockly como es el caso de App Inventor.

```
$ sqlite3 minibc.db
SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .quit
```

Diseño de campos de la tabla **nblock**.

```
CREATE TABLE nblock (
    id integer primary key AUTOINCREMENT
    , prevhash    VARCHAR NOT NULL
    , newhash    VARCHAR NOT NULL
    , ntrans      INTEGER NOT NULL
    ,nonce       INTEGER NOT NULL
    ,qrng        INTEGER NOT NULL
);

```

Creamos la tabla **nblock**.

```
$ sqlite3 minibc.db
SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open minibc.db
sqlite> CREATE TABLE nblock ( id integer primary key AUTOINCREMENT , prevhash VARCHAR NOT NULL , newhash VARCHAR NOT NULL, ntrans INTEGER NOT NULL ,nonce INTEGER NOT NULL ,qrng INTEGER NOT NULL);
```

Veremos algo similar a:

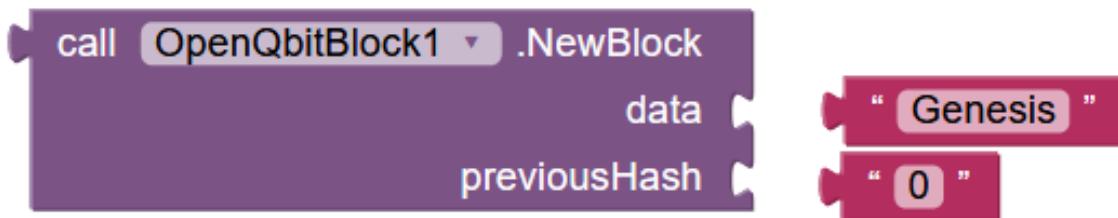


```
$ sqlite3 minibc.db
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
sqlite> CREATE TABLE nblock (
...>     id integer primary key AUTOINCREMENT
...>     , prevhash VARCHAR NOT NULL
...>     , newhash VARCHAR NOT NULL
...>     , ntrans INTEGER NOT NULL
...>     , nonce INTEGER NOT NULL
...>     , qrng INTEGER NOT NULL);
sqlite> .tables
nblock
sqlite> .quit
$
```

Sentencia SQL para la creación de tabla nblock.

Enseguida crearemos el hash inicial del sistema llamado “**Genesis**”, el cual está basado en crear un nuevo hash del primer bloque de la cadena de bloques usaremos el bloque (**NewBlock**). Después crearemos un segundo hash a este le llamaremos “uno” basado en el hash “Genesis” ya que este debe ser consistente con el primer hash debido a que la prueba de validez de hash en la cadena de bloques inicial siempre debe cumplir con: prevhash = newhash.

Bloque (NewBlock). Para crear el hash “Genesis” usaremos los parámetros de entrada:



Parámetro de entrada: **data** <String>, **previousHash** <String>

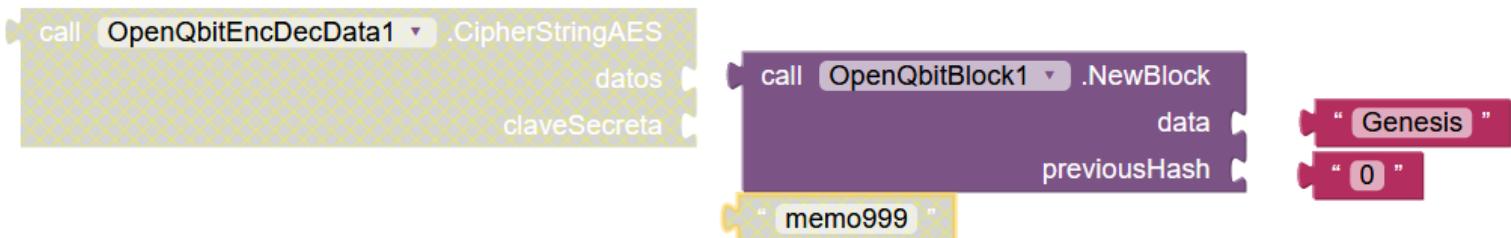
Parámetro de salida: Un Hash SHA256.

En este caso usaremos como “previousHash” inicial igual a “0” y en el caso de los datos de entrada “data” será la palabra “Genensis”.

Lo anterior nos dará un hash calculado de la combinación de ambos datos:

SHA256 (Genesis0) = eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642

La anterior cadena será insertada en la tabla nblock, esta cadena deberá estar cifrada para esto usamos la extensión (OpenQbitEncDecData).



Utilizaremos la extensión OpenQbitSSHClient para insertar los datos en la base de datos minibc.db, quedaría la sentencia de INSERT a la base de datos minibc.db de la tabla nblock.

```
sqlite3 keystore.db "insert into nblock (prevhash, newhash, ntrans, nonce, qrng) values ('0',
'eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642','1','0','0');"
```

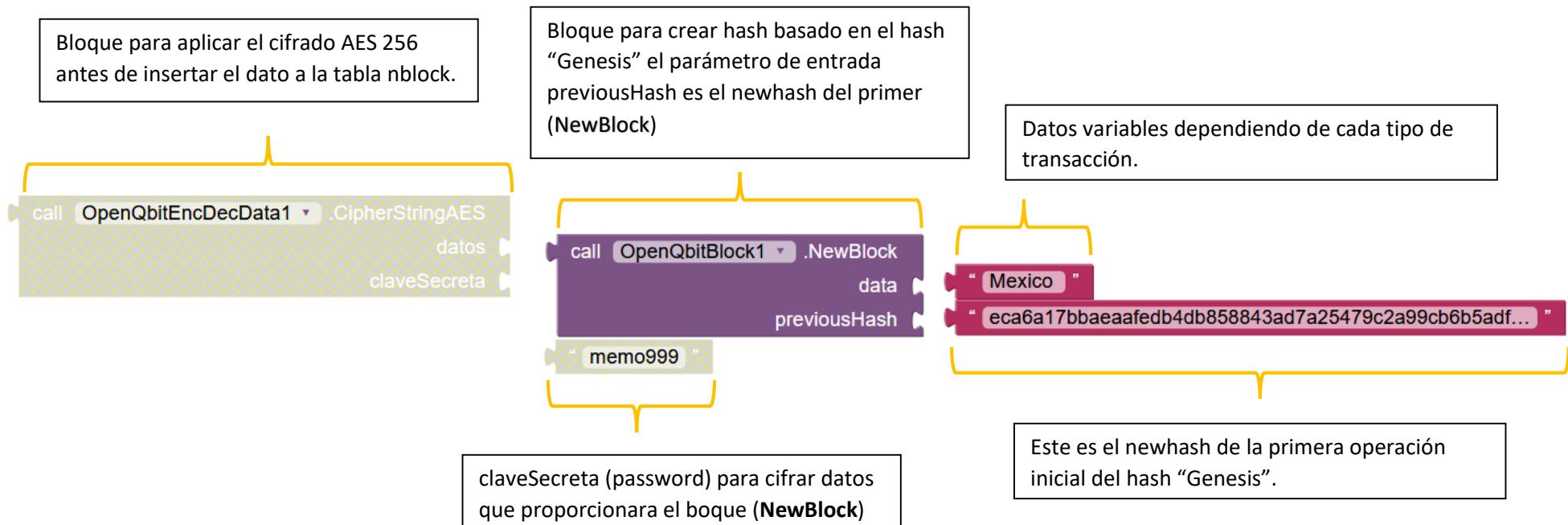
Creamos el hash “uno” con la sentencia SQL basada en el hash “Genensis”:

```
sqlite3 keystore.db "insert into nblock (prevhash, newhash, ntrans, nonce, qrng) values (
'eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642',
f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68','1','0','0');"
```

El newhash del hash “uno” se calcula como:

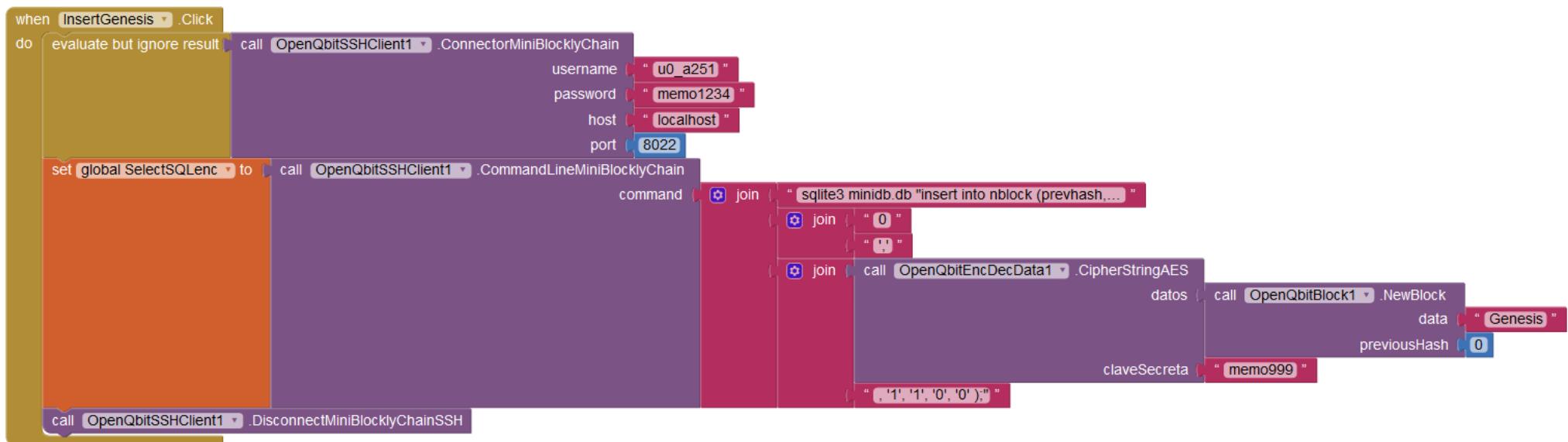
SHA256 (eca6a17bbaeaafedb4db858843ad7a25479c2a99cb6b5adfbb09ba54e802e642**Mexico**) =
f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68

Segundo hash basado en el primer hash “Genesis” de inicialización, debemos realizar dos INSERT en el inicio debido a que toda validación inicial de la cadena de bloques debe cumplir la igualdad de campos: prevhash (penúltimo dato) = newhash (último dato).



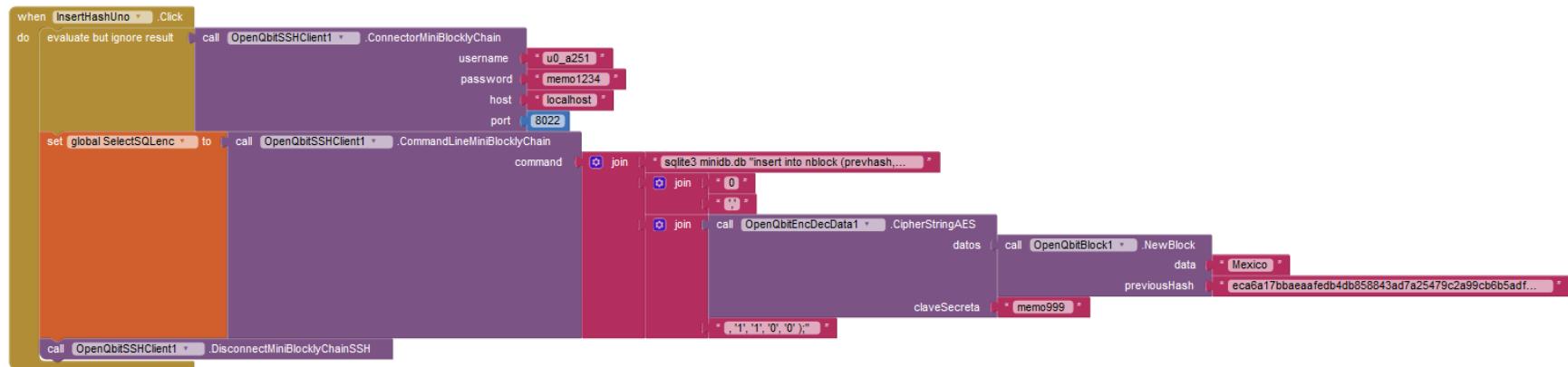
Proseguimos con la creación de la ejecución dentro de App Inventor de los hashes anteriormente; hash “Genesis” y hash “inicio”.

Ahora mostramos como se integraría del INSERT del hash “Genesis” en la estructura inicial de la cadena de bloques dentro de la tabla nblock en el sistema de App Inventor:



Ya que hemos insertado los primeros datos en la tabla nblock basados en el hash “Genesis”, procedemos a realizar el segundo INSERT referenciado al hash “uno”.

Integración del INSERT del hash “Uno” en la estructura inicial de la cadena de bloques dentro de la tabla nblock en el sistema de App Inventor:



Las dos anteriores estructuras de programación (hash “Genesis” y hash “uno”) deberían ser integradas por el nodo inicial del sistema de Mini BlocklyChain, un punto importante es que la base de datos minibc.db con toda su estructura (tablas) interna serán replicadas a través de la red de Mini BlocklyChain basada en una arquitectura “Peer to Peer” esto lo realizaremos a través de la herramienta de syncthing.

Hasta este momento hemos completado la estructura básica y fundamental del almacenamiento de la cadena de bloques y esta ya lista para recibir nuevos bloques de las futuras transacciones que se originen o soliciten en el sistema.

Ya hemos insertado el primer dato hash “Genesis” y el hash “uno” a nuestra base de datos **minibc.db**, ahora usaremos las siguientes sentencias SQL para realizar la consulta de los campos **prevhash** y **newhash** en la tabla **nblock**.

Este punto será fundamental ya que basado en la comparación de estos dos campos sabremos si la cadena de bloques está siendo consistente y si mantiene su integridad y seguridad de las transacciones. Este es solo un mecanismo de inicio para revisar la cadena de bloques ya que en el futuro revisaremos el uso del bloque para calcular el árbol de merkle que será otra herramienta esencial para asegurarnos también de la integridad y seguridad de la cadena de bloques en nuestro sistema.

Por ahora solo revisaremos las estructuras o sentencias SQL que debemos usar como lo hemos anteriormente con la extensión ([OpenQbitSSHClient](#)), con estas podremos consultar los campos prevhash y newhash. Después podremos hacer la comparación simple de igualdad de datos para dicha comprobación cada vez que vayamos a agregar un nuevo bloque.

Para prevhash:

```
sqlite3 minibc.db "SELECT prevhash FROM nblock ORDER BY id DESC LIMIT 1;"
```

Para newhash:

```
sqlite3 minibc.db "SELECT newhash FROM nblock ORDER BY id DESC LIMIT 1;"
```

Ahora nos enfocaremos de como haremos una solicitud de envío de una nueva operación y/o transacción para ser insertada en la cola de transacciones.

Hay dos pasos como requisitos para poder enviar una operación a la cola de transacciones.

Primer requisito que el balance de nuestra cuenta tenga los suficientes “activos” disponibles para cubrir el monto a ser enviado. Recordemos que los “activos” no solamente pueden ser un número o cantidad, sino que podemos crear “activos” de cualquier tipo dependiendo cada sistema a crear.

Ejemplos de activos:

- ✓ Cantidad intrínseca de una cantidad “moneda virtual o criptomoneda” de origen nuevo.
- ✓ Datos referenciados a datos digitales (todo tipo de documentos).
- ✓ Firmas de autenticación tipo “hash” de cadenas de caracteres o todo tipo de archivos.
- ✓ Cualquier transacciones o transferencia de información digital o representación de esta.

El balance de “activos” de cada nodo se obtiene usando el bloque (**GetBalance**).



Segundo requisito es proporcionar los mínimos parámetros cuando se envía una transacción, los cuales son:

- ✓ Dirección origen. - es la dirección de donde saldrán los activos para ser enviados.
- ✓ Dirección destino. – es la dirección del usuario que recibirá los activos enviados.
- ✓ Activo. - Es el dato con valor intrínseco dado en cada sistema para ser enviado en cada transacción.

Las direcciones anteriores (origen-destino) corresponden a las llaves públicas de cada usuario cuando fueron creadas las cuentas de cada usuario. Recordemos que al crear una cuenta de usuario se crean dos tipos de llaves pública y privada.

La llave publica es la dirección que será compartida en todo el sistema y que cualquier usuario del sistema puede verla y usarla para enviar o recibir transacciones.

La llave privada es la dirección que es usada de forma local por cada nodo y como su nombre lo indica es de uso privado que ayuda a crear firmas digitales para dar seguridad a las transacciones enviadas, esta llave nunca debe ser compartida y es de uso local y único del nodo origen.

Para poder usar los parámetros anteriores nos apoyaremos en dos repositorios donde se almacenan las direcciones “llaves privadas” que están en la base de datos keystore.db y será de uso local de cada nodo sin compartir en el sistema y el otro repositorio en donde se almacenan todas las direcciones “llaves públicas” que están en la base de datos publickeys.db será compartida a través del protocolo “Peer to Peer” en todos los nodos del sistema.

Las bases de datos “keystore.db” y “publickeys.db” ya fueron creadas en el Anexo “Creacion de bases de datos KeyStore & PublicKeys”.

La base de datos y sistema para la cola de transacciones ya la hemos creada en la sección de “Instalación y configuración red Mini Centinela RESTful”. En donde ya creamos una base de datos para las transacciones llamada op.sqlite3 en esta tenemos dos tablas una es “trans” que se encarga de las transacciones (**cola de transacciones**) y otra llamada “sign” donde se almacenan las firmas digitales de cada operación.

El sistema RESTful de SQLite es la red de respaldo en esta ocasión la usaremos por facilidad y para probar el sistema de respaldo, sin embargo, para cambiar y usar la misma base de datos op.sqlite3 en un modelo “Peer to Peer” solo tendremos que usar la base de datos en un modelo compartido nuevamente en el sistema de syncthing, esto lo veremos más adelante.

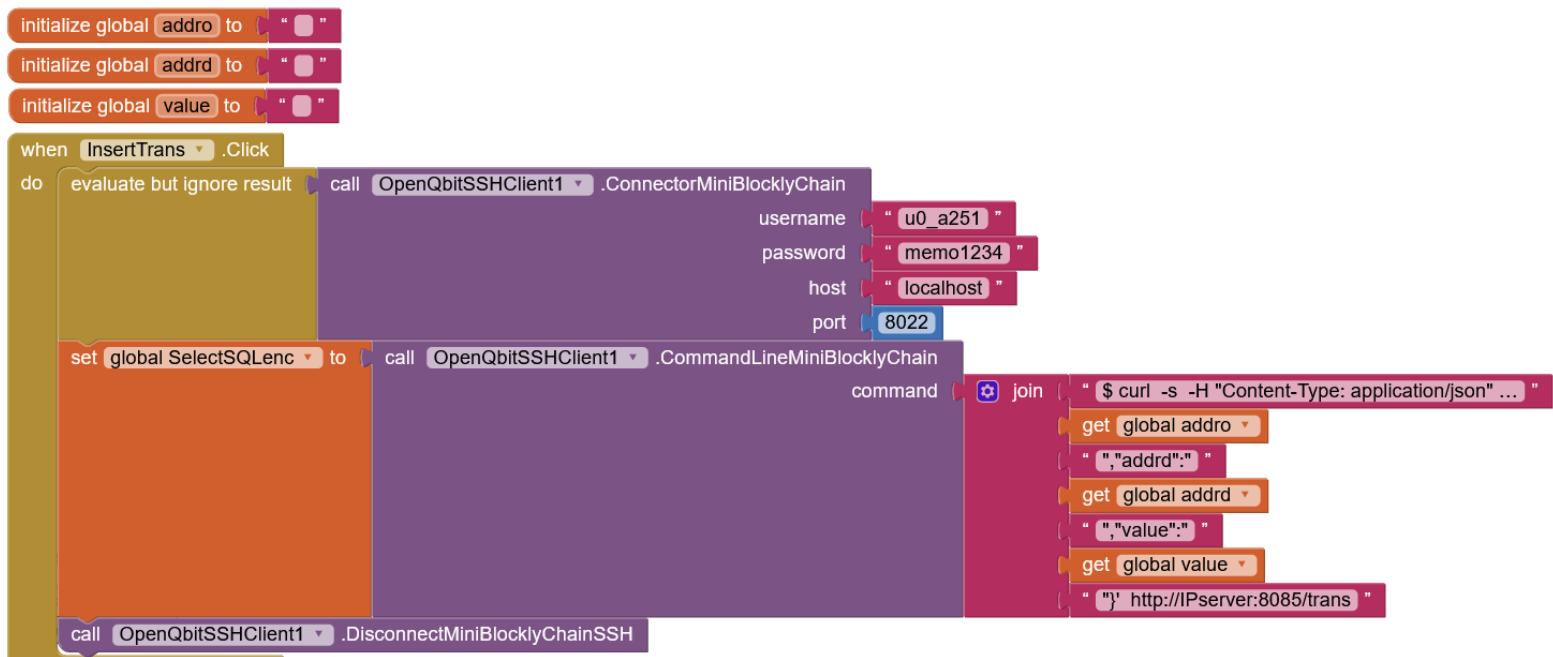
Empezaremos a enviar operaciones a la cola de transacciones usando RESTful aplicado a la base de datos op.sqlite.

Creamos una nueva transacción en la tabla “trans”

```
$ curl -s -H "Content-Type: application/json" -d
'{"addr": "MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9",
"addrd": "MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh", "value": "999"}'
http://IPserver:8085/trans
```

```
# outputs
{
  "id": 1,
  "addr": "MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9",
  "addrd": "MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh",
  "value": "999"
}
```

Implementación en App Inventor:



Los parámetros de entrada: addro, addrd, value son variables que se pueden introducir en el algoritmo y se extraen de la base de datos keystore.db

Ver Anexo “Creacion de base de datos KeyStore”.

Ahora insertamos las firmas digitales de la transacción anterior. En la tabla “**sign**”

```
$ curl -s -H "Content-Type: application/json" -d '{
  "trans_id":1
  "sign":"59ahGVn8pZ4oduxtTWCCvRiqjWEEy1c62",
  "hash": " f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68"
  http://IPserver:8085/sign

  # outputs
  {
    "id": 1,
    "trans_id": 1,
    "sign": "59ahGVn8pZ4oduxtTWCCvRiqjWEEy1c62",
    "hash": "f6a6b509376deebc8a5d70da9d0436943b76c3933f35c419ed48e78abab59a68"
  }
```

NOTA: la firma digital (sign) debe ser obtenida antes de enviar la sentencia del comando curl, esta se genera con el bloque (**GenerateSignature**).

Los parámetros de entrada: Trans_id, sign, hash son variables que se pueden introducir en el algoritmo y la firma digital de la transacción será generada con el bloque (**GenerateSignature**).

Enseguida veremos el procedimiento de generación de firma digital para aplicarla en cada transacción que se realice.

Generación de firma digital para transacción. Cuando usamos el bloque (**GenerateSignature**) tendremos como resultado un archivo tipo **file.sig** este archivo lo usaremos para ser guardado en el campo de sign en la tabla “**sign**”, esta firma será usada cuando la cola de transacciones sea procesada y es otro parámetro para dar seguridad entre el origen y destino, así como el monto o tipo de transacción entre ellos.

Para manejar datos binarios como el caso del archivo file.sig tendremos que convertirlo en base64 y posteriormente ya poder almacenarlo en la variable sign de la tabla “sign”. Para realizar la conversión nos apoyaremos con el bloque (**EncoderFileBase64**).

Como se calcula el valor del campo hash para la tabla “sign” de cada transacción:

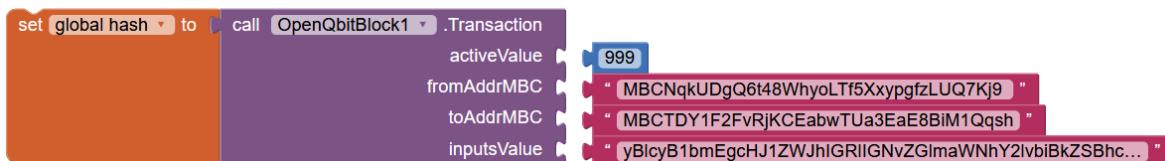
Hash de transacción = SHA256(Dirección Origen + Dirección Destino + Activo + fileBase64.sig)

Ejemplo de hash tipo SHA256:

SHA256(MBCNqkUDgQ6t48WhyoLTf5XxypgfzLUQ7Kj9 +
MBCTDY1F2FvRjKCEabwTUa3EaE8BiM1Qqsh + 999 +
yBlcyB1bmEgcHJ1ZWJhIGRIIGNvZGImaWNhY2lvbiBkZSBhcmNoaXZvIGJpbmFyaW8gYSBiYXNINjQKR
XN0byBlcyB1bmEgcHJ1ZWJhIGRIIGNvZGImaWNhY2lvbiBkZSBhcmNoaXZvIGJpbmFyaW8gYSBiYXNI
NjQKRXN0byBlcyB1bmEgcHJ1ZWJhIGRIIGNvZGImaWNhY2lvbiBkZSBhcmNoaXZvIGJpbmFyaW8gYSBiYXSB
iYXNINjQKRXN0byBlcyB1bmEgcHJ1ZWJhIGRIIGNvZGImaWNhY2lvbiBkZSBhcmNoaXZvIGJpbmFyaW
8gYSBiYXNINjQ=).

Output: 9d45198faaef624f2e7d1897dd9b3cede6ecca7fbac516ed1756b350fe1d56b4

Para el cálculo del hash deberemos apoyarnos en el Bloque(**Trasaccion**).



El parámetro de salida es el hash que se almacenara por cada transacción que se envía desde cualquier nodo del sistema este se almacena en el campo de sign de la tabla “sign” en la base op.sqlite

Con la operación anterior nos aseguramos que solo un hash único e irrepetible representara a cada transacción enviada a la cola y este hash representativo es representa la totalidad de información transmitida.

Solo nos falta incluir la variable de Trans_id podrá ser un identificador para diferenciar de que nodo es enviada la cada transacción, en nuestro ejemplo pondremos la variable Trans_id con valor fijo de “1”. Debido a que es el primer nodo que se está configurando en nuestra red. Este valor puede variar la sintaxis de acuerdo a cada diseño particular, por simplicidad hemos elegido un identificador simple.

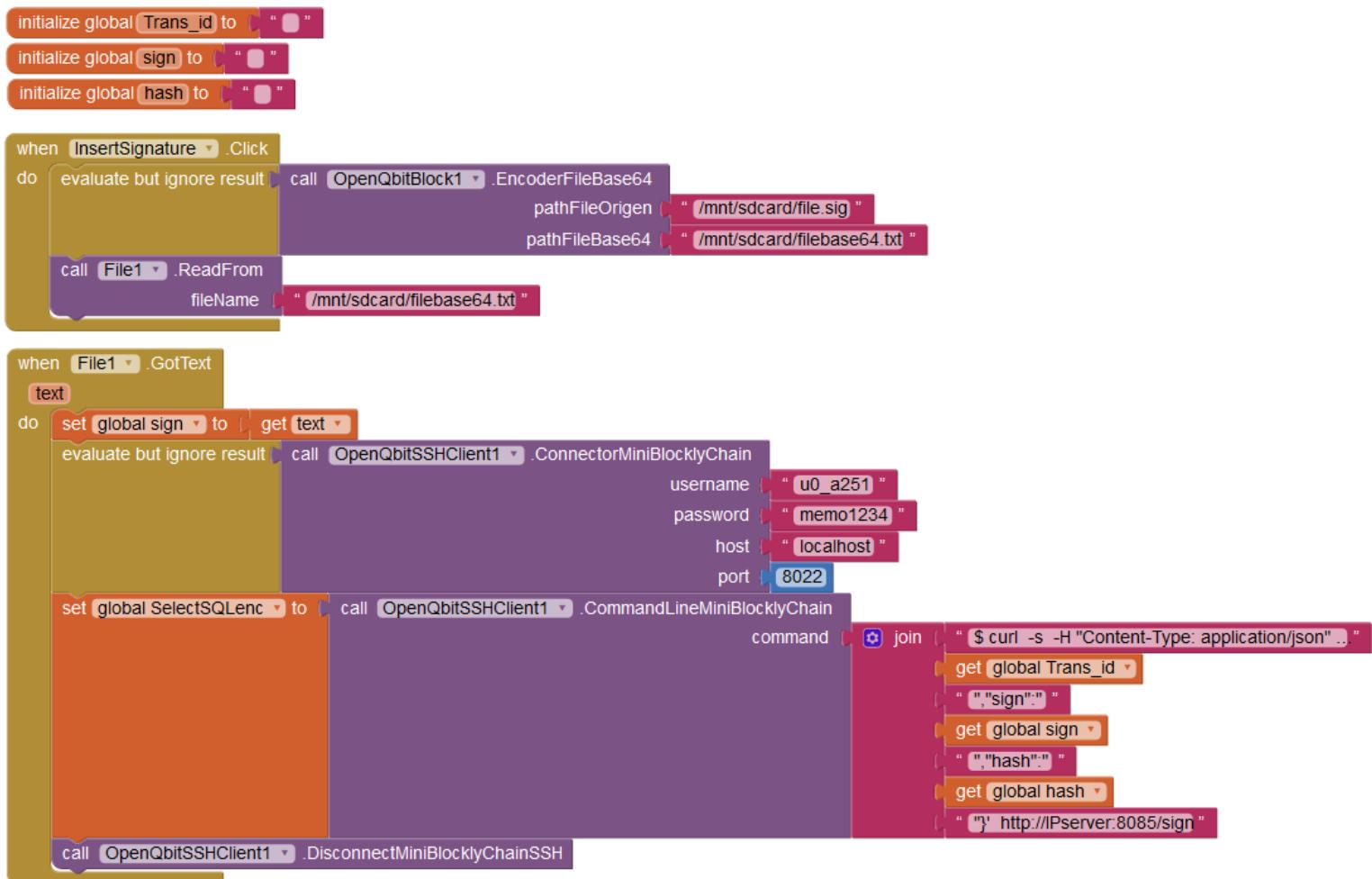
Ya que tenemos todas las variables definidas crearemos la estructura y secuencia de ejecución de bloques dentro de App Inventor, para realizar el INSERT en la tabla “sign”.

NOTA: Es importante tener en cuenta que cada envío de una transacción se compone de dos INSERT en la base de datos op.sqlite3, se deberá realizar uno en la tabla “trans” y sus respectivos valores en la tabla “sign”.

En el diseño de la base de datos op.sqlite3 se crearon dos tablas por separado debido a contemplar en el futuro poder cifrar la tabla “sign” únicamente ya que contiene información

que no debería enviarse en la red de forma plana, esta opción se deja a consideración de cada diseño futuro.

Crearemos la estructura de ejecución de bloques y métodos dentro de App Inventor del INSERT en la tabla “sign”.



Hasta este momento ya hemos finalizado el INSERT a la tabla “sign” y a la tabla “trans” estas se encuentran dentro de la base de datos **op.sqlite3** y los datos insertados en estas dos tablas servirán para la creación de la cola de transacciones que será enviada a todos los nodos para su procesamiento.

En este momento usaremos el Conector Java SQLite-Redis este conector lo que realizará es la conversión de los datos de la base de datos op.sqlite3 a la base de datos Redis. Ver Anexo “Conector Código Java SQLite-Redis”.

Después de haber implementado el Conector SQLite-Redis la cola de transacciones será entregada a todos los nodos del sistema mediante el sistema Redis.

Empezaremos el proceso de cómo podremos recibir la cola de transacciones de una forma adecuada para poder procesarla.

La cola de transacciones será entregada a través del servicio de Redis esta es una base de datos en tiempo real y que tiene sus respectivos bloques de control en el ambiente App Inventor.

Configuración del ambiente Redis dentro del sistema Blockly de App Inventor.

Un punto importante es comentar que los bloques para el control de un servidor Redis hasta el momento de realizar el presente manual únicamente está disponible en el sistema de App Inventor. En caso de usar un sistema de Blockly diferente al App Inventor se tendrá que usar la ejecución de Redis CLI (Command-Line) esto a través del envío de comandos a la terminal Termux vía la extensión (**OpenQbitSSHClient**).

Ejemplo de usos en Redis CLI (Command-Line):

Para obtener todas las **etiquetas** o llaves en Redis.

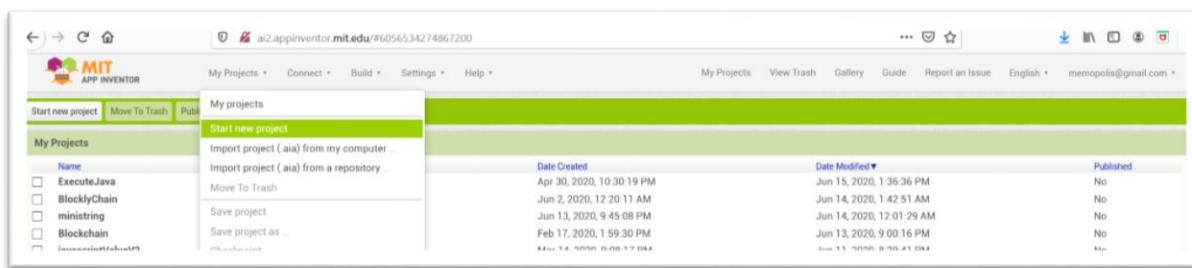
\$ redis-cli KEYS *

Para obtener valor de una llave.

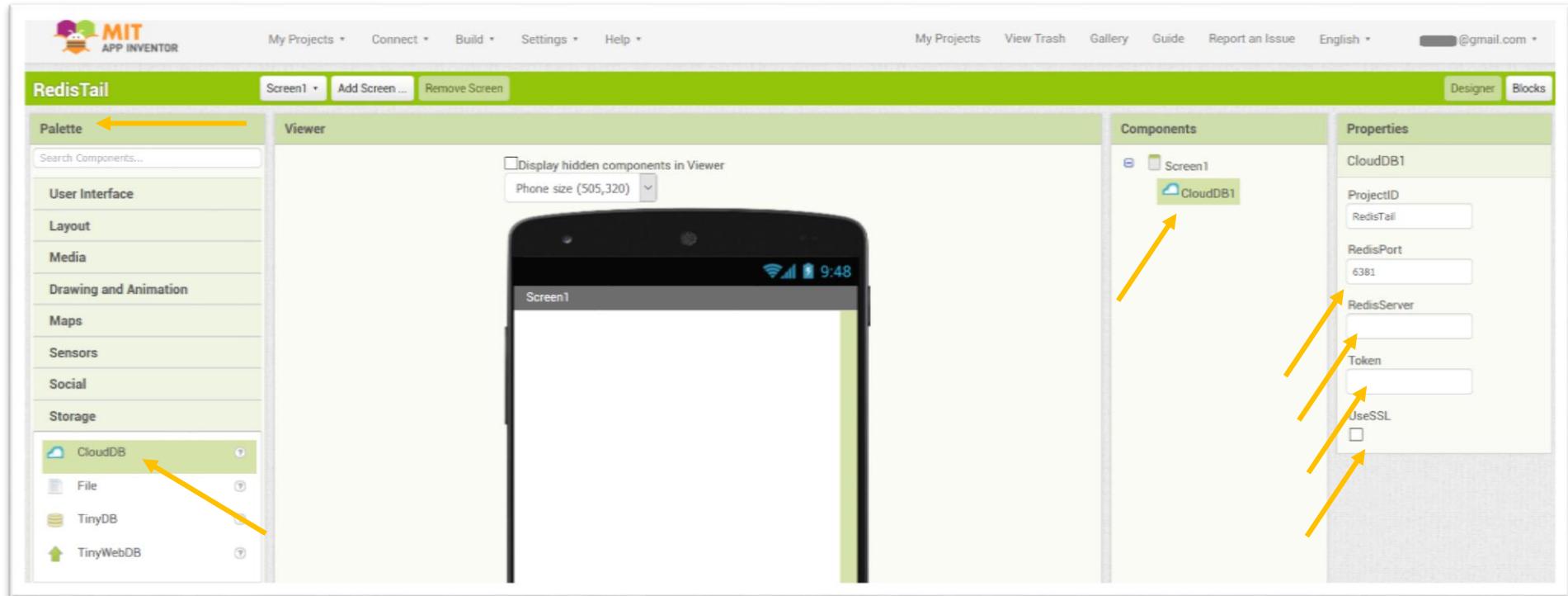
\$ redis-cli GET <key>

En nuestro caso por estar usando App inventor revisaremos como usar los bloques para trabajar con Redis.

Dentro de App Inventor tenemos varios bloques para usar con Redis, empezamos a crear un nuevo proyecto vamos a: My projects > Start new project



Después de crearlo el proyecto nos vamos a la parte superior izquierda y en la sección “Palette” damos click en “Storage” y arrastramos el objeto “CloudDB” esto nos integrara todas las funcionalidades para poder configurar una conexión a un servidor Redis.



La configuración es muy simple solo tenemos que dar los siguientes parámetros para podernos conectar a nuestro servidor de Redis (Local) que está ejecutándose en nuestro nodo (Teléfono móvil).

ProjectID. - Este es el ID con el que empezara la etiqueta con la que se identifica la cola de transacciones en Redis.

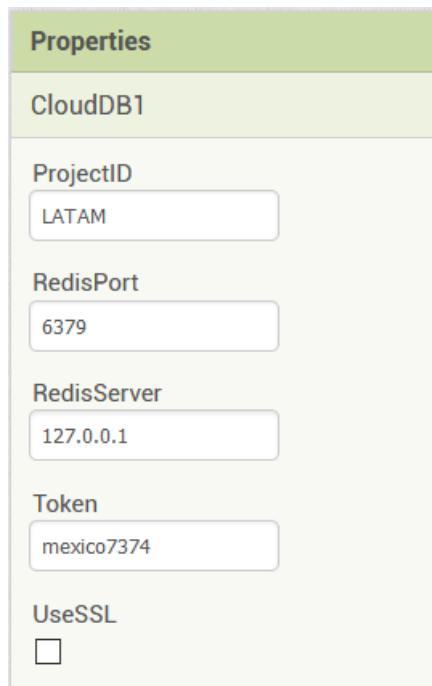
RedisPort. - Este es el puerto del servidor de Redis donde nos conectaremos por default es el 6379

RedisServer. - Este es el dominio o IP Local del nodo en nuestro caso y el de todos los nodos será 127.0.0.1

Token. - Este es el password del servidor Redis habilitado para poder conectarnos.

UseSSL. - Esta opción nos da la oportunidad de usar certificados SSL en nuestro caso lo dejamos sin habilitar.

En nuestro modelo y diseño de sistema tendremos los siguientes parámetros en todos los nodos de la red.



Es momento de revisar los bloques que nos proporcionan el control y procesamiento de datos sobre un ambiente de base de datos Redis.

Iniciamos con el bloque método (**DataChanged**)



Este método cada vez que haya un cambio en el servidor Redis que hemos configurado nos entregara dos valores:

tag. - Este valor es la etiqueta que identifica a la cola de transacciones.

value. - Este valor contiene la lista de todas las transacciones enviadas a ser procesadas. Este valor es una lista de cadena de caracteres tipo <String>.

En el caso del “value” es donde empezaremos a realizar nuestro procesamiento de información de la siguiente forma.

Como nos entrega una cadena de todas las transacciones de los valores hash, empezaremos por revisar si esta cadena es validad y verificar sino ha sido modificada desde su origen. Esto lo hacemos utilizando un algoritmo muy útil para la verificación de cantidades grandes de información.

Usaremos el algoritmo del árbol de merkle. Al aplicar este algoritmo nos da una seguridad en la integridad sobre la información que recibimos (cola de transacciones).

Veamos el siguiente ejemplo de una cola de transacciones en Redis, ejecutamos el Redis CLI (Command-Line) desde una terminal de Termux para revisar la llave “LATAM:HASH”, debemos ya haber ejecutado el Conector SQLite-Redis para poder consultar esta llave.

```
C:\memo\Redis>redis-cli
127.0.0.1:6379> get LATAM:HASH
"[\"9d45198faaef624f2e7d1897dd9b3cede6ecc7fbac516ed1756b350fe1d56b4 \",\""
f71c801a5fd25fc303ebc8c616204b4877ffb93006ec6a88bc30acf43ec250f5
\",\"60f8a3bcac1ea7d38e86efbc3e3e00480807d23f980391000766e804ed14ecb2
\",\"8a6dfe1d38c22e0f9212052efa6136da3edf1fb1b2a3e25224ac3d689124b754\"]"
127.0.0.1:6379>
```

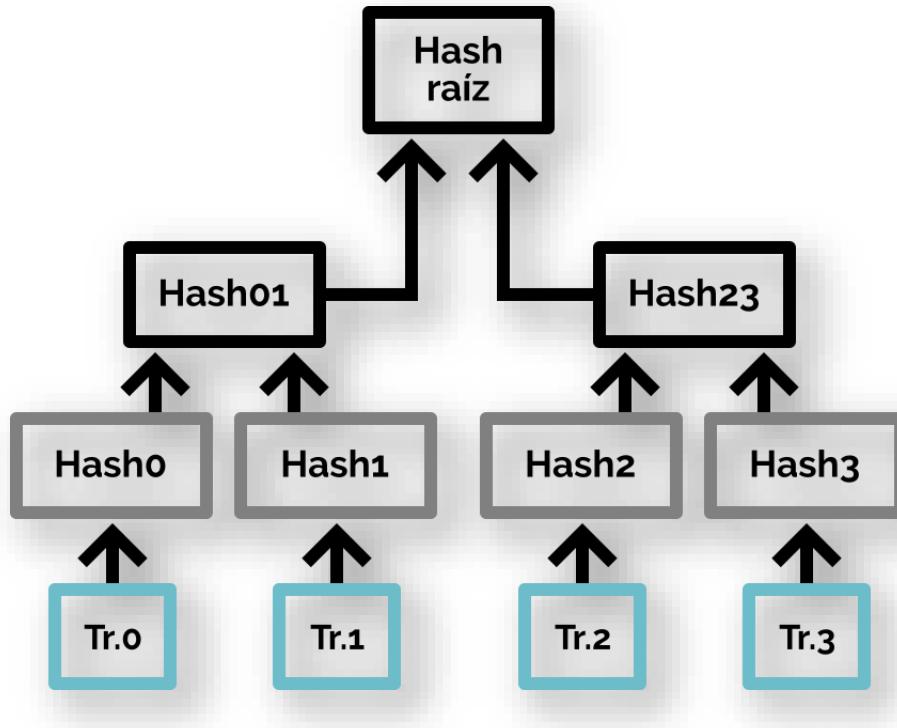
La anterior cola de transacciones únicamente tiene tres elementos ya que vemos cuatro hashes que representan una transacción individual que la integran y son los hashes de cada transacción que fue inyectada inicialmente en la base de datos op.sqlite3 dentro de las tablas “trans” y “sign”.

Ahora es momento de entender cómo funciona el árbol de merkle.

Un árbol hash de Merkle o árbol de merkle o árbol hash es una estructura de datos en árbol, binario o no, en el que cada nodo que no es una hoja está etiquetado con el hash de la concatenación de las etiquetas o valores de sus nodos hijo. Son una generalización de las listas hash y las cadenas hash.

En nuestro caso realizaremos el siguiente cálculo para calcular el árbol de merkle para cuatro elementos esto se realiza calculando el resultado de tomar parejas de datos concatenarlas y sacar sus respectivos hashes, los resultados del primer nivel se aplicarán a los resultados del segundo nivel hasta que solo quede un elemento final, a este se le llamará el hash merkleroot (hash raíz).

Veamos el siguiente diagrama que describe este proceso.



La cola de transacciones basada en hashes lo sacaremos mediante el Bloque (**GetMerkleRoot**)



Hash raíz: 51431822de7c94b90dc06d47b8f6275f315a4976c8479d30c32747fa90325432

Posteriormente el resultado se compara con la llave LATAM:merkleroot del sistema Redis local y deberán coincidir ambas llaves para verificar la integridad de datos.

Otro punto de seguridad que nos proporciona el árbol de merkle es confirmar que una específica transacción si está incluida en la cola de transacciones desde su origen y que no ha sido introducida por algún medio de comunicación externo o interno de forma fraudulenta.

En el caso de que la cadena sea de elementos impares en su sumatoria, el elemento ultimo de duplica para tener un arreglo para y empezar la ejecución del algoritmo.

Otro punto no menos importante es el momento de validar que cada transacción corresponda a su origen-destino y que el activo sea el que la dirección origen ha enviado.

Aquí es donde están el Bloque (VerifySignature).

```
call OpenQbitBlock1 .VerifySignature
```

Antes de ejecutar el anterior bloque primero deberá haberse bajado el archivo (file.sig) en formato binario de la tabla “sign” esto lo haremos ejecutando el siguiente SELECT:

```
sqlite3 op.sqlite3 "select sign from sign where=id_addr;"
```

id_addr: Es el id de la dirección origen de la tabla “trans”.

La anterior solicitud de búsqueda nos entregara un dato en formato Base64 de la firma digital de la transacción en curso, este para convertirlo en su formato original (binario) necesitaremos el Bloque(**DecoderFileBase64**).

Ya que tenemos nuestro archivo binario (file.sig) tendremos que cargarlo en el sistema la llave publica del origen y la llave publica del destinatario también en formato binario, teniendo los cuatro datos en sus respectivos formatos será el momento de ejecutar la verificación de la firma digital en el sistema.

- ✓ Llave publica dirección de origen
- ✓ Llave publica dirección del destinatario
- ✓ Activo enviado.
- ✓ Firma digital (file.sig)

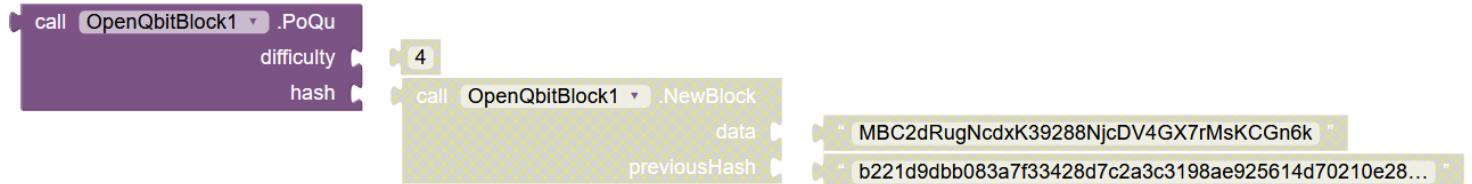
Las llaves públicas en formato binario podrán ser bajados en el formato adecuado (binario) desde la base de datos compartida publickeys.db

Este proceso deberá ser ejecutado por cada operación individual que integre la cola de transacciones.

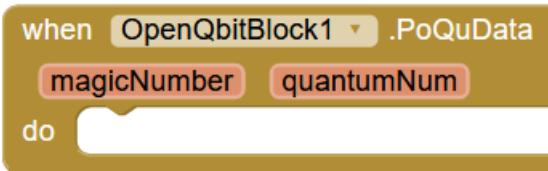
Por ultimo revisaremos como el sistema escoge de forma consensual a un nodo para poder ser elegido en agregar el siguiente bloque a la cadena de bloques y ser el que procese la cola de transacciones.

Esta forma de elegir al nodo ganador para procesar la cola de transacciones se basa en nuestro algoritmo desarrollado para un sistema Mini BlocklyChain.

Como implementamos el consenso PoQu “Proof of Quantum”. Este proceso de consenso está basado en la generación de números aleatorios cuánticos y se aplica usando el bloque (**PoQu**).



Primero obtenemos dos parámetros que nos proporciona el bloque (**PoQu**) entregados en el método (**PoQuData**) los parámetros son el **magicNumber** y **quantumNum**.



El **magicNumber** es el número “nonce”, es un numero entero que se obtiene realizando un PoW interno con dificultad no mayor a 5, este número se encarga de dar un primer requerimiento al nodo con el **magicNumber** el nodo podrá hacer un requerimiento para obtener un número del sistema de generación de números aleatorios cuánticos que este en el rango del 0 al 1, este número dará una probabilidad establecida aleatoriamente para el nodo en ejecución que se almacenará en la tabla llamada “vote”.

La tabla “vote” almacenará los **quantumNum** calculado por cada nodo, este almacenamiento se realizará con un numero de nodos hasta determinado tiempo establecido en cada diseño del sistema se recomienda tener de entradas $((N/2) + 1)$ donde “N” es la cantidad de nodos disponibles en el sistema y podrá ser establecido o controlado mediante una acción en la herramienta de administración de tareas “cron” de cada nodo.

Un punto importante es que para este punto ya debe de haber establecido una sincronización del tiempo local de cada nodo a través del sistema automático del teléfono móvil en caso de usar la red **“Peer to Peer”** en la transmisión de la cola de transacciones.

La configuración del agente cron en los nodos. Ver sección de “Sincronización de tiempo en los nodos del sistema (Teléfono móvil) en minutos y segundos”.

En este ejemplo como estamos usando la red de comunicaciones de respaldo no necesitamos la sincronización de minutos y segundos para los nodos debido a que nuestro ejemplo ocupa un esquema de “**Cliente-Servidor**” este tipo de comunicación es únicamente en el proceso de transmisión de la cola de transacciones. Todos los demás procesos entre nodos son a través de una comunicación “**Peer to Peer**”.

Ahora veremos la estructura, diseño y creación de la tabla “vote” que estará localizada dentro de la base de datos llamada “quorum.db” que también estaremos creando en este ejemplo.

\$ sqlite3

```
SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open quorum.db
sqlite> CREATE TABLE vote (id integer primary key AUTOINCREMENT NOT NULL, node_imei
VARCHAR NOT NULL, quantumNum INTEGER NOT NULL, nonce INTEGER NOT NULL);
sqlite> .quit
```

Enseguida se muestra la creación de la misma tabla **vote** pero es mediante la introducción de la sentencia SQL en forma segmentada:

\$ sqlite3

```
SQLite version 3.32.2 2020-06-20 15:25:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open quorum.db
sqlite> CREATE TABLE vote (
...>   id integer primary key AUTOINCREMENT,
...>   node_imei VARCHAR NOT NULL,
...>   quantumNum INTEGER NOT NULL,
...>   nonce INTEGER NOT NULL
...> );
sqlite> .tables
vote
sqlite> .quit
```

Veremos algo similar en la creación de la base “quorum.db” y tabla vote.

```

$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open quorum.db
sqlite> CREATE TABLE vote (
...>     id INTEGER primary key AUTOINCREMENT,
...>     node_imei VARCHAR NOT NULL,
...>     quantumNum INTEGER NOT NULL,
...>     time INTEGER NOT NULL
...> );
sqlite> .tables
vote
sqlite> .quit
$ 

```

The screenshot shows a terminal window on an Android device. The terminal output is as follows:

```

$ sqlite3
SQLite version 3.32.2 2020-06-04 12:58:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open quorum.db
sqlite> CREATE TABLE vote (
...>     id INTEGER primary key AUTOINCREMENT,
...>     node_imei VARCHAR NOT NULL,
...>     quantumNum INTEGER NOT NULL,
...>     time INTEGER NOT NULL
...> );
sqlite> .tables
vote
sqlite> .quit
$ 

```

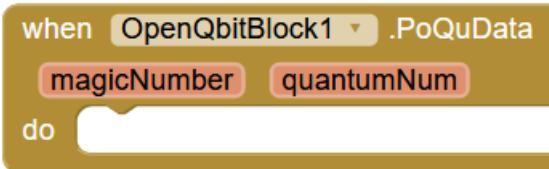
Below the terminal is a virtual keyboard. At the bottom of the screen are standard Android navigation buttons: back, home, and recent apps.

Ahora veamos como obtenemos los valores de la tabla “vote”.

node_imei. - Este valor lo sacamos usando el bloque (**GetDeviceID**).

call OpenQbitBlock1 .GetDeviceID

quantumNum. - Este valor es uno de los resultados del método (**PoQuData**) este se obtiene usando el bloque (**PoQu**).



nonce. - Este valor se obtiene de haber ya ejecutado el bloque (**PoQu**) de forma íntegra y el igual al magicNumber del método (**PoQuData**).

Después de haber concluido los INSERT en la tabla “vote” es necesario realizar una copia de la base de datos.

Después de un tiempo determinado y basado en el diseño de cada sistema, se ejecutará el servicio “cron” en cada nodo de la red y tendrá el siguiente SELECT para ser procesada en la tabla “vote”:

```
SELECT node_imei FROM vote WHERE magicNumber= (SELECT max(magicNumber) FROM vote);
```

El anterior SELECT nos devuelva el IMEI resultado con mayor probabilidad, ahora cada nodo que ejecute el SELECT realizará una comparación de su IMEI con el IMEI resultado y solo el nodo que coincida creará un archivo con el número de probabilidad mayor que será replicado en la red “Peer to Peer” mediante un archivo con el formato IMEI.mbc que contendrá el IMEI del nodo ganador.

El nodo ganador podrá empezar a realizar el procesamiento de la cola de transacciones. Basado en todos los bloques anteriores.

Dos puntos importantes es dependiendo de cada creación de sistema se tendrá que revisar tres procesos que serán personalizados por cada diseñador.

1.- Cuando el nodo ganador empiece el procesamiento de la cola de transacciones se deberá implementar un método o proceso en donde se tendrá que revisar que el nodo ganador está en línea y no se ha perdido la comunicación con la red.

2.- Cuando se empiece el procesamiento de la cola de transacciones el nodo ganador deberá de iniciar dos banderas de control señalando el inicio del procesamiento y otra para el confirmar que se haya terminado el procesamiento de la cola de transacciones estas dos banderas deberán estar compartida en la red hacia todos los nodos, esto ayudara a localizar fallas de conexión o falla en procesamiento o tiempo de procesamiento demasiado.

3.- En caso de fallo de comunicación u otro evento en donde el nodo ganador no haya podido procesar la cola de transacciones se deberá escoger al nodo siguiente en el rango de probabilidad inmediato.

El anterior punto podrá ser controlado con un servicio que compruebe que el nodo ganador está en línea pudiendo usar el servicio “cron”, el script deberá ser desarrollado para cada caso diseñado sin embargo en seguida se muestra un ejemplo genérico de un script Shell para poder ser modificado según sea las necesidades de cada sistema Mini Blocklychain.

```
#!/bin/bash
dir="/data/data/com.termux/files/home/Sync/imei";
if [ !"$(ls $directorio)" ]
then
    sqlite3 quorum.db "UPDATE vote SET magicNumber=0 WHERE magicNumber= (SELECT
max(magicNumber) FROM vote);"
else
    MAX_NUM=$(sqlite3 quorum.db "SELECT max(magicNumber) FROM vote;")
    IMEI_quorum=$(sqlite3 quorum.db "SELECT node_id FROM vote WHERE=MAX_NUM")
    IMEI_local=$(cat device_imei) // Usar el bloque (GetDevice)
    if [ IMEI_quorum -eq IMEI_local ]
    then
        touch $MAX_NUM > IMEI.mbc
    fi
fi
exit
```

31. Anexo “Integracion con ambientes Ethereum & Bitcoin”.

Ahor veremos como podemos integrar los dos sistemas de blockchain mas conocidos a nivel mundial especializados en criptomonedas como son Ethereum y Bitcoin.

Empecemos con la instalación del software que nos ayudara a realizar todas las transacciones posibles en el ambiente Ethereum.

¿Que es Ethereum?

Ethereum es una plataforma open source, descentralizada a diferencia de otras cadenas de bloques, Ethereum puede hacer mucho más. Es programable, lo que significa que los desarrolladores pueden usarlo para crear nuevos tipos de aplicaciones.

Estas aplicaciones descentralizadas (o "dapps") obtienen los beneficios de la criptomoneda y la tecnología blockchain. Son confiables y predecibles, lo que significa que una vez que se "cargan" en Ethereum, siempre se ejecutarán según lo programado. Pueden controlar los activos digitales para crear nuevos tipos de aplicaciones financieras. Se pueden descentralizar, lo que significa que ninguna entidad o persona los controla.

En este momento, miles de desarrolladores de todo el mundo están creando aplicaciones en Ethereum e inventando nuevos tipos de aplicaciones, muchas de las cuales puede usar hoy en día:

- Carteras de criptomonedas que le permiten realizar pagos baratos e instantáneos con ETH u otros activos
- Aplicaciones financieras que le permiten pedir prestado, prestar o invertir sus activos digitales
- Mercados descentralizados, que le permiten intercambiar activos digitales, o incluso intercambiar "predicciones" sobre eventos en el mundo real.
- Juegos donde tienes activos en el juego e incluso puedes ganar dinero real.
- Tiene contratos inteligentes que son programas con acuerdos para ser ejecutados al cumplirse las premisas con las que se elaboro o creo.

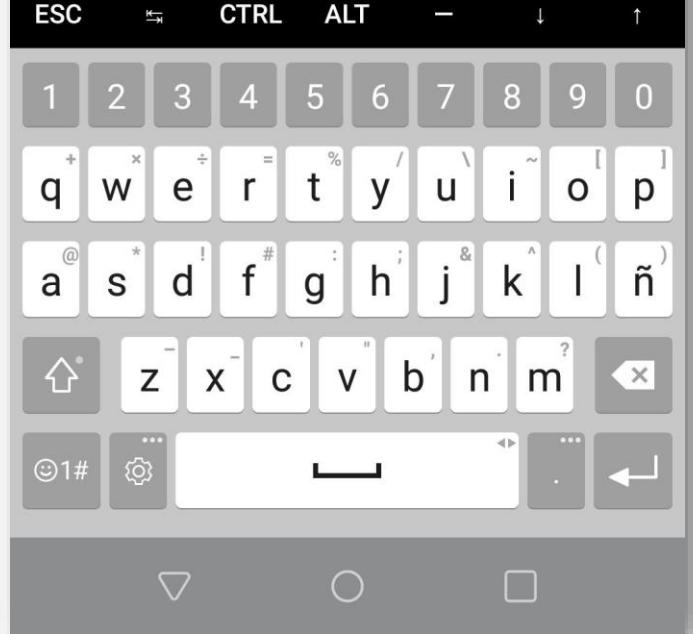
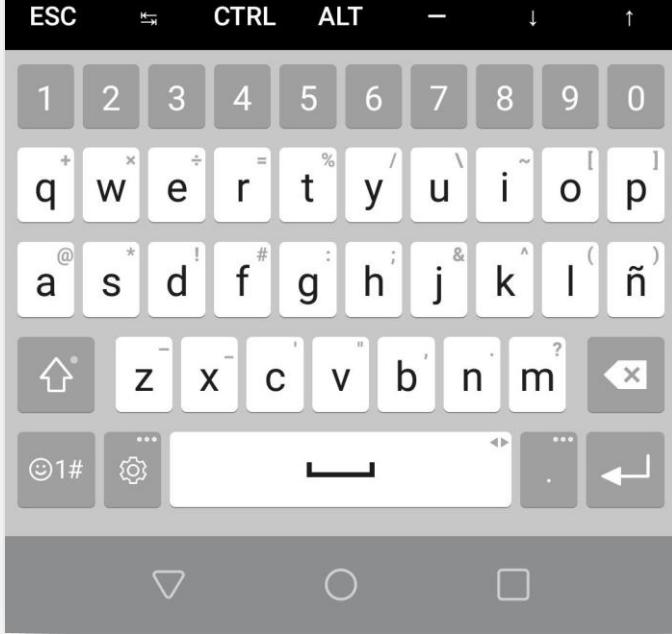
Los contratos inteligentes comparten similitudes con las DApps (aplicaciones descentralizadas), pero también les separan algunas diferenciales importantes.

Al igual que los contratos inteligentes, una DApp es una interfaz que conecta a un usuario con un servicio de un proveedor a través de una red de pares descentralizada. Pero, mientras que los contratos inteligentes necesitan un número fijo de participantes para ser creados, las DApps no tienen límite de usuarios. Además, no solo se reducen a aplicaciones financieras como los contratos inteligentes: una DApp puede tener cualquier finalidad que a uno se le ocurra.

En nuestro caso usaremos la librería llamada “Web3j” que está desarrollada en Java, y que permite interactuar con la blockchain de Ethereum de manera sencilla e intuitiva.

Ejecutamos el siguiente comando para la instalación de “Web3j”:

```
$ curl -L get.web3j.io | sh
```



Despues debemos crear la variable de ambiente **\$JAVA_HOME** con la ruta en donde se encuentra el ejecutable “java” ya que la librería buscara en esta variable para poder ser ejecutada exitosamente.

```
$ JAVA_HOME= /data/data/com.termux/files/usr/bin
```

Ya creada la variable tendremos que pasarnos al directorio en donde se instaló la libreria “Web3j” ejecutando el siguiente comando, un punto importante es que el directorio esta oculto despues de dr el comando de “cd” ponemos un punto “.” Y enseguida el directorio sin espacios, como sigue:

```
$ cd .web3j
```

Ya dentro enseguida probamos si esta ejecutandose la libreria de manera adecuada con el siguiente comando:

```
$ ./web3j versión
```

Nos da como resultado algo muy similar a:

```

$ ls
source.sh  web3j  web3j-4.5.16
$ ./web3j version

Version: 4.5.16
Build timestamp: 2020-03-06 14:13:49.943 UTC
$ 
```

Posteriormente crearemos una Wallet para poder usarla en el ambiente de blockchain de Ethereum de la siguiente forma:

\$./web3j wallet create

El anterior comando nos entrega la dirección de ethereum:

4598fe2fd6afe2508f58343c7d42f2ab492edf34

```
$ ./web3j wallet create
Please enter a wallet file password:
Please re-enter the password:
Please enter a destination directory location [/data/data/com.termux/files/home/.ethereum/testnet/keystore]:
Wallet file UTC--2020-06-27T06-12-23.819752000Z-4598fe2fd6afe2508f58343c7d42f2ab492edf34.json successfully created in: /data/data/com.termux/files/home/.ethereum/testnet/keystore
$
```

Nos da como resultado un archivo en formato JSON en donde contiene la dirección y datos cifrados para que posteriormente sean usados para generar la llave privada de la cuenta que se acaba de crear.

Este archivo tipo JSON sera creado en un directorio por default llamado keystore.

De aquí en adelante podremos ya realizar operaciones usando y/o ejecutando el comando Web3j.

Lo anterior lo podremos hacer usando la extencion (**ConnectorSSHClient**).

Para saber como usar los diferentes parámetros y operaciones de la librería “Web3j” podremos ayudarnos consultando la documentación en su sitio oficial.

<https://docs.web3j.io/>

Para el ambiente de Bitcoin tenemos dos opciones usar el bloque () que genera la cuenta de Bitcoin y sus respectivas llaves publica y privada, esta las podremos usar para integrarnos al ambiente de Bitcoin de la siguiente forma instalamos la librería de java “Bitcoij”.

\$ npm install bitcoinj



```
3% 1:43 a.m.  
$ npm install bitcoinj  
+ bitcoinj@0.0.0  
added 1 package from 1 contributor and audited 2  
09 packages in 20.528s  
$
```

Para poder usar esta librería nos apoyaremos en su sitio oficial.

<https://bitcoinj.github.io/>

Una segunda opción para poder integrarnos al ambiente de blockchain Bitcoin, lo podremos conseguir instalando el paquete de Bitcoind para Termux como se muestra a continuación.

\$ apt install bitcoin



```
61% 9:26 p.m.  
$ pkg install bitcoin  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following NEW packages will be installed:  
  bitcoin  
0 upgraded, 1 newly installed, 0 to remove and 1  
9 not upgraded.  
Need to get 3601 kB of archives.  
After this operation, 15.0 MB of additional disk  
space will be used.  
Get:1 https://dl.bintray.com/termux/termux-pac  
ges-24 stable/main arm bitcoin arm 0.20.0 [3601  
kB]  
Fetched 3601 kB in 2s (1253 kB/s)  
Selecting previously unselected package bitcoin.  
(Reading database ... 19110 files and directo  
ries currently installed.)  
Preparing to unpack .../bitcoin_0.20.0_arm.deb .  
..  
Unpacking bitcoin (0.20.0) ...  
Setting up bitcoin (0.20.0) ...  
$
```

Ahora cuando ejecutamos el agente de “bitcoind” sobre la terminal de Termux automáticamente se creará una dirección en el directorio:

/data/data/com.termux/files/hme/.bitcoin

En este caso de Bitcoin nos apoyaremos con la siguiente documentación del agente “Bitcoind”.

Para este caso también nos podremos apoyar en la extensión ([ConnectorSSHCliente](#)) para ejecutar el agente “bitcoind” dependiendo cada necesidad.

Descripción de parámetros para uso con bitcoind.

NOMBRE

bitcoind - iniciar Bitcoin Core Daemon

SINOPSIS

bitcoind [*opciones*] Iniciar *Bitcoin Core Daemon*

DESCRIPCIÓN

Iniciar Bitcoin Core Daemon

OPCIONES

-?

Imprima este mensaje de ayuda y salga

-alertnotify=<cmd>

Ejecute el comando cuando se reciba una alerta relevante o veamos una bifurcación muy larga (%s en cmd es reemplazado por el mensaje)

-assumevalid=<hex>

Si este bloque está en la cadena asuma que él y sus ancestros son válidos y potencialmente saltee su verificación de escritura (0 para verificar todo, por defecto:
00000000000000000000f1c54590ee18d15ec70e68c8cd4cfbadb1b4f11697eee, testnet:
000000000000000037a8cd3e06cd5edbfe9dd1dbcc5dacab279376ef7fcf2b4c75)

-blocknotify=<cmd>

Ejecutar el comando cuando el mejor bloque cambie (%s en cmd es reemplazado por el hash del bloque)

-blockreconstructionextratxn=<n>

Transacciones extra para mantener en la memoria para reconstrucciones de bloques compactos (por defecto: 100)

-blocksdir=<dir>

Especificar el directorio de bloques (por defecto: <datadir>/bloques)

-sólo en bloque

Si rechazar las transacciones de los compañeros de la red. Las transacciones de la cartera o RPC no se ven afectadas. (por defecto: 0)

-conf=<archivo>

Especifique el archivo de configuración. Las rutas relativas serán prefijadas por la ubicación del datadir. (por defecto: bitcoin.conf)

-daemon

Corre en el fondo como un demonio y acepta los comandos

-datadir=<dir>

Especificar el directorio de datos

-dbcache=<n>

Tamaño máximo de la caché de la base de datos <n> MiB (4 a 16384, por defecto: 450). Además, la memoria mempool no utilizada se comparte para esta caché (ver **-maxmempool**).

-debuglogfile=<fichero>

Especifique la ubicación del archivo de registro de depuración. Las rutas relativas se prefijarán con una ubicación del datadir específica de la red. (-**nodedebuglogfile** para desactivar; por defecto: debug.log)

-includeconf=<archivo>

Especificar un archivo de configuración adicional, relativo a la ruta **-datadir** (sólo se puede utilizar desde el archivo de configuración, no desde la línea de comandos)

-loadblock=<fichero>

Importa bloques del archivo externo blk000???.dat al inicio

-maxmempool=<n>

Mantenga el fondo de memoria de transacciones por debajo de <n> megabytes (por defecto: 300)

-maxorphantx=<n>

Mantener como máximo <n> transacciones desconectables en la memoria (por defecto: 100)

-mempoolexpiry=<n>

No mantenga las transacciones en el mempool más de <n> horas (por defecto: 336)

-par=<n>

Establece el número de hilos de verificación del guión (-6 a 16, 0 = auto, <0 = deja libres todos los núcleos, por defecto: 0)

-persistmempool

Si guardar el mempool al apagar y cargarlo al reiniciar (por defecto: 1)

-pid=<archivo>

Especifique el archivo PID. Las rutas relativas se prefijarán con una ubicación del datadir específica de la red. (por defecto: bitcoind.pid)

-punza=<n>

Reducir las necesidades de almacenamiento permitiendo la poda (eliminación) de los bloques viejos. Esto permite llamar a la cadena de poda RPC para eliminar bloques específicos, y permite la poda automática de bloques viejos si se proporciona un tamaño objetivo en MIB. Este modo es incompatible con **-txindex** y **-rescan**. Advertencia: Para revertir esta configuración es necesario volver a descargar toda la cadena de bloques. (por defecto: 0 = deshabilitar la poda de bloques, 1 = permitir la poda manual a través de RPC, >=550 = podar automáticamente los archivos de bloques para que se mantengan por debajo del tamaño objetivo especificado en MIB)

-reindex

Reconstruye el estado de la cadena y el índice de bloques de los archivos blk*.dat en el disco

-reindex-chainstate

Reconstruir el estado de la cadena a partir de los bloques actualmente indexados. Cuando esté en modo de poda o si los bloques en el disco pueden estar corruptos, use el **re-index** completo en su lugar.

-suspiros

Crear nuevos archivos con permisos por defecto del sistema, en lugar de umask 077 (sólo efectivo con la funcionalidad de la cartera desactivada)

-txindex

Mantener un índice de transacciones completo, utilizado por la llamada rpc de getrawtransaction (por defecto: 0)

-versión

Versión para imprimir y salir

Opciones de conexión:

-addnode=<ip>

Añade un nodo al que conectarse e intenta mantener la conexión abierta (ver la ayuda del comando RPC del "addendum" para más información). Esta opción puede ser especificada varias veces para añadir varios nodos.

-banscore=<n>

Umbral para desconectar a los compañeros que se comportan mal (por defecto: 100)

-Mientras tanto...

Número de segundos para evitar que los compañeros que se comportan mal se reconecten (por defecto: 86400)

-bind=<addr>

Ate a la dirección dada y siempre escuche en ella. Use la notación [host]:port para IPv6

-conexión=<ip>

Conectar sólo al nodo especificado; **-noconnect** desactiva las conexiones automáticas (las reglas para este par son las mismas que para **-addnode**). Esta opción puede especificarse varias veces para conectar con varios nodos.

-descubrir

Descubrir las propias direcciones IP (por defecto: 1 cuando se escucha y no **-externalip** o **-proxy**)

-dns

Permitir búsquedas DNS para **-addnode**, **-seednode** y **-connect** (por defecto: 1)

-dnsseed

Consulta de direcciones de pares a través de la búsqueda de DNS, si es baja en las direcciones (por defecto: 1 a menos que se utilice **-conexión**)

-enablebip61

Enviar mensajes de rechazo por BIP61 (por defecto: 1)

-externalip=<ip>

Especifique su propia dirección pública

-forcednsseed

Siempre consulta las direcciones de los compañeros a través de la búsqueda de DNS (por defecto: 0)

-escucha

Aceptar conexiones desde el exterior (por defecto: 1 si no hay **-proxy** o **-conexión**)

-listenonionion

Crear automáticamente el servicio oculto de Tor (por defecto: 1)

-maxconexiones=<n>

Mantener como máximo <n> conexiones con los compañeros (por defecto: 125)

-maxreceivebuffer=<n>

Máximo buffer de recepción por conexión, <n>*1000 bytes (por defecto: 5000)

-maxsendbuffer=<n>

Buffer de envío máximo por conexión, <n>*1000 bytes (por defecto: 1000)

-ajuste de tiempo máximo

El máximo permitido para el ajuste de la compensación del tiempo medio de los pares. La perspectiva local del tiempo puede estar influenciada por los pares hacia adelante o hacia atrás por esta cantidad. (por defecto: 4200 segundos)

-maxuploadtarget=<n>

Trata de mantener el tráfico de salida bajo el objetivo dado (en MiB por 24h), 0 = sin límite (por defecto: 0)

-onion=<>ip:port>

Usar un proxy SOCKS5 separado para alcanzar a los compañeros a través de los servicios ocultos de Tor, establecer **-noonion** para desactivar (por defecto: **-proxy**)

-onlynet=<net>

Hacer conexiones salientes sólo a través de la red <net> (ipv4, ipv6 o cebolla). Las conexiones entrantes no se ven afectadas por esta opción. Esta opción puede especificarse varias veces para permitir varias redes.

-filtros de pares

Soporta el filtrado de bloqueos y la transacción con filtros de floración (por defecto: 1)

-permitbaremultisig

Releva no P2SH multisig (por defecto: 1)

-puerto=<puerto>

Escuche las conexiones en <puerto> (por defecto: 8333, testnet: 18333, regtest: 18444)

-proxy=<ip:port>

Conectar a través del proxy SOCKS5, configurar **-noproxy** para desactivar (por defecto: desactivado)

-proxyrandomize

Aleatorizar las credenciales para cada conexión de proxy. Esto habilita el aislamiento del flujo de Tor (por defecto: 1)

-seednode=<ip>

Conéctese a un nodo para recuperar las direcciones de los pares, y desconéctese. Esta opción puede ser especificada varias veces para conectar a varios nodos.

-timeout=<n>

Especificar el tiempo de espera de la conexión en milisegundos (mínimo: 1, por defecto: 5000)

-torcontrol=<ip>:<port>

Puerto de control de Tor a usar si la escucha de la cebolla está activada (por defecto: 127.0.0.1:9051)

-contraseña=<paso>

Contraseña del puerto de control de Tor (por defecto: vacío)

-upnp

Utilice UPnP para asignar el puerto de escucha (por defecto: 0)

-whitebind=<addr>

Vincularse a una dirección determinada y a los compañeros de la lista blanca que se conectan a ella. Use la notación [host]:port para IPv6

-whitelist=<dirección IP o red>

Los pares de la lista blanca se conectan desde la dirección IP dada (por ejemplo, 1.2.3.4) o la red anotada del CIDR (por ejemplo, 1.2.3.0/24). Puede especificarse varias veces. Los pares de la lista blanca no pueden ser prohibidos por la DoS

Opciones de cartera:

-tipo de dirección

Qué tipo de direcciones utilizar ("legacy", "p2sh-segwit", o "bech32", por defecto: "p2sh-segwit")

-evitar gastos parciales

Agrupar las salidas por dirección, seleccionando todas o ninguna, en lugar de seleccionar por cada salida. Se mejora la privacidad ya que una dirección sólo se utiliza una vez (a menos que alguien la envíe después de haberla gastado), pero puede dar lugar a tarifas

ligeramente más altas ya que la selección de monedas puede resultar subóptima debido a la limitación añadida (por defecto: 0)

-cambio de tipo

Qué tipo de cambio usar ("legacy", "p2sh-segwit", o "bech32"). El valor por defecto es el mismo que **-addressstype**, excepto cuando **-addressstype=p2sh-segwit** se utiliza una salida nativa segwit cuando se envía a una dirección nativa segwit)

-desde la billetera

No cargues la cartera y desactiva las llamadas RPC de la cartera

-discardfee=<amt>

La tasa de la tarifa (en BTC/kB) que indica su tolerancia para descartar el cambio añadiéndolo a la tarifa (por defecto: 0.0001). Nota: Una salida se descarta si es polvo a esta tasa, pero siempre descartaremos hasta la tasa de retransmisión de polvo y una tasa de descarte por encima de eso está limitada por la estimación de la tasa para el objetivo más largo

-fallbackfee=<amt>

Una tasa de tasas (en BTC/kB) que se utilizará cuando la estimación de las tasas tenga datos insuficientes (por defecto: 0,0002)

-keypool=<n>

Establecer el tamaño de la piscina de llaves a <n> (por defecto: 1000)

-mintxfee=<amt>

Las tarifas (en BTC/kB) menores que esto se consideran como una tarifa cero para la creación de la transacción (por defecto: 0.00001)

-paytxfee=<amt>

Tarifa (en BTC/kB) para añadir a las transacciones que envíe (por defecto: 0.00)

-rescan

Vuelva a escanear la cadena de bloques para buscar transacciones de cartera faltante al inicio

-salvagewallet

Intentar recuperar las llaves privadas de una cartera corrupta en el arranque

-gastando el intercambio de información

Gasta el cambio no confirmado al enviar las transacciones (por defecto: 1)

-txconfirmtarget=<n>

Si no se establece una tarifa de pago, incluya una tarifa suficiente para que las transacciones comiencen la confirmación en promedio dentro de n bloques (por defecto: 6)

-mantenimiento de la cartera

Actualizar la cartera al último formato al inicio

-wallet=<path>

Especifique la ruta de la base de datos de la cartera. Puede especificarse varias veces para cargar varias carteras. La ruta se interpreta en relación con <walletdir> si no es absoluta, y se creará si no existe (como un directorio que contiene un archivo wallet.dat y archivos de registro). Para la compatibilidad con el pasado, también aceptará los nombres de los archivos de datos existentes en <walletdir>).

-walletbroadcast

Hacer las transacciones de difusión de la cartera (por defecto: 1)

-walletdir=<dir>

Especificando el directorio para guardar las carteras (por defecto: <datadir>/carteras si existe, de lo contrario <datadir>)

-walletnotify=<cmd>

Ejecutar el comando cuando cambie una transacción de la cartera (%s en cmd es reemplazado por TxID)

-walletrbf

Envíe las transacciones con la opción de inclusión de todo el RBF activada (sólo RPC, por defecto: 0)

-zapwallettxes=<modo>

Borrar todas las transacciones de la cartera y sólo recuperar las partes de la cadena de bloqueo a través de **-rescanear** al inicio (1 = mantener los metadatos tx, por ejemplo, información de solicitud de pago, 2 = dejar caer los metadatos tx)

Opciones de notificación de ZeroMQ:

-zmqpubhashblock=<dirección>

Habilitar el bloque de publicación en <dirección>

-zmqpubhashblockhwm=<n>

Establecer el bloqueo de publicación del mensaje saliente con una marca de agua alta (por defecto: 1000)

-zmqpubhashtx=<dirección>

Habilitar la publicación de la transacción de hachís en <dirección>

-zmqpubhashtxhwm=<n>

Establecer publicar el mensaje de salida de la transacción hash marca de agua alta (por defecto: 1000)

-zmqpubrawblock=<dirección>

Habilitar el bloque de publicación en bruto en <dirección>

-zmqpubrawblockhwm=<n>

Establecer publicar bloque crudo mensaje saliente marca alta de agua (por defecto: 1000)

-zmqpubrawtx=<dirección>

Habilitar la publicación de la transacción en bruto en <dirección>

-zmqpubrawtxhwm=<n>

Establecer publicar mensaje de salida de transacción bruta marca de agua alta (por defecto: 1000)

Opciones de depuración/prueba:

-debug=<categoría>

Información de depuración de salida (por defecto: **-nodebug**, suministrando <categoría> es opcional). Si no se suministra <categoría> o si <categoría> = 1, se da salida a toda la información de depuración. <categoría> puede ser: net, tor, mempool, http, bench, zmq, db, rpc, estimatefee, addrman, selectcoins, reindex, cmpctblock, rand, prune, proxy, mempoolrej, libevent, coindb, qt, leveldb.

-debugexclude=<categoría>

Excluir la información de depuración de una categoría. Puede utilizarse junto con **-debug=1** para generar registros de depuración para todas las categorías excepto una o más categorías especificadas.

-help-debug

Imprimir mensaje de ayuda con opciones de depuración y salir

-logips

Incluir las direcciones IP en la salida de depuración (por defecto: 0)

-logtimestamps

Prepárese la salida de depuración con el sello de tiempo (por defecto: 1)

-maxtxfee=<amt>

Total máximo de tasas (en BTC) para utilizar en una sola transacción de cartera o en una transacción en bruto; si se fija en un nivel demasiado bajo puede abortar grandes transacciones (por defecto: 0,10)

-impresión para la consola

Enviar información de rastreo/depuración a la consola (por defecto: 1 cuando no hay **-daemon**. Para desactivar el registro en el archivo, establecer **-nodebuglogfile**)

-shrinkdebugfile

Reducir el archivo debug.log al inicio del cliente (por defecto: 1 cuando no hay **-debug**)

-uacomment=<cmt>

Añadir un comentario a la cadena de agentes de usuario

Opciones de selección de la cadena:

-testnet

Use la cadena de prueba...

Opciones de retransmisión de nodos:

-bytespersigop

Bytes equivalentes por sigop en las transacciones de retransmisión y minería (por defecto: 20)

-datacarrier

Transacciones de portadores de datos de retransmisión y de mina (por defecto: 1)

-datacarrierize

Tamaño máximo de los datos en las transacciones de los portadores de datos que retransmitimos y extraemos (por defecto: 83)

-mempool-reemplazo

Habilitar la sustitución de transacciones en el pool de memoria (por defecto: 1)

-minrelaytxfee=<amt>

Las tarifas (en BTC/kB) menores que esto se consideran como una tarifa cero para la retransmisión, la extracción y la creación de transacciones (por defecto: 0.00001)

-whitelistforcerelay

Forzar la retransmisión de las transacciones de los compañeros de la lista blanca, incluso si las transacciones ya estaban en el mempool o violan la política de retransmisión local (por defecto: 0)

-whitelistrelay

Aceptar las transacciones transmitidas recibidas de los pares de la lista blanca incluso cuando no se transmitan transacciones (por defecto: 1)

Bloquear las opciones de creación:

-blockmaxweight=<n>

Establecer el peso máximo del bloque BIP141 (por defecto: 3996000)

-blockmintxfee=<amt>

Fijar la tasa de comisión más baja (en BTC/kB) para las transacciones que se incluyan en la creación de bloques. (por defecto: 0.00001)

Opciones del servidor RPC:

-Restaurante

Aceptar solicitudes públicas de REST (por defecto: 0)

-rpcallowip=<ip>

Permitir las conexiones JSON-RPC de la fuente especificada. Son válidas para <ip> una sola IP (por ejemplo, 1.2.3.4), una red/máscara de red (por ejemplo, 1.2.3.4/255.255.255.0) o una red/CIDR (por ejemplo, 1.2.3.4/24). Esta opción puede especificarse varias veces

-rpcauth=<userpw>

Nombre de usuario y contraseña HMAC-SHA-256 para conexiones JSON-RPC. El campo <userpw> viene en el formato: <nombre de usuario>:<SALT>\$<HASH>. Una escritura pitón canónica se incluye en share/rpcauth. El cliente entonces se conecta normalmente usando el rpcuser=<USERNAME>/rpcpassword=<PASSWORD> par de argumentos. Esta opción puede especificarse varias veces

-rpcbind=<addr>[:port]

Vincularse a una dirección determinada para escuchar las conexiones JSON-RPC. ¡No exponga el servidor RPC a redes no confiables como la Internet pública! Esta opción es ignorada a menos que **-rpcallowip** también sea pasada. El puerto es opcional y anula **-rpcport**. Usa la notación [host]:port para IPv6. Esta opción puede ser especificada varias veces (por defecto: 127.0.0.1 y ::1 es decir, localhost)

-rpccookiefile=<loc>

Ubicación de la cookie de autorización. Las rutas relativas serán prefijadas por una ubicación del datadir específica de la red. (por defecto: data dir)

-rpcpassword=<pw>

Contraseña para las conexiones JSON-RPC

-rpcport=<port>

Escuche las conexiones JSON-RPC en <puerto> (por defecto: 8332, testnet: 18332, regtest: 18443)

-rpcserialversion

Establece la serialización de la transacción en bruto o el hexágono de bloque devuelto en modo no verboso, no segwit(0) o segwit(1) (por defecto: 1)

-rpcthreads=<n>

Establecer el número de hilos para atender las llamadas RPC (por defecto: 4)

-rpcuser=<usuario>

Nombre de usuario para las conexiones JSON-RPC

-servidor

Aceptar los comandos de línea de comando y JSON-RPC

32. Licenciamiento y uso de software.

Android

<https://source.android.com/setup/start/licenses>

Termux

<https://github.com/termux/termux-app/blob/master/LICENSE.md>

Node

<https://raw.githubusercontent.com/nodejs/node/master/LICENSE>

SQLite

<https://www.sqlite.org/copyright.html>

Git

<https://git-scm.com/about/free-and-open-source>

sqlite-to-rest

<https://github.com/olsonpm/sqlite-to-rest/blob/dev/license.txt>

Redis DB

<https://redis.io/topics/license>

WorldTimeAPI NTP

<http://worldtimeapi.org/pages/faqs#commercial-apps>

Tor Network

<https://github.com/torproject/tor/blob/master/LICENSE>

Syncthing Network

<https://forum.syncthing.net/t/syncthing-is-now-mpfv2-licensed/2133>

OpenSSH

<https://www.openssh.com/features.html>

Putty SSH

<https://www.chiark.greenend.org.uk/~sgtatham/putty/licence.html>

MIT App Inventor 2 Companion y App Inventor Blockly

<https://appinventor.mit.edu/about/termsofservice>

SQLite Expert Personal -freeware

<http://www.sqliteexpert.com/download.html>

Apache Ant

<https://ant.apache.org/license.html>

WGET

<https://www.gnu.org/software/wget/>

OpenJDK

<https://openjdk.java.net/legal/>

Extensiones externas:

JSONTOOLs

<https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html>

Licenciamiento versiones opensource y comercial de sistema Mini BlocklyChain consultar la página oficial <http://www.openqbit.com>

Mini BlocklyChain, MiniBlockly, BlocklyCode, MiniBlockMiniChain, QBlockly son marcas registradas por OpenQbit.

Mini BlocklyChain es dominio público.

Todo el código y la documentación en Mini BlocklyChain ha sido dedicado al dominio público por los autores. Todos los autores de códigos y representantes de las empresas para las que trabajan han firmado declaraciones juradas dedicando sus contribuciones al dominio público y los originales de esas declaraciones juradas se almacenan en una caja fuerte en las oficinas principales de OpenQbit México. Cualquier persona es libre de publicar, usar o distribuir las extensiones de Mini BlocklyChain (OpenQbit) original, ya sea en forma de código fuente o como binario compilado, para cualquier propósito, comercial o no comercial, y por cualquier medio.

El párrafo anterior se aplica al código y la documentación entregables en Mini BlocklyChain, aquellas partes de la biblioteca de Mini BlocklyChain que realmente agrupa y envía con una aplicación más grande. Algunos scripts utilizados como parte del proceso de compilación (por ejemplo, los scripts de "configuración" generados por autoconf) pueden estar incluidos en otras licencias de código abierto. Sin embargo, nada de estos scripts de compilación llega a la biblioteca entregable final de Mini BlocklyChain, por lo que las licencias asociadas con esos scripts no deberían ser un factor en la evaluación de sus derechos para copiar y usar la biblioteca Mini BlocklyChain.

Todo el código entregable en Mini BlocklyChain ha sido escrito desde cero. No se ha tomado ningún código de otros proyectos o de Internet abierto. Cada línea de código puede rastrearse hasta su autor original, y todos esos autores tienen dedicaciones de dominio público en el archivo. Por lo tanto, la base de código Mini BlocklyChain es limpia y no está contaminada con código con licencia de otros proyectos de código abierto, no contribución abierta

Mini BlocklyChain es de código abierto, lo que significa que puede hacer tantas copias como desee y hacer lo que quiera con esas copias, sin limitación. Pero Mini BlocklyChain no es de contribución abierta. Para mantener Mini BlocklyChain en el dominio público y garantizar que el código no se contamine con contenido patentado o con licencia, el proyecto no acepta parches de personas desconocidas. Todo el código en Mini BlocklyChain es original, ya que ha sido escrito específicamente para su uso por Mini BlocklyChain. No se ha copiado ningún código de fuentes desconocidas en Internet.

Mini BlocklyChain es de dominio público y no requiere una licencia. Aun así, algunas organizaciones quieren pruebas legales de su derecho a usar Mini BlocklyChain. Las circunstancias donde esto ocurre incluyen lo siguiente:

- Su empresa desea indemnización por reclamos de infracción de derechos de autor.
- Está utilizando Mini BlocklyChain en una jurisdicción que no reconoce el dominio público.
- Está utilizando de Mini BlocklyChain en una jurisdicción que no reconoce el derecho de un autor a dedicar su trabajo al dominio público.
- Desea tener un documento legal tangible como evidencia de que tiene el derecho legal de usar y distribuir de Mini BlocklyChain.
- Su departamento legal le dice que debe comprar una licencia.

Si alguna de las circunstancias anteriores se aplica a usted, OpenQbit, la compañía que emplea a todos los desarrolladores de Mini BlocklyChain, le venderá una Garantía de título para Mini BlocklyChain. Una garantía de título es un documento legal que afirma que los autores reclamados de Mini BlocklyChain son los verdaderos autores, y que los autores tienen el derecho legal de dedicar el Mini BlocklyChain al dominio público, y que OpenQbit se defenderá enérgicamente contra los reclamos de licenciamiento. Todos los ingresos de la venta de las garantías de título de Mini BlocklyChain se utilizan para financiar la mejora continua y el soporte de Mini BlocklyChain.

Código contribuido

Para mantener Mini BlocklyChain completamente libre y libre de derechos de autor, el proyecto no acepta parches. Si desea hacer un cambio sugerido e incluir un parche como prueba de concepto, sería genial. Sin embargo, no se ofenda si reescribimos su parche desde cero. El tipo de licenciamiento no-comercial u opensource quien lo use en esta modalidad y alguna similar sin compra de soporte ya sea uso individual o corporativo no importando el tamaño de empresa se rigira por las siguientes premisas legales.

Renuncia de garantía. A menos que lo exija la ley aplicable o acordado por escrito, el Licenciante proporciona el Trabajo (y cada El Colaborador proporciona sus Contribuciones) "TAL CUAL", **SIN GARANTÍAS O CONDICIONES DE NINGÚN TIPO**, ya sea expreso o implícito, incluidas, entre otras, cualquier garantía o condición de TÍTULO, NO INFRACCIÓN, COMERCIABILIDAD O APTITUD PARA UN PROPÓSITO PARTICULAR. Usted es el único responsable de determinar el correcto uso o redistribuir el Trabajo y asumir cualquier riesgo asociado con su ejercicio de permisos bajo esta Licencia.

Cualquier perdida económica o de algún tipo por el uso de este software el afectado no tendrá opción de pago de ningún tipo. Todo litigio legal las partes se someterán a tribunales únicamente en la jurisdicción de la Ciudad de México, país México.

Para soporte, uso y licenciamiento comercial se tendrá que realizar un acuerdo o contrato establecido entre OpenQbit o su corporativo y la parte interesada.

Los términos y condiciones de distribución comercialización pueden cambiar sin previo aviso, dirigirse al portal oficial de www.openqbit.com para ver cualquier modificación de cláusulas de soporte y licenciamiento no-comercial y comercial.

Cualquier persona, usuario, entidad privada, publica de cualquier índole legal o de cualquier parte del mundo quien simplemente use el software acepta sin condicionantes las cláusulas establecidas en este documento y las que se puedan modificar en cualquier momento en el portal de www.openqbit.co sin previo aviso pudiendo ser aplicadas a discreción de OpenQbit en uso no-comercial o comercial.

Cualquier duda e información de Mini BlocklyChain dirigirse a la comunidad de App Inventor o a las comunidades de diversos sistemas Blockly como son: AppBuilder, Trunkable, etc. y/o al correo opensource@openqbit.com por la demanda de preguntas puede tardar la respuesta de 3 a 5 días hábiles.

Soporte con uso comercial.

support@openqbit.com

Ventas uso comercial.

sales@openqbit.com

Información legal y preguntas o dudas de licenciamiento.

legal@openqbit.com

