



EX<sup>change</sup>  
tensions

Configuración & Administración.

Ethereum Exchange Extensión (EEE).

# Guía de usuario

versión 1.0.0 Beta

Noviembre 2021.

MayanCoin y MayaCoin son marcas registradas Mayan Inc, bajo licencia de uso libre y comercial. Términos y condiciones de uso en: [www.MayanCoin.com](http://www.MayanCoin.com)

MayanCoin usa tecnología basada en [www.COINlidation.org](http://www.COINlidation.org) uso autorizado de oneCkey.

## Contenido

1.	Introducción.....	3
2.	¿Qué es la programación Blockly? .....	5
3.	¿Qué es una extensión? .....	5
4.	¿Qué es BlockoinEthereum y BlockoinINFURA? .....	5
5.	Conceptos básicos aplicados en la plataforma de Ethereum.....	6
6.	Instalación y configuración de extensión y ambiente de pruebas Ethereum.....	13
7.	Definición y uso de bloques (función genérica). ....	20
8.	Funciones y eventos de Exchange Ethereum Extensión (EEE).....	21
9.	Pasos para crear un CryptoToken o Criptomoneda Token. ....	32
10.	Como poner a la venta un nuevo activo o tu criptoToken (Token ERC20). ....	33
11.	Calculo de costo de transacción standard y transacción Smart contract. ....	56
12.	Tarifas de MayanCoin.org .....	60
13.	Creación tu App (Exchange) para Android en 15 minutos.....	61
14.	Token MayaCoin hace referencia al proyecto mundial MayanCoin.....	64
15.	Licenciamiento y uso de software.....	64

## 1. Introducción.

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. Su moneda digital se denomina "Ether".

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.

### ¿Cómo funciona ether?

Ether, como otras criptomonedas, utiliza un libro digital compartido donde se registran todas las transacciones. Es de acceso público, completamente transparente y muy difícil de modificar a posteriori.

Esto se denomina **blockchain** o cadena de bloques, y se construye a través del proceso de **minería**.

Los mineros son los responsables de verificar grupos de transacciones de ether para formar "bloques" y codificarlos resolviendo complejos algoritmos. Estos algoritmos pueden ser a su vez más o menos difíciles, como forma de mantener cierta constancia en el tiempo de procesamiento de los bloques (alrededor de uno cada 14 segundos).

Los nuevos bloques se enlazan entonces a la cadena de bloques anterior y el minero en cuestión recibe una **recompensa**, es decir, un número fijo de *tokens* de ether. Normalmente son 5 unidades de ether, aunque esta cifra puede verse variable si la criptomoneda continúa subiendo.

## ¿Cómo funciona Ethereum?

El *blockchain* de Ethereum es muy similar al de bitcoin, pero su lenguaje de programación les permite a los desarrolladores crear software a través del cual gestionar las transacciones y automatizar ciertos resultados. Este software se conoce como **contrato inteligente**.

Si un contrato tradicional describe los términos de una relación, un contrato inteligente se asegura de que esos términos se cumplan escribiéndolos en código. Son programas que automáticamente ejecutan el contrato una vez que las condiciones predefinidas se cumplen, eliminando el retraso y el coste que existe al ejecutar un acuerdo de manera manual.

Por poner un ejemplo sencillo, un usuario de Ethereum podría crear un contrato inteligente para enviar una cantidad establecida de ether a un amigo en una fecha determinada. Escribirían este código en la cadena de bloques y cuando el contrato se complete (es decir, cuando se alcance la fecha acordada) los ether se enviarán automáticamente.

Esta idea básica puede aplicarse a configuraciones más complejas, siendo su potencial probablemente ilimitado, con proyectos que ya han logrado un notable progreso en sectores como seguros, inmobiliarias, servicios financieros, servicios legales y micro mecenazgo.

Los contratos inteligentes también poseen varios beneficios adicionales:

1. Eliminan la figura del intermediario, ofreciendo al usuario control total y minimizando los costes extra
2. Se registran, encriptan y duplican en la cadena de bloques pública, donde todos los usuarios pueden ver la actividad del mercado
3. Eliminan el tiempo y el esfuerzo requerido en procesos manuales

Por supuesto, los contratos inteligentes son todavía un sistema muy nuevo con muchos detalles que pulir. El código se traduce literalmente, por lo que cualquier error durante la creación del contrato podría provocar resultados no deseados que no pueden modificarse.

## DApps vs contratos inteligentes

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.

## 2. ¿Qué es la programación Blockly?

Blockly es una metodología de programación visual basada en el lenguaje de programación JavaScript 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.

## 3. ¿Qué es una extensión?

Una extensión es un modulo realizado en lenguaje de programación Java dado en un archivo con extensión .AIX

En el proyecto de Blockoin.org nos basamos en crear extensión de fácil uso para el área financiera con los cuales pueden llegar a realizar aplicaciones móviles en pocos minutos sin tener un vasto conocimiento de programación.

## 4. ¿Qué es BlockoinEthereum y BlockoinINFURA?

BlockoinEthereum y BlockoinINFURA son Software (extensiones) de uso libre que incluye las siguientes soluciones tecnológicas (algoritmos) para poder crear ser usadas en la red-plataforma de Ethereum:

- CREACION DE NUEVAS CUENTAS EN LA PLATAFORMA ETHEREUM.
- CONSULTA DE BALANCE, TRANSFERENCIAS DE ACTIVOS (CRIPTOMONEDA-ETHER Y TOKENS)
- CREACION DE NUEVOS TOKEN ERC20 Y CRIPTOMONEDA-TOKEN.
- COMPILACION, CREACION, CONSULTA Y PUBLICACION DE SMART CONTRACT.
- CREACION DE TRANSACCIONES OFFLINE Y ONLINE.
- CARGA DE LLAVES PRIMARIAS Y PUBLICAS.
- CONSULTA DE EXCHANGE RATES & GAS STATION.
- AMBIENTES (REDES) DE DESARROLLO, PRUEBAS Y RED PRINCIPAL ETHEREUM.
- INCLUYE LAS 39 FUNCIONALIDADES DE INFURA.

## 5. Conceptos básicos aplicados en la plataforma de Ethereum.

### ¿Qué es un blockchain?

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**.

### ¿Qué es una dirección o cuenta dentro de la plataforma blockchain Ethereum?

Es una cadena de 42 caracteres en la plataforma Ethereum que representan un numero en base hexadecimal, en donde serán depositados o enviados los activos definidos en Ethereum. En otras plataformas de blockchain el número de caracteres de la cuenta o dirección puede ser diferente, ejemplo:

0x5d2Acdb34c279Aa6d1e94a77F7b18aB938BFb2bB

## ¿Qué es una criptomoneda?

Es una moneda digital o virtual diseñada para funcionar como medio de intercambio. Utiliza criptografía (seguridad digital) para asegurar y verificar transacciones, así como para controlar la creación de nuevas unidades de una criptomoneda particular.

## ¿Qué es Ether?

Ether es la moneda digital nativa de la plataforma de blockchain Ethereum, una plataforma descentralizada desarrollada en 2013. Un Ether es una unidad que se puede dividir en unidades más pequeñas llamadas WEI y tiene la siguiente equivalencia.

1 Ether = 1,000,000,000,000,000,000 Wei. (En expresión matemática es  $10^{18}$ ).

## ¿Qué unidades se manejan cuando se usa un Ether?

1	ether
$10^3$	finney
$10^6$	szabo
$10^9$	Shannon o GWEI este se usa para el GasPrice.
$10^{12}$	babbage
$10^{15}$	lovelace
$10^{18}$	wei

## ¿Qué es un token?

Los tokens son activos digitales que se pueden usar dentro del ecosistema de un proyecto determinado.

La principal distinción entre tokens y criptomonedas es que las primeras requieren otra plataforma blockchain (no propia) para funcionar. Ethereum es la plataforma más común para crear tokens, principalmente debido a su función de contratos inteligentes. Los tokens creados en la blockchain de Ethereum generalmente se conocen como tokens ERC-20 aunque se tienen otros tipos de token más especializados por ejemplo el token ERC-721 usado principalmente para activos colecciónables (tarjetas, uso en video-juegos, obras de arte, etc).

## ¿Qué es Gas?

El Gas es el coste que tiene el realizar una operación o un conjunto de operaciones en la red Ethereum. Estas operaciones pueden ser varias: desde realizar una transacción hasta ejecutar un contrato inteligente o crear una aplicación descentralizada.

En otras palabras, más simples, el Gas es la unidad para medir el trabajo realizado en Ethereum.

Al igual que en el mundo físico, en Ethereum también hay trabajos que cuestan más que otros: si la operación que queremos realizar requiere un mayor uso de recursos por parte de los nodos que forman la plataforma, esto hará que el Gas aumente también y viceversa.

El Gas sirve principalmente para realizar tres funciones en la plataforma Ethereum:

1.- Asigna un coste a la ejecución de tareas; En Ethereum, realizar tareas también tiene un coste. **Dependiendo de la dificultad de la tarea o de la rapidez a la que queramos que se procese esa tarea, el coste computacional de esa operación será mayor o menor** en consecuencia y el número de Gas aumentará o disminuirá en proporción.

2.- Asegura el sistema; El sistema Ethereum es un sistema seguro y esto es posible en gran medida gracias al Gas.

Al exigir que se pague una comisión por cada operación realizada, la plataforma se asegura de no procesar transacciones inservibles en la red. Esto ayuda a que la cadena de bloques sea más ligera, ya que no añadirá montones de Megabytes de información inútil a la blockchain.

Además, con el Gas, el sistema se protege también contra el ‘spam’ y el uso infinito de los bucles: instrucciones para realizar tareas repetitivas por código.

Por ejemplo, de no existir el Gas, nada evitaría que una tarea se repitiese infinitas veces colapsando el sistema y haciéndolo inservible.

3.- Recompensa a los mineros; Los mineros son sistemas independientes externos distribuidos en todo el mundo que se encargan de ejecutar las transacciones dentro de la plataforma Ethereum. Cuando realizamos una transacción o ejecutamos un smart contract, “pagamos” una cantidad determinada de Gas.

**Este Gas sirve para “pagar” a los mineros por los recursos que han utilizado** (hardware, electricidad y tiempo) **y además añadir una recompensa** por su trabajo realizado. Por lo tanto, podríamos decir que el Gas también ayuda a mantener el equilibrio de la plataforma.

Hablamos de “pagar” Gas -entre comillas- porque es lo que cuestan las operaciones. En otras palabras, se “paga” por el gasto que le cuesta a Ethereum procesar las transacciones.

### ¿Qué es Gas Precio (Gas Price)?

Una unidad de Gas corresponde a la ejecución de una instrucción, como, por ejemplo, un paso computacional.

Entonces, ¿cómo “cobran” los mineros el Gas que adquieren como recompensa?

El Gas en sí no vale nada, y por lo tanto no se puede cobrar. Para “cobrar” ese Gas usado hay que darles valor a esos recursos consumidos, es decir, valorar monetariamente el trabajo realizado por los mineros. La cantidad de Gas usado en una transacción o en un contrato inteligente tiene un precio equivalente en Ether. Este precio se llama ‘Gas price’, este normalmente lo asignan los mineros y es variable dependiendo de qué tan ocupado este la plataforma blockchain de Ethereum. Normalmente se maneja en unidades de GWEI.

### ¿Qué es Gas Limite (Gas Limit)?

Este dato indica el valor máximo de Gas que una transacción puede llegar a consumir para ser válida.

Normalmente, el software que se suele utilizar para realizar transacciones en la red Ethereum calcula de forma automática una estimación de la cantidad de Gas necesaria para llevar a cabo dicha transacción y nos lo muestra de forma inmediata.

### ¿Qué es Gas Station?

Es el lugar donde se hace referencia los valores que manejan los mineros para decidir en consenso cual es el GasPrice que se necesita para realizar los diferentes tipos de transacciones en el blockchain de Ethereum.

Dependiendo de qué cantidad se seleccione de GasPrice, será ponderado el tiempo de prioridad que se le dará a la ejecución de la transacción, normalmente se manejan tres tipos de GasPrice: TRADER: ASAP, FAST < 2 minutos, STANDARD < 5 minutos. Estos son valores promedio y pueden variar el tiempo según sea la carga de trabajo (tracciones) de la red principal de Ethereum.

<https://ethgasstation.info/>

### ¿Qué es un Exchange?

Un Exchange de criptomonedas es el punto de encuentro donde se realizan los intercambios de estas a cambio de dinero fiat o de otras criptomonedas. En estas casas de cambio online donde se genera el precio de mercado que marca el valor de las criptomonedas en base a la oferta y demanda.

### **¿Qué es Exchange Rates?**

Son las tarifas del valor de un Ether en monedas de circulación de cada país por ejemplo al día de la creación de este manual un ether tiene un valor en dólares americanos USD de \$430.94

### **¿Qué es un Smart Contract?**

En ethereum un Smart contract es un programa en lenguaje de programación llamado Solidity que se encuentra dentro del blockchain Ethereum, el cual tiene instrucciones para ser ejecutadas automáticamente basadas en reglas pre-establecidas, esta propiedad hace una evolución en todos los blockchain existentes ya que se pueden crear infinidad de Smart contract adaptados a los diferentes sectores privados y públicos haciendo más eficientes las operaciones de las empresas o en su caso adaptarlos a sistemas o programas de todo tipo.

### **¿Qué es “nonce”?**

Es un contador “numero hexadecimal” incremental que lleva la cuenta de las transacciones en cada dirección o cuenta creada en Ethereum.

### **¿Qué es una transacción?**

Es la ejecución o transferencia de algún tipo de activo no tangible que se le puede dar un valor pre-establecido dentro del sistema de Ethereum y que posteriormente puede ser cambiado a un valor tangible para una empresa o persona.

### **¿Qué es txHash?**

Es un numero hexadecimal que ayuda a rastrear el resultado a detalle de cada transacción.

### **¿Qué tipos de transacciones hay?**

Se tienen dos tipos, un es la transacción “offline” esta crea sin la necesidad de tener conexión a la red principal de Ethereum se pueden almacenar hasta que se escoga conectarse a la red de Ethereum y liberar la transacción, tienen la ventaja de seguridad ya que toda la transacción se procesa fuera de línea lo que impide cualquier anomalía que pudiera estar en la red de conexión. La otra transacción es la “online” la cual siempre se necesita estar conectado a internet con las ventajas y desventajas de seguridad que trae implícita la conexión.

### **¿Qué es INFURA.io?**

Infura.io es una plataforma que proporciona un conjunto de herramientas e infraestructuras que permiten a los desarrolladores llevar fácilmente su aplicación blockchain de la prueba, a la implementación a escala, con acceso simple y confiable a Ethereum e IPFS.

## ¿Qué es una dirección en la red Ethereum?

Una dirección o cuenta está compuesta por tres partes, la dirección, la llave pública y la llave privada, estas dos llaves son una cadena de números y caracteres en formato hexadecimal que se ocupan para enviar y recibir (activos) o ether (monedad digital).

La llave primaria nunca debe ser compartida con nadie ya que es la que autoriza la liberación del saldo (firma las transacciones) que se tenga en la cuenta.

La llave publica es conocida por todo el público y es compartida a cualquier persona ya que es la referencia para confirmar que la transacción es correcta tanto el valor como a quien se envía.

Ejemplos:

```
{  
  "private":  
    "429a043ea6393b358d3542ff2aab9338b9c0ed928e35ec0aed630b93adb14a1c",  
  "public":  
    "049b4b7e72701a09d3ee09165bba460f2549494a9d9fd7a95aaac57c2827eac162fd9e105b  
    2461cd6594ca8ca6a8daf10fe982f918be1b0060c87db9cfbcd289a8",  
  "address": "88ab6dcecc3603c7042f4334fc06db8e8d7062d5"  
}
```

## ¿Qué es MayanCoin.org?

Es una plataforma en consolidar criptomonedas, creamos las primeras direcciones hibridas en el mundo financiero centralizado y descentralizado, usando la tecnología Blockly que es una forma de programación visual sin necesidad de tener conocimientos avanzados de programación basado en extensiones funcionales. Ver nuestro WhitePaper en [www.MayanCoin.org](http://www.MayanCoin.org)

## ¿Qué tipos de redes para pruebas y red principal en el blockchain Ethereum?

<https://mainnet.infura.io>

Red principal, red de producción donde se realizan todas las transacciones en tiempo real.

<https://ropsten.infura.io>

Red de prueba Ropsten permite que los desarrollos de blockchain prueben su trabajo en un entorno en vivo, pero sin la necesidad de tokens ETH reales, con algoritmo llamado (*Proof-of-Work*).

<https://kovan.infura.io>

Red de prueba Kovan, que a diferencia de su predecesor ropsten utiliza un algoritmo llamado Prueba de Autoridad (Proof of Authority).

<https://rinkeby.infura.io>

Red de prueba, utiliza un algoritmo llamado Prueba de Autoridad (Proof of Authority). Una de las más utilizadas para pruebas.

<https://goerli.infura.io>

Goerli es una red de prueba pública para Ethereum que el motor de consenso de POA (Prueba de autoridad) es compatible con varios clientes. A prueba de spam.

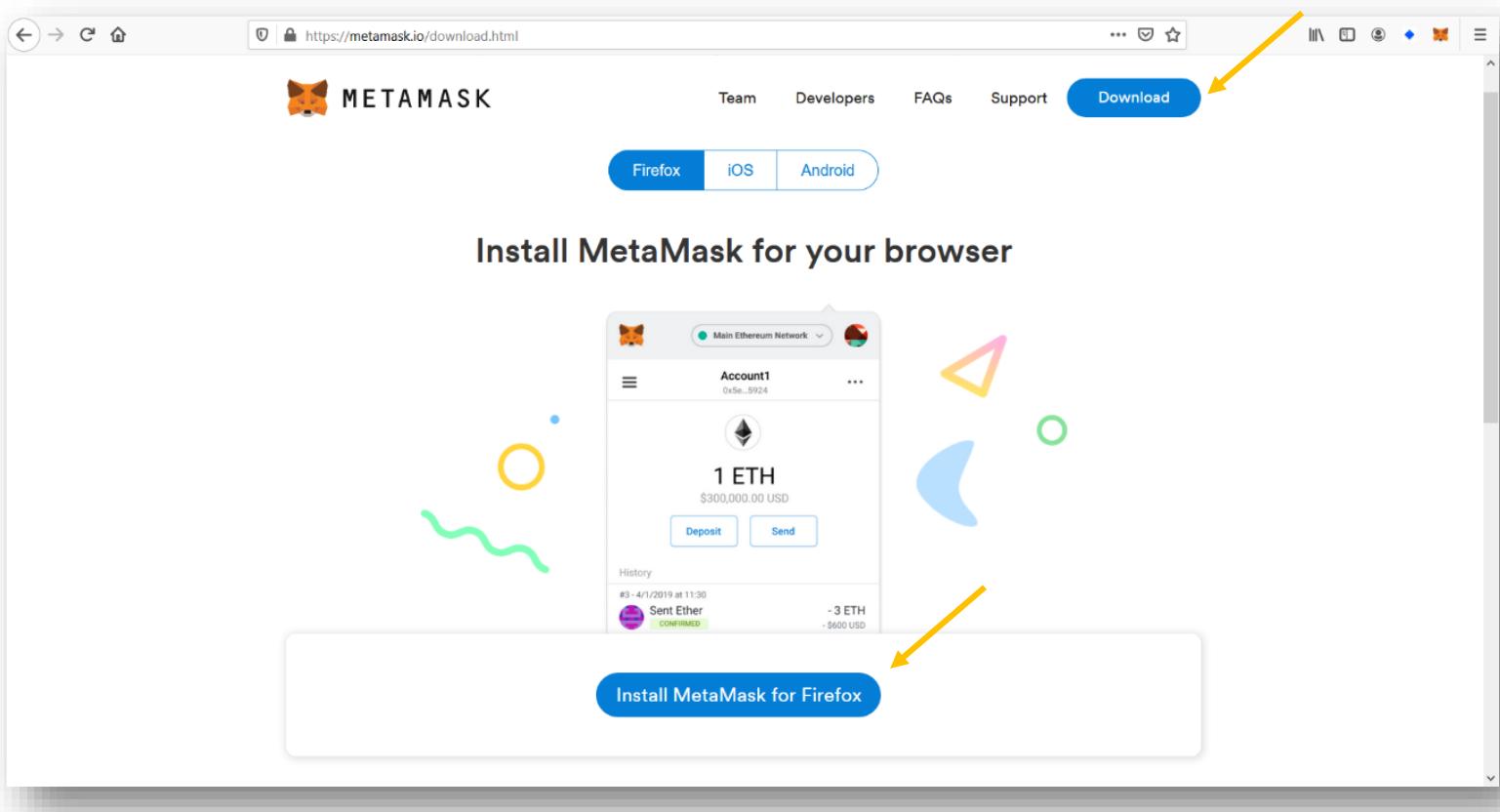
En total se tienen 5 redes basadas en el blockchain Ethereum, una de producción o principal y cuatro para pruebas, en seguida la que usaremos será la red de Rinkeby para instalar nuestro ambiente de pruebas.

## 6. Instalación y configuración de extensión y ambiente de pruebas Ethereum.

Necesitamos primero un ambiente de pruebas. Para aprender a usar las diferentes funcionalidades de las dos extensiones que serán usadas para crear, enviar, publicar, revisar y obtener datos de todas las transacciones que realicemos en la plataforma de Ethereum.

Lo primero que tenemos que realizar es la instalación de nuestro ambiente de pruebas. Tendremos que instalar la aplicación de **METAMASK** este es un monedero para crear, importar cuentas de Ethereum y manejar transacciones desde el navegador en nuestro navegador de internet que usaremos es Mozilla pudiendo ser usado también en Chrome.

Vamos al sitio oficial de [www.metamask.io](https://www.metamask.io) y en el botón de “Down load” damos click.

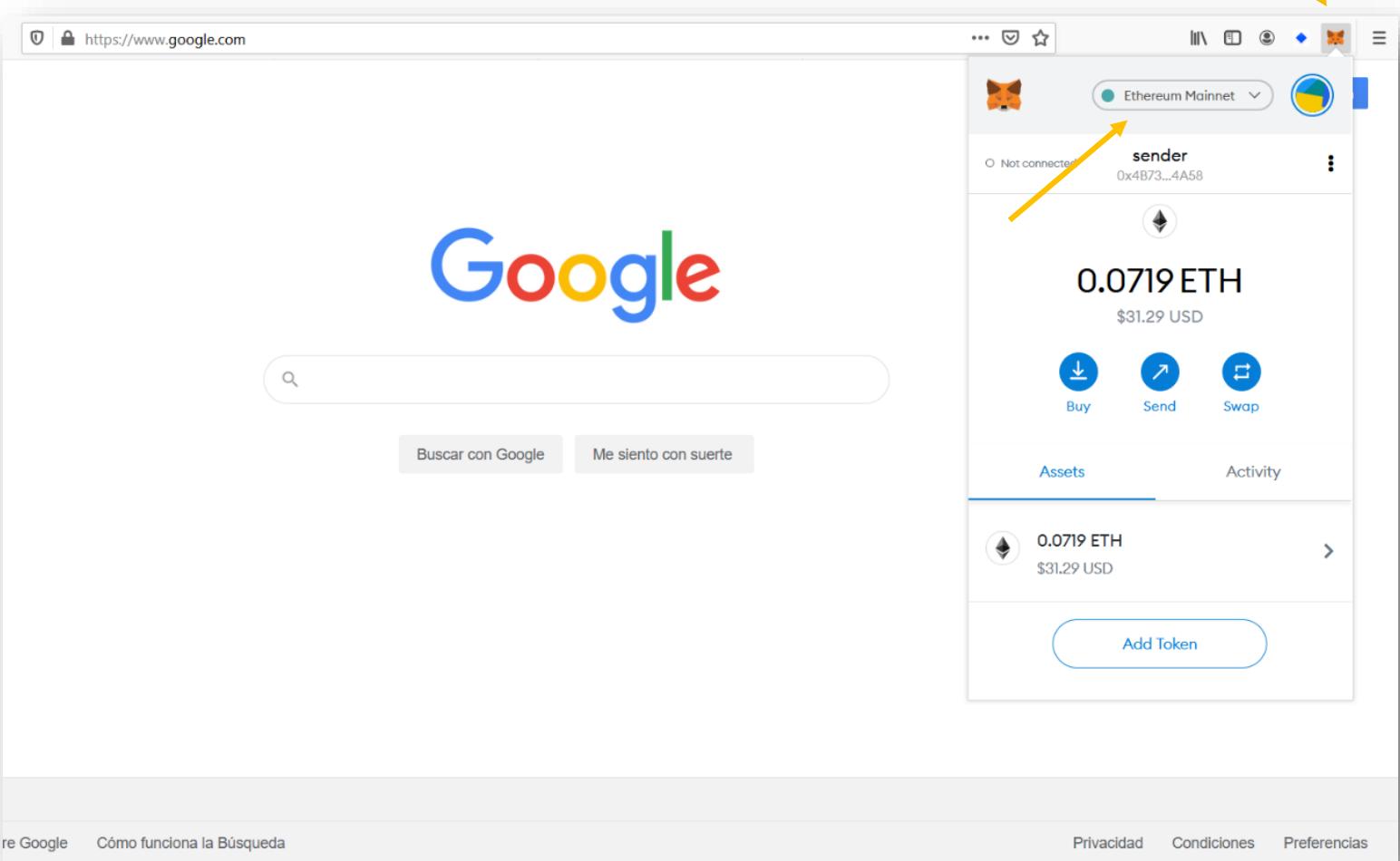


Posteriormente damos click en el botón de “Install MetaMask for Firefox” y damos aceptamos las opciones por default. Después de la instalación nos aparecerá un ícono en la parte superior derecha en donde veremos ya instalado el software de Metamask.

Damos click en el ícono de Metamask y nos aparecerá desplegado la opción de importar una cuenta que ya tengamos o crear una nueva. En nuestro caso importaremos una cuenta que ya tenemos funcionando usando el método de “restaurar a través de semilla”. En caso de no

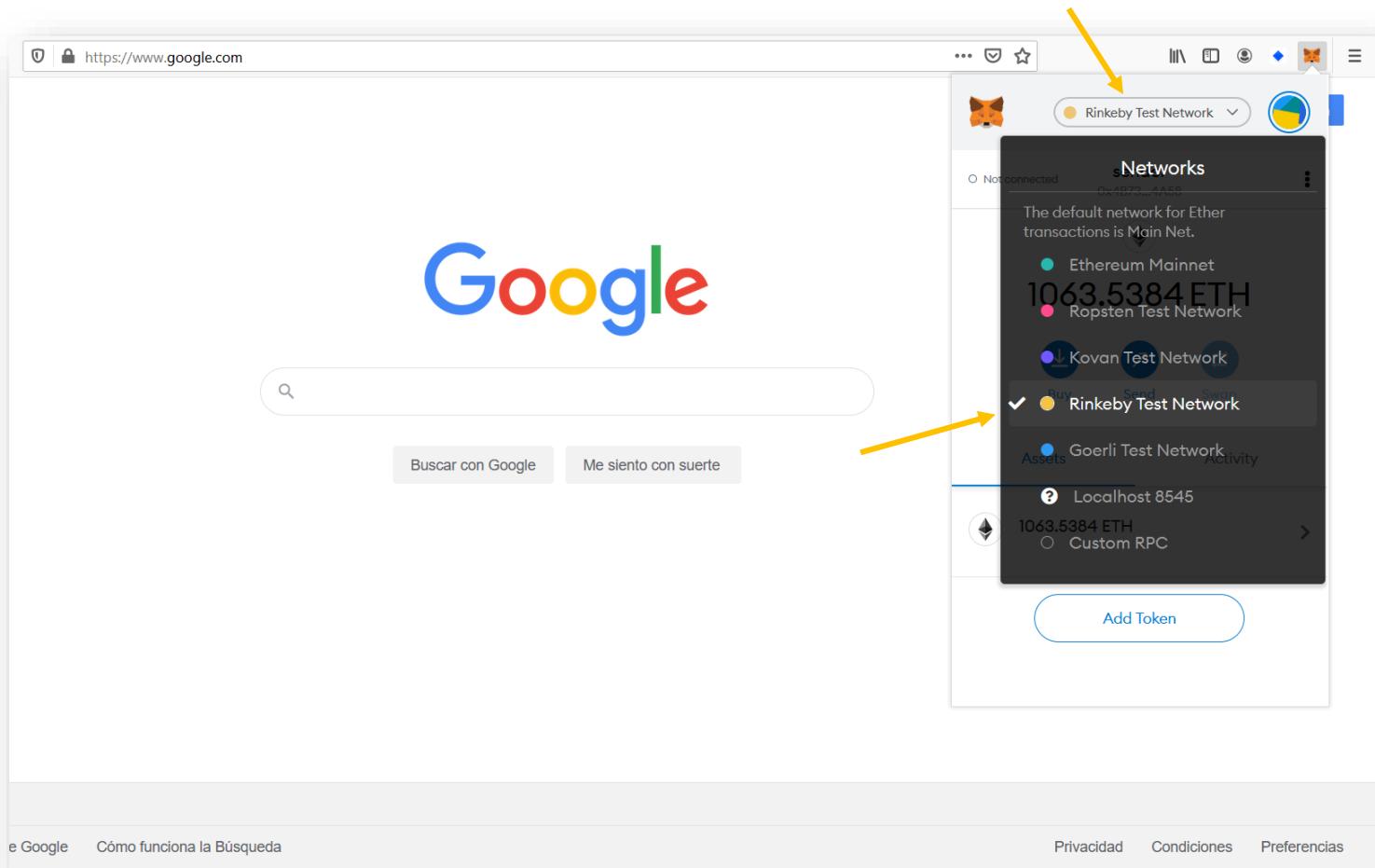
tener simplemente damos crear una nueva cuenta. En este caso nos pedirá para cualquiera de los dos casos un “password” para ser asignado.

Posteriormente entramos con el “password” y podremos revisar que saldo tenemos, lógicamente si es nueva el saldo será nulo.



Ahora procedemos a revisar cómo vamos a depositar unos ethers (monedas digitales) a nuestra cuenta de prueba Rinkeby.

En la parte superior tenemos la opción de elegir el tipo de red que usaremos, por default al entrar te coloca en la “Ethereum Mainnet”, sin embargo, al dar click podemos revisar todas las redes opcionales que podemos escoger, en nuestro caso seleccionamos la red de Rinkeby.



El siguiente paso es depositar ethers a nuestra cuenta de prueba referenciada a la red Rinkery.

**NOTA IMPORTANTE:** Los ethers (moneda digital) que depositemos en nuestra cuenta referenciada a la red Rinkery de prueba no tienen ningún valor en el mercado de criptomonedas, solamente nos servirán para realizar pruebas de software. Siempre revisar en qué red estamos trabajando para evitar errores en las transacciones.

Para poder conseguir ether de prueba necesitamos realizar el siguiente procedimiento.

En cualquier cuenta de twitter que se tenga tendremos que entrar a twitter y crear un comentario que únicamente incluya la cuenta y/o dirección que tenemos en Metamask posteriormente tendremos que copiar el enlace del comentario ya que tengamos esto nos vamos a la siguiente liga para fondear nuestra cuenta.

<https://faucet.rinkeby.io/>

Creamos comentario en twitter y copiamos la liga del comentario.

A screenshot of a Twitter tweet page. The URL in the browser's address bar is highlighted with a yellow arrow. The tweet content is as follows:

AGAVE  
@oomissede

0x88ac6dcecc3603c7042f4334fc06db8e8d7062d5

Traducir Tweet

2:22 a. m. · 6 nov. 2020 · Twitter Web App

Ver actividad del Tweet

Reply Retweet Like Share

En el sitio <https://faucet.rinkeby.io/> copiamos la liga de twitter y en el botón “Give me Ether” escogemos la cantidad deseada.

A screenshot of the Rinkeby Authenticated Faucet website. A yellow arrow points to the input field where a Twitter URL has been pasted. Another yellow arrow points to the dropdown menu labeled "Give me Ether" which shows three options:

- 3 Ethers / 8 hours
- 7.5 Ethers / 1 day
- 18.75 Ethers / 3 days

The main content area includes:

Rinkeby Authenticated Faucet

https://twitter.com/oomissede/status/134626802388750337

Give me Ether ▾

3 Ethers / 8 hours

7.5 Ethers / 1 day

18.75 Ethers / 3 days

How does this work?

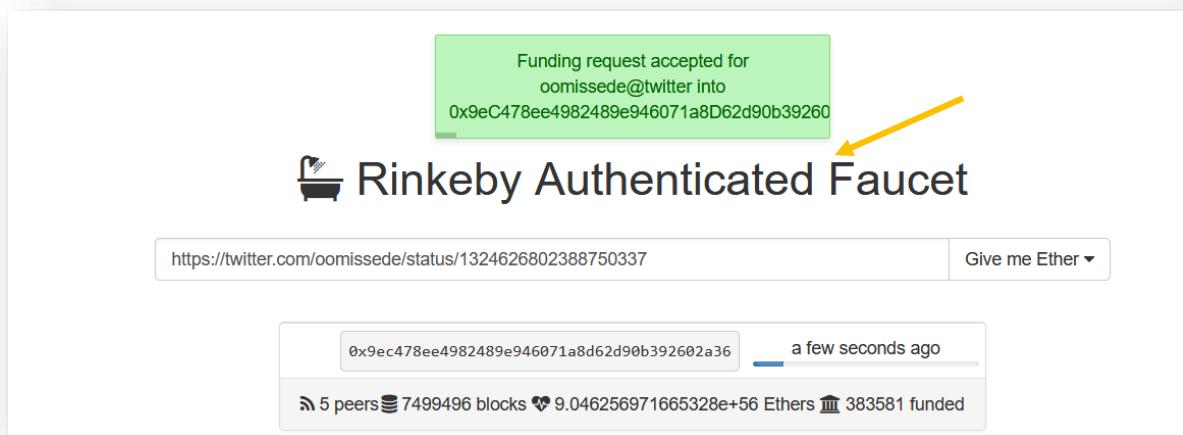
This Ether faucet is running on the Rinkeby network. To prevent malicious actors from exhausting all available funds or accumulating enough Ether to mount long running spam attacks, requests are tied to common 3rd party social network accounts. Anyone having a Twitter or Facebook account may request funds within the permitted limits.

To request funds via Twitter, make a [tweet](#) with your Ethereum address pasted into the contents (surrounding text doesn't matter).  
Copy-paste the [tweets URL](#) into the above input box and fire away!

To request funds via Facebook, publish a new [public post](#) with your Ethereum address embedded into the content (surrounding text doesn't matter).  
Copy-paste the [posts URL](#) into the above input box and fire away!

You can track the current pending requests below the input field to see how much you have to wait until your turn comes.  
*The faucet is running invisible reCaptcha protection against bots.*

Por ultimo si todo es correcto nos dará un anuncio que fue aceptado el depósito y dependiendo de la carga de trabajo del sistema de la red Rinkeby el deposito será en unos minutos o puede tomarse más tiempo.



Ahora que ya tenemos ethers en nuestra cuenta podemos empezar a realizar las pruebas en la plataforma de Ethereum.

Tenemos dos extensiones para interactuar con el blockchain de Ethereum.

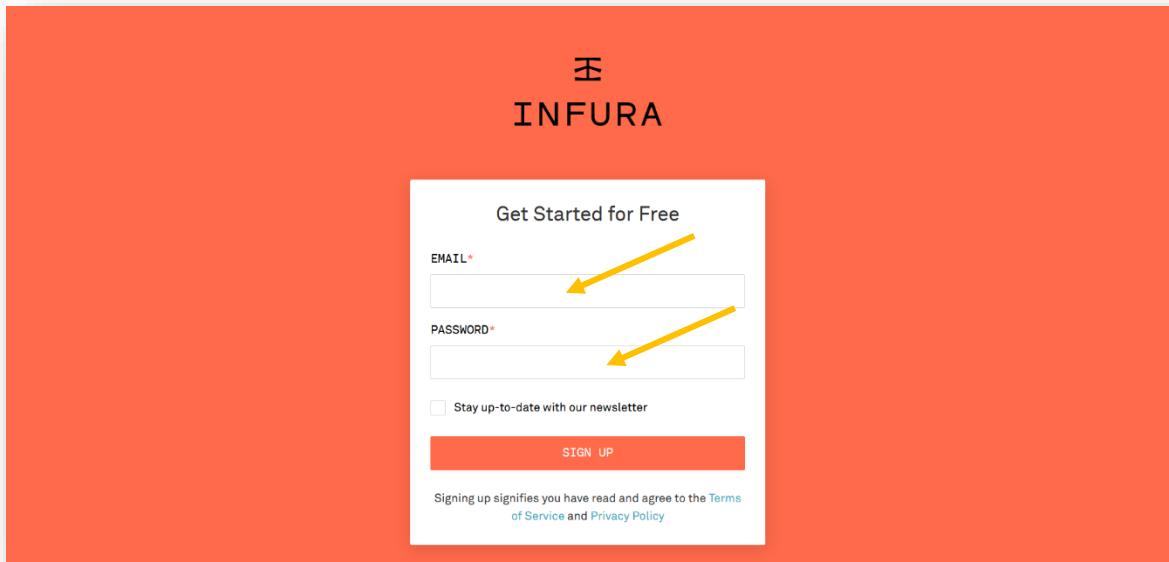
La extensión **MayanCoin.BlockoinETHEREUM.aix** contiene las funciones para realizar las siguientes funciones:

- Creación de nuevas cuentas (direcciones) en blockchain Ethereum (privateKey, publicKey, Address).
- Almacenamiento de datos cuentas (direcciones) en archivos binarios.
- Importar cuentas (direcciones) de archivos binarios.
- Obtención de cuenta (dirección) a través de privateKey.
- Creación y envío de transacciones entre cuentas (direcciones) “online”.
- Creación, firma y envío de transacciones PushRaw offline.
- Consulta de detalles de transacciones Tx.
- Verificación de saldo en direcciones y smart contracts.
- Compilador para Smart contracts.
- Creación, publicación y ejecución de smart contracts.
- Creación, publicación y ejecución de Token ERC20 (criptomoneda token).
- Obtención de código ABI de smart contract.
- Verificación de conexión red.
- Consulta de valor de ether en el mercado de criptomonedas en cualquier país del mundo (moneda nativa del país) – Exchange Rates.
- Consulta de GasPrice.
- Consulta de “nonce” de una cuenta específica.

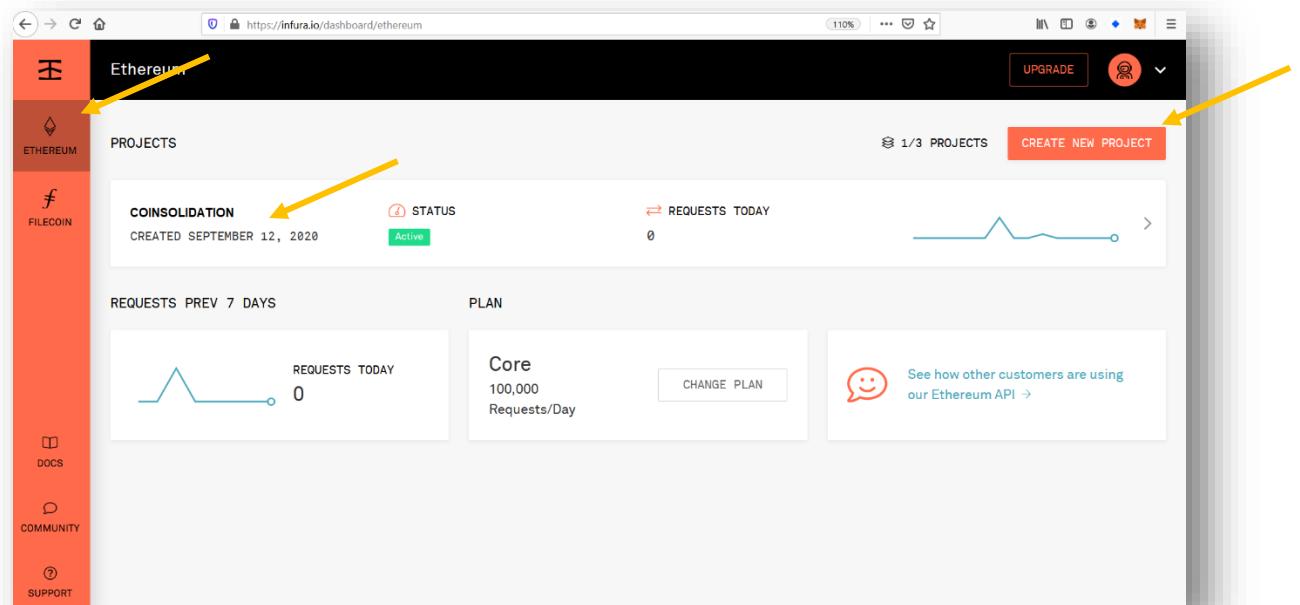
La extensión **coinsoidation.BlockoinINFURA.aix** nos proporciona las 40 funcionalidades de la plataforma Infura.io, Para ver detalle consultar la documentación json-rpc directo en <https://infura.io/docs>

Para poder usar la extensión INFURA se necesita crear una cuenta en el sitio de infura.io ya que necesitamos un API KEY para poder enviar consultas a la red de Ethereum, así como para poder usar las redes de pruebas de Ethereum.

Como abrimos una cuenta es de una forma simple como se muestra a continuación.



Ya que es creado la cuenta entramos y podremos tener el API KEY de las diferentes redes. Nos vamos a la parte izquierda superior y le damos "click" en Ethereum, posteriormente nos aparecerá los proyectos, creamos un proyecto nuevo y nos introducimos a dicho proyecto.

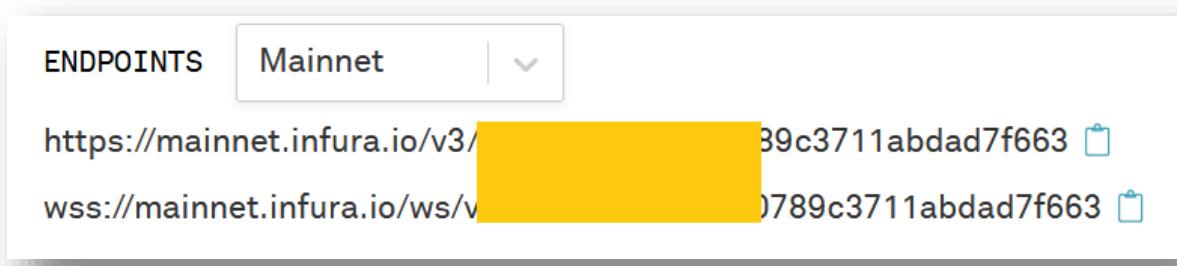


Dentro del proyecto nos aparecerán los datos de la API KEY y las diferentes redes que podemos utilizar.

Dentro del proyecto podremos revisar el API KEY (PROJECT ID) y seleccionar en cual red trabajaremos o las ligas completas de los ENDPOINTS a elegir.

The screenshot shows the Infura.io dashboard for an Ethereum project. On the left sidebar, there are links for COINSOLIDATION, ETHEREUM (selected), FILECOIN, DOCS, COMMUNITY, and SUPPORT. The main area is titled "COINSOLIDATION" and has tabs for "REQUESTS" (selected) and "SETTINGS". Under "SETTINGS", there is a "KEYS" section with fields for "PROJECT ID" (containing a long string of characters) and "PROJECT SECRET" (containing a long string of characters). Below these are dropdown menus for "ENDPOINTS" (set to "Mainnet") and "PROJECT SEC" (checkboxes for "Require project secret for all requests" and "Require JWT for all requests"). Arrows point from the text descriptions to the corresponding fields in the screenshot.

Por ejemplo, para usar la red principal de Ethereum tomaremos la siguiente liga:

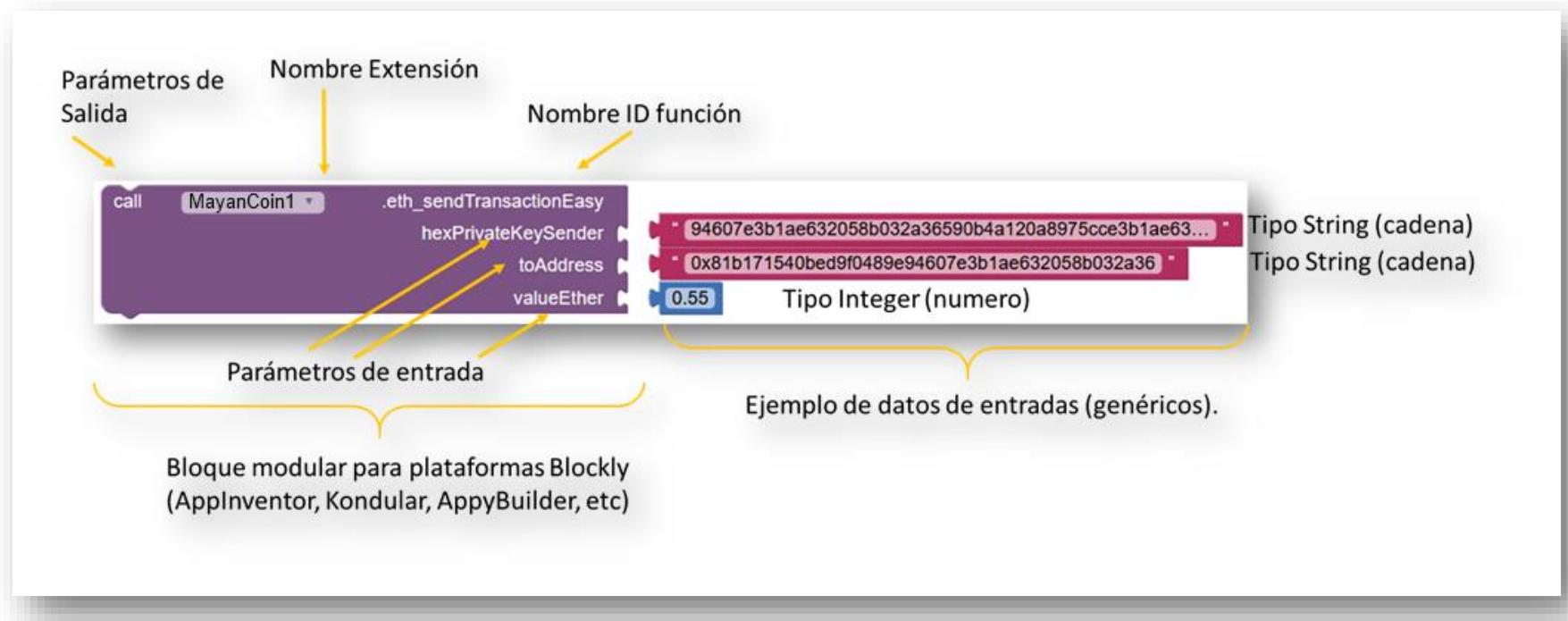


**NOTA:** Las extensiones han sido probadas en los sistemas de AppInventor, Kondular, Thunkable y AppyBuilder.

## 7. Definición y uso de bloques (función genérica).

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 (número 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.

## 8. Funciones y eventos de Exchange Ethereum Extensión (EEE).

\*\*\*Red de pruebas que usaremos **Rinkeby**, como urlNetwork, cuando se quiera realizar transacciones reales solo se tiene que cambiar la red urlNetwork por **mainnet**.

Bloque para revisar conexión a internet – (**CheckInternetConnection**).

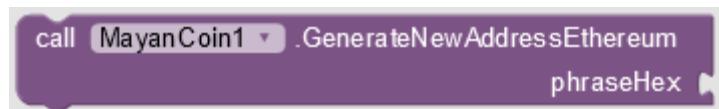


Parámetros de entrada: No aplica.

Parámetros de salida: Regresa “True” si tiene conexión o regresa “Falso” si no hay conexión.

Descripción: Bloque para revisar conexión a internet y poder enviar datos (transacciones).

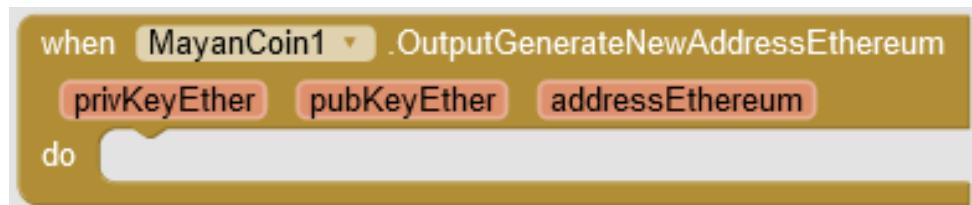
Bloque para genera una nueva dirección “Offline” – (**GenerateNewAddressEthereum**).



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

Parámetros de salida: Evento (**OutputGenerateNewAddressEthereum**).

Outputs: **PrivateKey<String>**, **PublicKey<String>**, **addressEthereum<String>**.



Descripción: Crea una nueva dirección (cuenta) de ethereum basada en una frase o secuencia de números. Se puede crear la nueva dirección sin necesidad de tener conexión a red o internet – “Offline”.

Bloque para genera una nueva dirección “Online” – (**GenerateNewAddressEthereumAPI**).



Parámetros de entrada: No aplica.

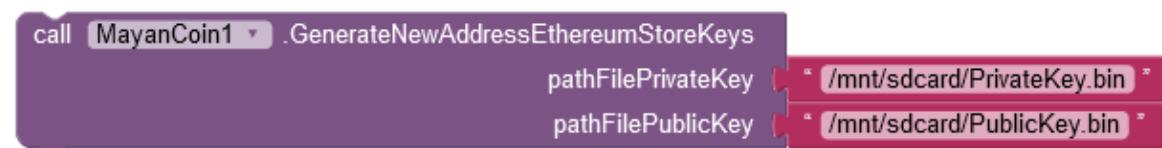
Parámetros de salida: Regresa en formato JSON datos; privateKey, publicKey, address.

Output ejemplo:

```
{  
"private": "9ab4b2fba728a55643414f26adc04eea080740860cbe39b13fe4acb43dbb9f83",  
"public":  
"0421d559aa98d6fc77688ae9f19b3dc502c05f0b7f70bbe924cb712d6243a489695b1c1ee03993b3b6d97277  
97e780677a5469800b4d98374bdb910ed99fa2b5c8",  
"address": "92a2f157d5aec3fa79f92995fea148616d82c5ef"  
}
```

Descripción: Crea una nueva dirección (cuenta) de ethereum. Se necesita tener acceso o conexión a internet ya que la generación es a través de servicio REST API – “Online”.

Bloque para genera una nueva dirección “Offline” y guarda las llaves pública y privada en archivos binarios – (**GenerateNewAddressEthereumStoreKeys**).



Parámetros de entrada: **pathFilePrivateKey <String>**, **pathFilePublicKey<String>**.

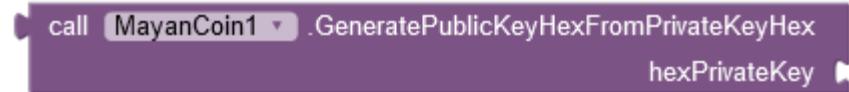
Parámetros de salida: Evento (**OutputGenerateNewAddressEthereumStoreKeys**).

Outputs: **addressEthereum<String>**, **privateKeyECC<String>**, **publicKeyECC<String>**, **privateKeyHex<String>**, **publicKeyHex<String>**.



Descripción: Crea una nueva dirección (cuenta) de ethereum aleatoria y almacena las llaves pública y privada en archivos binario que servirán para la importación y exportación de los datos de la cuenta. Se puede crear la nueva dirección sin necesidad de tener conexión a red o internet – “Offline”.

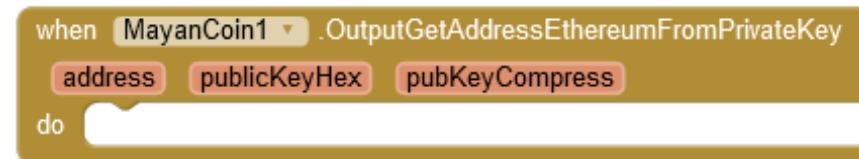
Bloque para genera llave publica – (**GeneratePublicKeyHexFromPrivateKeyHex**).



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

Parámetros de salida: Evento (**OutputGeneratePublicKeyHexFromPrivateKeyHex**).

Outputs: **address<String>, publicKeyHex<String>**.



Descripción: Crea la llave publica basada en la entrada de una llave privada.

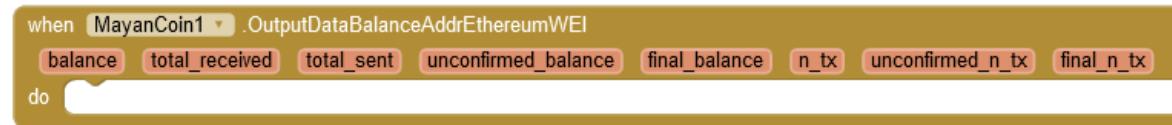
Bloque para obtener el balance de una dirección – (**GetBalanceAddrEthereum**).



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

Parámetros de salida: Evento (**OutputGeneratePublicKeyHexFromPrivateKeyHex**).

Outputs: **balance<String>, total\_received<String>, total\_sent<String>, unconfirmed\_balance<String>, final\_balance<String>, n\_tx<String>, unconfirmed\_n\_tx<String>, final\_n\_tx<String>**.



Descripción: Muestra balance y datos detallados de la cuenta (dirección).

Bloque para revisar si el móvil tiene activada la interface de red – (GetDataNetworkConnection).

call MayanCoin1 .GetDataNetworkConnection

Parámetros de entrada: No aplica.

Parámetros de salida: Evento (OutputGeneratePublicKeyHexFromPrivateKeyHex).

Outputs: interfacename<String>, isconnected<String>.

when MayanCoin1 .OutputGetDataNetworkConnection

interfacename isconnected

do

Descripción: Muestra el nombre de la interface del móvil y entrega si está activado o no la interface.

Bloque para firmar transacción “Offline” – (SignerGenericPushRawTransactionOffline)

call MayanCoin1 .SignerGenericPushRawTransactionOffline  
urlNetwork https://rinkeby.infura.io/v3/XXXXXXXXXXXXXXXXXXXX  
hexPrivateKeySender C299C9e722773d2A04B401e831a6DC8C299C9e722773d2A0...  
nonceNumber 3456  
gasPrice 25000000000  
gasLimit 21000  
toAddress 0x776BB566dC299C9e722773d2A04B401e831a6DC8  
valueWei 0

Dependencia(s) obligatoria(s): Bloque (eth\_getTransactionCount), Bloque (eth\_SendRawTransactionInfura).

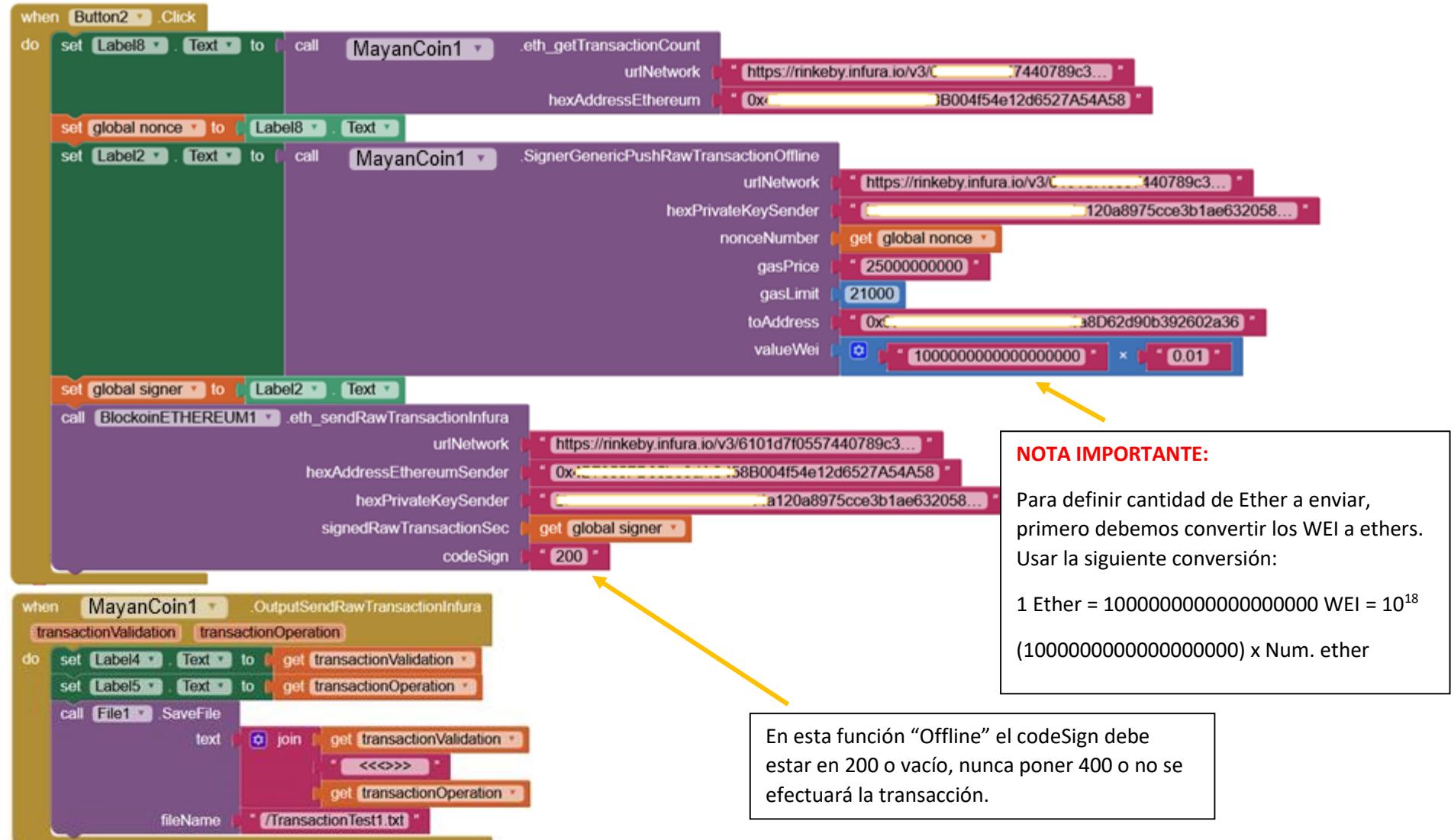
Parámetros de entrada: urlNetwork <String>, hexPrivateKeySender <String>, nonceNumber <String>, gasPrice <String>, gasLimit <Integer>, toAddress <String>, valueWEI <Integer>.

Parámetros de salida:

Outputs: Transacción firmada para enviar. <String>.

Descripción: Prepara una nueva transacción para ser enviada (cifrada y firmada). Esta puede ser procesada sin tener conexión a red o internet – “Offline”.

Ejemplo de uso completo con dependencias del bloque (`SignerGenericPushRawTransactionOffline`).



#### NOTA IMPORTANTE:

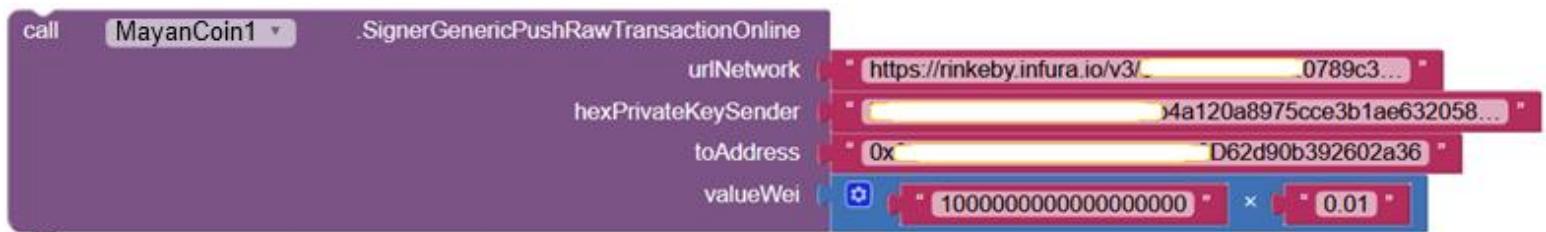
Para definir cantidad de Ether a enviar, primero debemos convertir los WEI a ethers. Usar la siguiente conversión:

$$1 \text{ Ether} = 1000000000000000000 \text{ WEI} = 10^{18}$$

$$(1000000000000000000) \times \text{Num. ether}$$

En esta función “Offline” el codeSign debe estar en 200 o vacío, nunca poner 400 o no se efectuará la transacción.

Bloque para firmar transacción “Online” – (SignerGenericPushRawTransactionOnline).

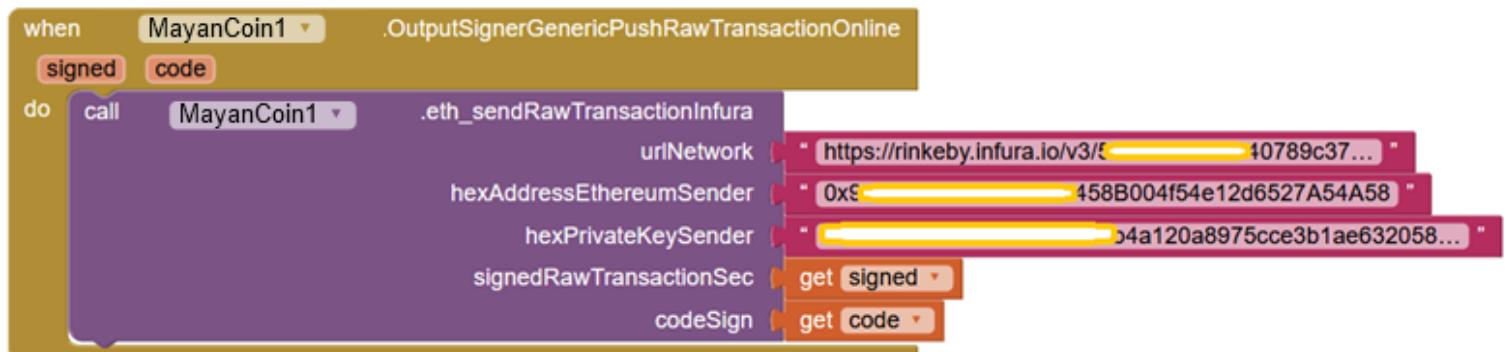


Dependencia(s) obligatoria(s): Bloque (eth\_SendRawTransactionInfura).

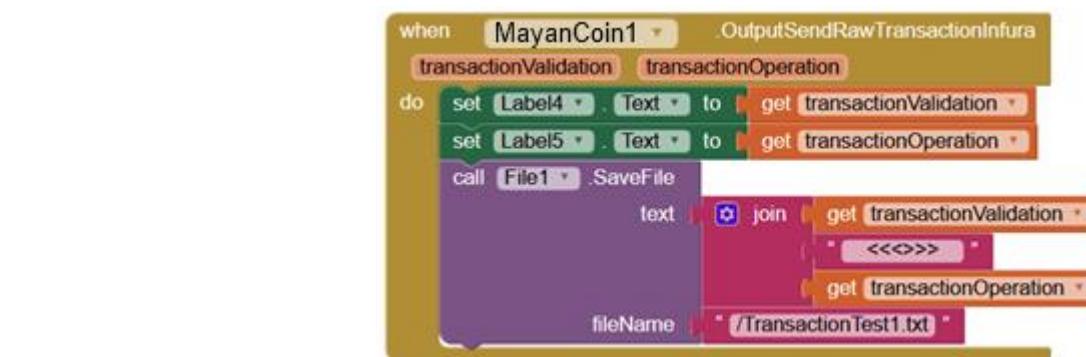
Parámetros de entrada: urlNetwork <String>, hexPrivateKeySender <String>, toAddress <String>, valueWEI <Integer>.

Parámetros de salida: Eventos usar en el siguiente orden (OutputSignerGenericPushRawTransactionOnline) y (OutputSendRawTransactionInfura).

Outputs: signed<String>, code<String>.



Outputs bloque eth\_sendRawTransactionInfura: transactionValidation<String>, transactionOperation<String>.



Descripción: Prepara una nueva transacción para ser enviada (cifrada y firmada). Se necesita tener conexión a red o internet – “Online”.

Bloque para compilar Smart contract “Online” – (**SmartContractCompilerSolidity**).



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

Parámetros de salida: Muestra el smart contract compilado, esta función nos sirve para comprobar si está bien escrito antes de publicar un Smart contract en la red de Ethereum.

Outputs: **Código compilado**.

El Smart contract debe estar en un archivo con formato JSON.

Ejemplo de Smart contrato básico en lenguaje Solidity.

```
pragma solidity ^0.5.0;

contract mortal {
address owner;
function mortal() { owner = msg.sender; }
function kill() { if (msg.sender == owner) suicide(owner); }
contract greeter is mortal {
string greeting;
function greeter(string _greeting) public { greeting = _greeting; }
function greet() constant returns (string) {return greeting;}
```

Ejemplo de anterior Smart contract en Formato JSON con comentarios.

```
# Check solidity compilation via non-published test
# Using "greeter" contract solidity example, the "hello world" of
Ethereum.
```

Archivo: **smartcontract.json**

```
{
  "solidity": "contract mortal {\n    /* Define variable owner of the type\n     address */\n    address owner;\n    /* this function is executed at\n     initialization and sets the owner of the contract */\n    function\nmortal() { owner = msg.sender; }\n    /* Function to recover the funds on\n     the contract */\n    function kill() { if (msg.sender == owner)\n      suicide(owner); }\n}\n\ncontract greeter is mortal {\n    /* define\n     variable greeting of the type string */\n    string greeting;\n    /*\n     this runs when the contract is executed */\n    function greeter(string\n_greeting) public {\n        greeting = _greeting;\n    }\n    /* main
```

```
function */\n      function greet() constant returns (string) {\nreturn greeting;\n      }\n    },\n  \"params\": ["Hello MayanCoin Test"]\n}
```

**NOTA IMPORTANTE:** El formato JSON siempre cada final de línea debe estar colocado un salto de línea “`\n`”.

Ejemplo de salida de Smart contract compilado.

```
[\n{\n  \"name\": \"\\u003cstdin\\u003e:greeter\", \n  \"solidity\": \"contract mortal {\\n    /* Define variable owner of the type address*/\\n    address owner;\\n    /* this function is executed at initialization and sets the owner of the contract */\\n    function mortal() { owner = msg.sender; }\\n    /* Function to recover the funds on the contract */\\n    function kill() { if (msg.sender == owner) suicide(owner); }\\n}\\ncontract greeter is mortal {\\n    /* define variable greeting of the type string */\\n    string greeting;\\n    /* this runs when the contract is executed */\\n    function greeter(string _greeting) public {\\n        greeting = _greeting;\\n    }\\n}\\n    /* main function */\\n    function greet() constant returns (string) {\\n        return greeting;\\n    }\\n}\", \n  \"bin\": \"606060405260405161023e38038061023e8339810160405280510160008054600160a060020a031916331790558060016000509080519060200190828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f10609f57805160ff19168380011785555b50608e9291505b8082111560cc57600081558301607d565b50505061016e806100d06000396000f35b828001600101855582156076579182015b8281115607657825182600050559160200191906001019060b0565b509056606060405260e060020a600035046341c0e1b58114610026578063cfae321714610068575b005b6100246000543373fffffffffffffffffffff908116911614156101375760005473fff\n  ffffffffffffff16ff5b6100c9600060609081526001805460a06020601f600260001961010086881615020190941693909304928301819004028101604052608028152929190828280156101645780601f1061013957610100808354040283529160200191610164565b60405180806020018281038252838181518152602001915080519060200190808383829060006004602084601f0104600f02600301f150905090810190601f1680156101295780820380516001836020036101000a031916815260200191505b509250505060405180910390f35b565b820191906000526020600020905b81548152906001019060200180831161014757829003601f168201915b5050505090509056\", \n  \"abi\": [\n    {\n      \"constant\": false,\n      \"inputs\": [],\n      \"name\": \"kill\", \n      \"outputs\": [],\n      \"type\": \"function\"\n    },\n    {\n      \"constant\": true,\n      \"inputs\": [],\n      \"name\": \"greet\", \n      \"outputs\": [\n
```

```

{
    "name": "",
    "type": "string"
}
],
"type": "function"
},
{
"inputs": [
{
    "name": "_greeting",
    "type": "string"
}
],
"type": "constructor"
}
,
"params": [
"Hello MayanCoin Test"
]
},
{
"name": "mortal",
"solidity": "contract mortal {\n    /* Define variable owner of the\n    type address*/\n    address owner;\n    /* this function is executed at\n    initialization and sets the owner of the contract */\n    function\nmortal() { owner = msg.sender; }\n    /* Function to recover the funds on\n    the contract */\n    function kill() { if (msg.sender == owner)\nsuicide(owner); }\n}\ncontract greeter is mortal {\n    /* define\n    variable greeting of the type string */\n    string greeting;\n    /*\n    this runs when the contract is executed */\n    function greeter(string _greeting) public {\n        greeting = _greeting;\n    }\n    /* main\n    function */\n    function greet() constant returns (string) {\n        return greeting;\n    }\n}",
"bin": "606060405260008054600160a060020a03191633179055605c8060226000396000f36060\n60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffffffff\nfffffffffffffffffffff90811691161415605a5760005473ffffffffffff\nfffffffffffffffffffff16ff5b56",
"abi": [
{
    "constant": false,
    "inputs": [],
    "name": "kill",
    "outputs": [],
    "type": "function"
},
{
    "inputs": [],
    "type": "constructor"
}
],
"params": [
"Hello MayanCoin Test"
]
}
]

```

Bloque para compilar, crear y publicar Token ERC20 – (**SmartContractCreateTokenERC20v1**)



Dependencia(s) obligatoria(s): Bloque (**CreateTestingFie**). **IMPORTANTE**: Primero se debe usar este bloque para asegurarse que la ruta (path) es correcta, esto es debido a que si no se da una ruta valida en el Bloque (**SmartContractCreateTokenERC20v1**) no se ejecutara la creación de Token ya que el parámetro de entrada “nameFileSolidityJSON” se usa para crear un archivo temporal.

Parámetros de entrada: **tokenName <String>**, **tokenSymbol <String>**, **totalAmount <String>**, **numberDecimal <String>**, **privateKeyHex <Integer>**, **nameFileSolidityJSON <String>** Este de archive es la ruta válida para crear un archivo temporal, se debe asegurar que la ruta es válida para probar que se crea el archivo se puede usar el Bloque (**CreateTestingFile**) después de usarla revisar que se haya creado exitosamente.

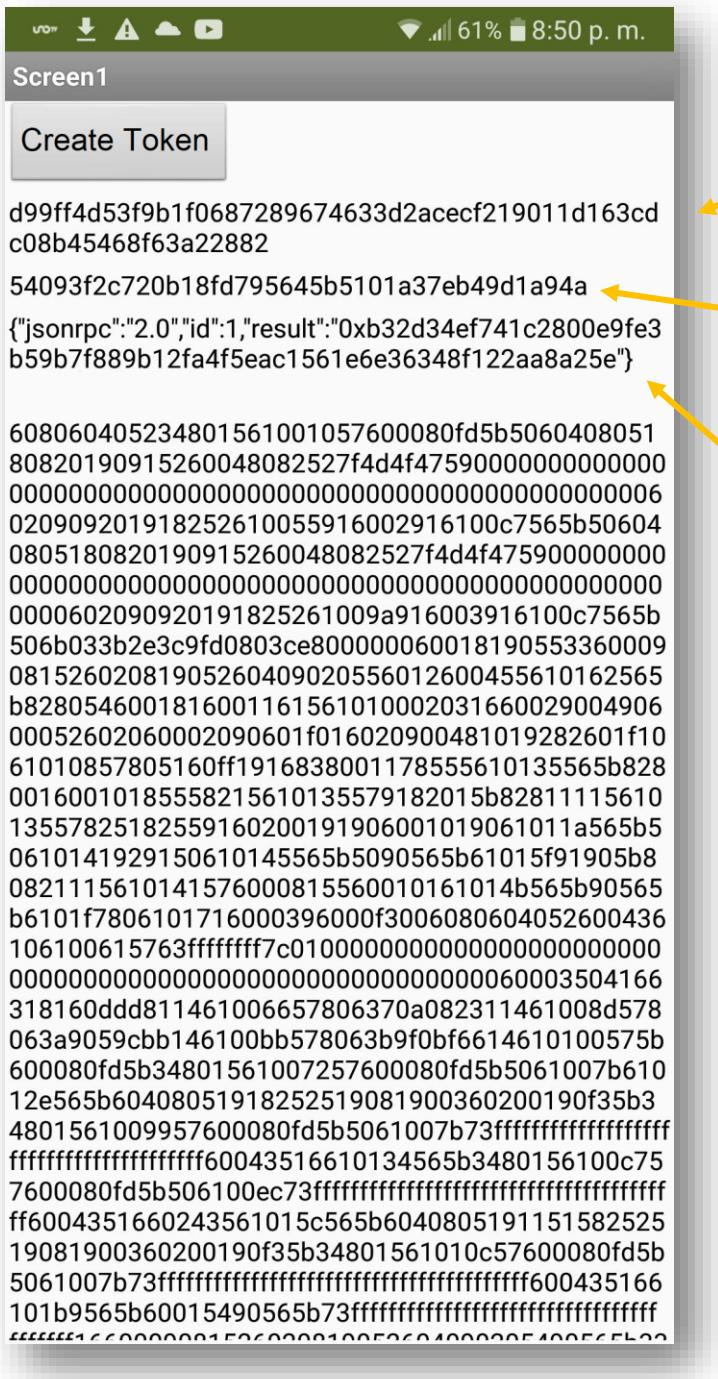
Parámetros de salida: Evento (**OutputSmartContractTokenERC20v1**).

Outputs: **tx\_hashToken<String>**, **addressToken<String>**, **binCodeToken<String>**, **trasactionOperation<String>**.



Descripción: Smart contract “Token ERC20” – activo publicado en la red de Ethereum. La versión 1 (v1) ya tiene pre-establecido el parámetro de gas Limit igual a 500,000 WEI.

Ejemplo de salida de la función anterior `SmartContractCreateTokenERC20v1`.



Transacción (Tx\_hash) de la creación en red Ethereum. Esta puede ser consultada en: [www.etherscan.io](http://www.etherscan.io)

Dirección del nuevo contrato que hace referencia al nuevo Token ERC20 creado.

Transacción (Tx\_hash) de la operación validada en el sistema de MayanCoin.org

Esta puede ser consultada en:  
[www.etherscan.io](http://www.etherscan.io)

Código binario (BIN) del contrato del nuevo Token que fue procesado en la EVM (Ethereum Virtual Machine).

## 9. Pasos para crear un CryptoToken o Criptomoneda Token.

Paso 1.

Verificar que la ruta temporal en el dispositivo móvil en donde se creara el Smart contract es válida y se puede crear un archivo exitosamente. Esto se realiza usando el Bloque (**CreateTestingFile**). Verificar que se haya creado el archivo de prueba que se dio en la variable de entrada “pathTestFile”, genéricamente la ruta esta dada por: [/mnt/sdcard/name\\_file.txt](#)

Paso 2 (opcional).

En este paso verificaremos si la cuenta (dirección) de donde se cobrara la operación tiene el suficiente saldo para realizar la transacción de la creación y publicación de un Smart contract. Esto lo podemos verificar usando el Bloque (**eth\_VerifiBalanceForTransaccionSmartContract**). Este uso de este bloque es opcional ya que los bloques que generan los token **SmartContractCreateTokenERC20v1** o **SmartContractCreateTokenERC20v2** ya contienen internamente esta verificación.

Paso 3.

Seleccionar cual Bloque para crear Token ERC20 será usado, tenemos dos opciones:

- a.- Bloque **SmartContractCreateTokenERC20v1** ya cuenta con el valor implícito de Gas Limit asignado con un valor de 500,000 WEI.
- b.- Bloque **SmartContractCreateTokenERC20v2** tiene la opción de poder configurar el Gas Limit a la necesidad del usuario final o desarrollador. Se debe de tener en cuenta que si se da un Gas Limit muy bajo menor a 350,000 Wei es muy posible que no se ejecute el Smart contrac.

Paso 4.

Utilizar cualquiera de los Bloques **SmartContractCreateTokenERC20v1** o **SmartContractCreateTokenERC20v2** asegurándose al usar la variable de entrada “nameFileSolidityJSON” sea igual a la variable de la entrada “pathTestFile” del Bloque (**CreateTestingFile**) que ya se verifica en el paso 1.

Paso 5.

Antes de ejecutar la creación de un Token ERC20 con cualquiera de los Bloques **SmartContractCreateTokenERC20v1** o **SmartContractCreateTokenERC20v2** se recomienda guardar los valores del Evento (resultados) según sea el caso para guardar los resultados (tx\_hashToken, addressToken, binCodeToken, transactionOperation). Ver ejemplo de salida de la función anterior **OutPutSmartContractCreateTokenERC20v1**.

Paso 6.

Ejecutar la creación del Token ERC20 y posteriormente publicarlo para su venta. Ver el apartado 10.

## 10.Como poner a la venta un nuevo activo o tu criptoToken (Token ERC20).

Ya que hemos creado un Token ERC20 – Criptomoneda Token (Ver Bloque **SmartContractCreateTokenERC20v1** o con el Bloque **SmartContractCreateTokenERC20v2**) tenemos que subirlo a algún Exchange para que cualquier persona del mundo pueda comprarlo. Un Exchange es un sitio en internet en donde se publican los Token nuevos.

Los Exchange se clasifican de dos tipos, Centralizados y Descentralizados. La principal diferencia es que uno está regido y auditado por algún tipo de organismo internacional (Centralizado) y los descentralizados no tienen nadie con los audite. Si bien esto podría dar más confianza la realidad es que últimamente los descentralizados han tomado más fuerza y son usados la mayoría sin problemas mayores.

Una de las mejores prácticas al manejar activos de cualquier tipo es no tener todos en una sola cuenta sino distribuirla en varios esto por seguridad tanto de las llaves privada y pública.

En nuestro caso usaremos un Exchange Descentralizado pero ya con un historial no malo, usaremos [www.forkdelta.app](https://www.forkdelta.app)

En este momento debemos ya tener instalado la aplicación para el navegador (Mozilla o Chrome) **METAMASK** [www.metamask.io](https://www.metamask.io) esto es debido a que el Exchange al visitar su pagina [www.forkdelta.app](https://www.forkdelta.app) se necesitará conectar a la cuenta que tengamos de Ethereum.

Un punto importante es que nuestra cuenta de Ethereum que ya tenemos en METAMASK deberá tener saldo más o menos un mínimo de 10 USD esto es debido a que cuando publiquemos nuestro nuevo Token ERC20 que creamos con el Bloque **SmartContractCreateTokenERC20v1** o con el Bloque **SmartContractCreateTokenERC20v2** necesitaremos pagar la transacción de publicarlo en el Exchange.

Para poder usar el Exchange [www.forkdelta.app](https://www.forkdelta.app) deberemos tener los siguientes datos de Token que deseamos publicar para vender en el Exchange.

Dirección del nuevo Token ERC20 que creamos anteriormente.

**0x54093F2C720b18Fd795645b5101A37EB49d1A94a**

Número de decimales que usa nuestro token.

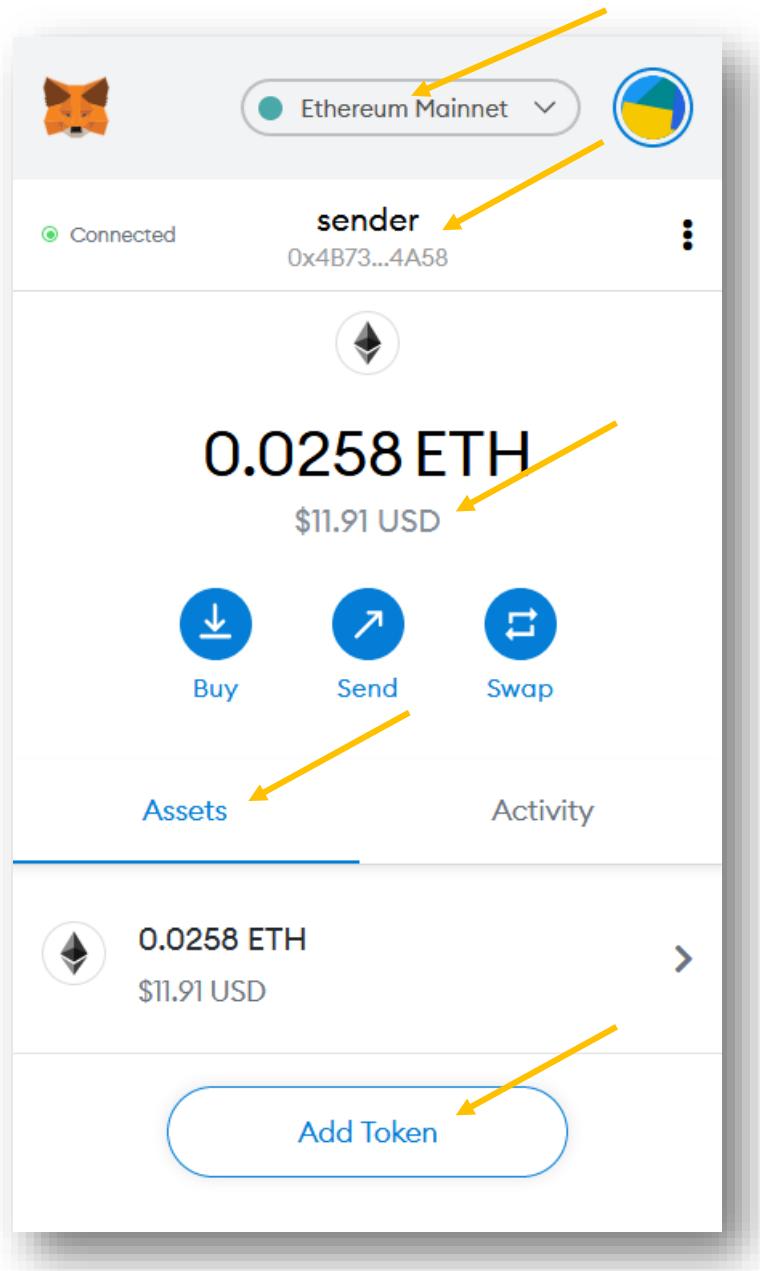
**18**

El nombre que identifica al token.

**MOGY**

Empecemos revisando que el Token que creamos tenga los parámetros antes mencionados y que fueron con los que inicialmente fue creado.

Vamos a **METAMASK** y primero nos aseguramos que estamos en la cuenta con la que creamos el Token. Despues vamos a la parte inferior y damos “click” en el botón de “Add Token”.



Ahora agregaremos nuestro token nuevo para ver su saldo actual. Después de dar “Click” en la parte superior escogemos el apartado de “Custom token” y en los siguientes campos escribimos la información que nos solicitan, posteriormente damos “Click” en el botón de “Next”. Después nos aparece nuestro nuevo token con el saldo que tiene. En caso de no tener saldo revisar si estamos en la cuenta con que creamos el Token ya que si no es así lógicamente como estamos en una cuenta con la que no lo creamos nos aparecerá un saldo de cero porque aún no compramos ningún token de ese tipo.

The image consists of two side-by-side screenshots of a mobile application interface for managing Ethereum tokens.

**Left Screenshot: Add Tokens**

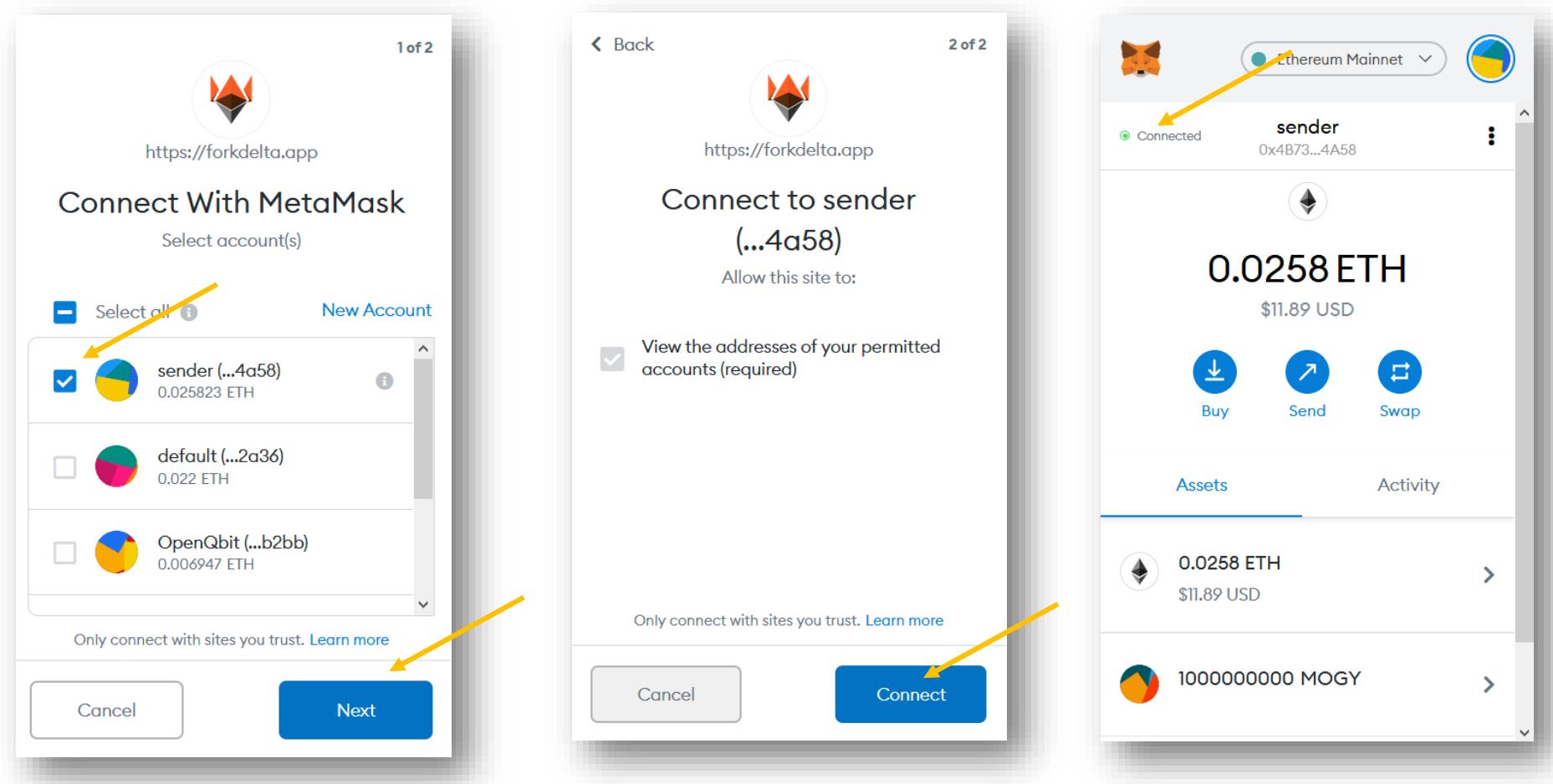
- Header: Ethereum Mainnet
- Section: Add Tokens
- Fields:
  - Search
  - Custom Token (highlighted by a yellow arrow)
  - Token Contract Address: 0x54093F2C720b18Fd795645b5101A37...
  - Token Symbol: MOGY (highlighted by a yellow arrow)
  - Decimals of Precision: 18 (highlighted by a yellow arrow)
- Buttons: Cancel, Next

**Right Screenshot: sender / MOGY**

- Header: Ethereum Mainnet
- Section: sender / MOGY
- Icon: Fox head icon
- Balance: 1000000000 MOGY (highlighted by a yellow arrow)
- Buttons: Send, Swap
- Text: You have no transactions

En momento de subir nuestro nuevo Token a la venta en el Exchange [www.forkdelta.app](https://forkdelta.app)

Al estar en el sitio del Exchange nos pedirá que nos conectemos al sitio <https://forkdelta.app> le damos “Click” en el bajo inferior “Next”, después “Connect” y al final ya podremos revisar que estamos conectados al sitio de forkdelta.app



Es hora de dar de alta el token nuevo en el Exchange <https://forkdelta.app>

Damos “Click” en el menú superior “DAI” y vamos al final del scroll y escogemos la opción “Other”

The screenshot shows the ForkDelta exchange interface. At the top, there is a navigation bar with links for FAQs, Help, Tokens, Smart Contract, English, and MetaMask. Below the navigation bar, the URL https://forkdelta.app/#!/trade/DAI-ETH is displayed. On the left side, there is a sidebar with sections for Balance, Deposit, Token, Amount, ETH, and Volume. A search bar is also present in the sidebar. The main area features an Order Book, a Price Chart, and a Trades section. In the Order Book, several bids and asks are listed for DAI/ETH. The Price Chart shows price levels from 0.00 to 1.00 with depth markers. The Trades section lists recent trades for DAI/ETH, DAI, and ETH. At the bottom of the page, there is a "My Transactions" section with tabs for Important, Trades, Orders, and Funds. A "Tweets" section by @ForkDelta is also visible. Two yellow arrows point to the "DAI" dropdown menu at the top left and the "Other" option in the dropdown menu itself.

Posteriormente damos de alta el nuevo token con los datos que ya sabemos.

The screenshot shows the ForkDelta application interface. On the left, there's a sidebar with 'Balance' and 'Volume' sections. The 'Volume' section lists various tokens like ASTRO, REQ, SXDT, and SNOV with their current prices and volumes. In the center, there's a 'New Order' form for buying or selling DAI, DAI/ETH, or ETH. A modal window titled 'Other token' is open, prompting for the token's address (0x54093F2C720b18Fd795645b5101A37EB49d1A94a), name (MOGY), and decimals (18). The right side of the screen shows a list of recent trades and a 'My Transactions' section. At the bottom, there's a 'Tweets' section from the @ForkDelta Twitter account.

En este momento aparecerá el nuevo token en la parte superior izquierda del Exchange, solo lo hemos subido al Exchange falta publicarlo para que todo el mundo pueda comprarlo y verlo. Ahora necesitamos depositar un poco de Ether (0.015) es suficiente de nuestra cuenta al Exchange.

The image consists of two side-by-side screenshots of the ForkDelta exchange interface, both titled "ForkDelta MOGY".

**Left Screenshot (Initial State):**

- Balance:** Shows a table with three rows: Token (MOGY), Wallet (1000000000.000), and ForkDelta (0.000). Below the table are two input fields: "Amount" (for MOGY) and "Deposit".
- Volume:** Shows a search bar and a list of tokens with their current values. The tokens listed are ASTRO, REQ, SXDT, and SNOV.
- Bottom:** A search bar with placeholder text "Escribe aquí para buscar" and a Windows logo.

**Right Screenshot (After Deposit):**

- Balance:** The same table as the left screenshot, but now the ForkDelta column shows 0.026. Below the table, the "Amount" field for MOGY contains "0.015" and the "Deposit" button is highlighted with a yellow arrow.
- Volume:** The same search bar and list of tokens as the left screenshot.
- Bottom:** A search bar with placeholder text "Escribe aquí para buscar" and a Windows logo.

Ya que hemos realizado el depósito podremos depositar la cantidad de Token que queramos en el Exchange [www.forkdelta.app](https://www.forkdelta.app)

Para depositar una cantidad definida de Tokens necesitamos crear una orden de compra, esto lo hacemos en la parte inferior donde dice “New Order” y damos “Click” en la opción de “Sell”. Posteriormente introducimos la cantidad de tokens que deseamos poner a la vista de cualquier comprador en el mundo, el precio en Ether que deseamos vender cada uno (precio unidad) y el parámetro de “Expires” es la cantidad de tiempo que deseamos tener a la venta estos tokens esto se calcula con la siguiente formula:

Expires Time = 14 segundos X cantidad ingresada.

Ejemplo: 14 segundos X 10000 = 140,000 segundos = 1.62 días

The screenshot shows the ForkDelta app interface for trading MOGY tokens. On the left, there's a 'Balance' section with tabs for Deposit, Withdraw, and Transfer. Under the Deposit tab, it shows 1000000000.000 MOGY and 1.057 ETH. Below this are 'Amount' and 'Deposit' buttons. On the right, there's an 'Order Book' section which is currently empty, displaying the message "There are no orders here." In the bottom right corner, there's a 'New Order' form. This form has tabs for Buy and Sell, with 'Sell' being active. It contains fields for MOGY (10000), MOGY/ETH (0.05), ETH (500.000), and Expires (10000). A large orange 'Sell' button is at the bottom of the form. Yellow arrows point from the text above to each of these fields: one to the 'Sell' tab, one to the MOGY input field, one to the MOGY/ETH input field, one to the ETH input field, and one to the Expires input field.

Listo, Ya están a la venta tus tokens nuevos, cualquiera puede entrar y comprarlos. A veces no reconoce el nuevo token por lo que se tendrán que vender desde la aplicación Metamask.

Ejemplo de consulta en el sitio [www.etherscan.io](https://www.etherscan.io) del nuevo Token creado.

The screenshot shows the Etherscan interface for a transaction hash: 0xd99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882. The transaction was successful (Status: Success) and is included in Block 11240201 with 224 confirmations. It occurred 47 minutes ago (Nov-12-2020 02:49:56 AM +UTC) and was confirmed within 30 seconds. The transaction originated from address 0x4b7355fd05be6dac458b004f54e12d6527a54a58 and sent Ether to a newly created contract at address [Contract 0x54093f2c720b18fd795645b5101a37eb49d1a94a Created]. The value sent was 0 Ether (\$0.00), and the transaction fee was 0.011014691 Ether (\$5.06). The gas price was 0.000000041 Ether (41 Gwei) and the gas limit was 500,000. The gas used by the transaction was 268,651 (53.73%). A yellow arrow points from the 'Tx\_hash' label to the transaction hash in the details. Another yellow arrow points from the 'Created' label to the contract address. A third yellow arrow points from the 'Gas Price' label to the note below it.

Etherscan

All Filters | Search by Address / Txn Hash / Block / Token / Ens

Eth: \$459.17 (-0.93%) | 24 Gwei

Home | Blockchain | Tokens | Resources | More | Sign In

Transaction Details

Sponsored: Crypto.com - The Crypto Super App - Buy Crypto at True Cost with 0% fee on credit card purchase. [Get App Now.](#)

Overview State Comments

② Transaction Hash: 0xd99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882

② Status: Success

② Block: 11240201 224 Block Confirmations

② Timestamp: 47 mins ago (Nov-12-2020 02:49:56 AM +UTC) | Confirmed within 30 secs

② From: 0x4b7355fd05be6dac458b004f54e12d6527a54a58

② To: [Contract 0x54093f2c720b18fd795645b5101a37eb49d1a94a Created] ✓

② Value: 0 Ether (\$0.00)

② Transaction Fee: 0.011014691 Ether (\$5.06)

② Gas Price: 0.000000041 Ether (41 Gwei)

② Gas Limit: 500,000

② Gas Used by Transaction: 268,651 (53.73%)

Transacción (Tx\_hash) de la creación en red Ethereum.

Dirección de contrato de nuevo Token creado.

Solo costo de red Ethereum.  
No incluye costo de operación + 15 USD.

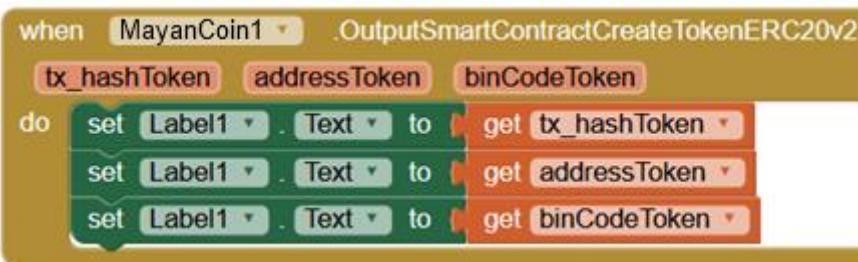
Bloque para compilar, crear y publicar Token ERC20 – (**SmartContractCreateTokenERC20v2**)



Parámetros de entrada: **tokenName** <String>, **tokenSymbol** <String>, **totalAmount** <String>, **numberDecimal** <String>, **gas\_limit** <Integer>, **privateKeyHex** <Integer>, **nameFileSolidityJSON** <String>.

Parámetros de salida: Evento (**OutputSmartContractTokenERC20v2**).

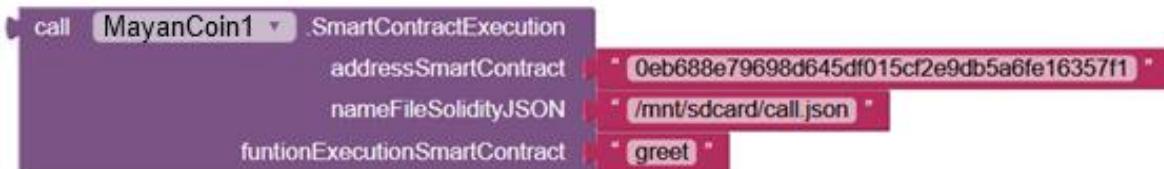
Outputs: **tx\_hashToken**<String>, **addressToken**<String>, **binCodeToken**<String>.



Descripción: Smart contract “Token ERC20” – activo publicado en la red de Ethereum. La versión 2 (v2) puede ser optimizado el valor del Gas Limit ya que dependiendo del smart contract que tan rápido se desee realizar la publicación en la red de ethereum. Se recomienda que mínimo debe ser un valor de 350,000 WEI para que se realice la publicación sin fallas o demoras.

La diferencia entre las funciones de creación de TokenERC20v1 y creación de TokenERC20v2 es que en el caso de la versión 1 el parámetro de GasLimit ya esta pre configurado y en la versión 2 se puede configurar a las necesidades del usuario o desarrollador.

Bloque para llamar o ejecutar Token ERC20 – (SmartContractExecution)



Parámetros de entrada: `addressSmartContract<String>`, `nameFileSolidityJSON<String>`, `funtionExecutionSmartContract<String>`.

Parámetros de salida: Ejecución de función especificada en el Smart contract referenciado.

Outputs: **Ejecución de smart contract**.

**NOTA:** Para ser ejecutado se debe crear un archive con format JSON el cual contiene los parámetros de la llave primaria de la dirección que quiere ejecutar el Smart contract, se necesita introducir el gas limite (WEI).

Ejemplo de archivo JSON para ejecutar las funciones del Smart contract compilado en ejemplo de la función del compilador antes mencionado. El nombre del archivo puede ser arbitrario.

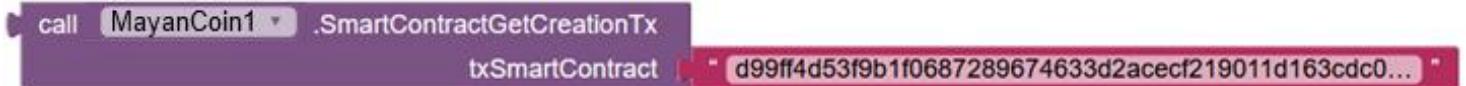
Archivo: call.json

```
{  
  "private": "3ca40...",  
  "gas_limit": 20000  
}
```

Ejemplo de salida de ejecución de la función “greet” del Smart contrato:

```
{  
  "gas_limit": 20000,  
  "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",  
  "results": [  
    "Hello MayanCoin Test"  
  ]  
}
```

Bloque para mostrar propiedades de Token ERC20 – (**SmartContractGetCreationTx**)



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

Parámetros de salida: Muestra propiedades de Smart contract referenciado con (**Tx\_hash**).

Descripción: Muestra las características principales y código ABI del Smart contract referenciado.

Ejemplo de propiedades de Token ERC20, tokenName “MOGY” de ejemplo anteriormente creado usando la función (**martContractCreateTokenERC20v1**)

```
{
  "block_hash": "cadb8914c43ed28fb370c9e1b6f5d8818ba31a1e944d1f7049441b982902dea8",
  "block_height": 11240201,
  "block_index": 170,
  "hash": "d99ff4d53f9b1f0687289674633d2acecf219011d163cdc08b45468f63a22882",
  "addresses": [
    "4b7355fd05be6dac458b004f54e12d6527a54a58"
  ],
  "total": 0,
  "fees": 11014691000000000,
  "size": 958,
  "gas_limit": 500000,
  "gas_used": 268651,
  "gas_price": 41000000000,
  "contract_creation": true,
  "relayed_by": "200.77.24.87",
  "confirmed": "2020-11-12T02:49:56Z",
  "received": "2020-11-12T02:50:13.185Z",
  "ver": 0,
  "double_spend": false,
```



```
        "54093f2c720b18fd795645b5101a37eb49d1a94a"  
    ]  
}  
]
```

Bloque para mostrar código de compilación de Token ERC20 – (SmartContractGetcodeABI)

call MayanCoin1 .SmartContractGetcodeABI

**addressSmartContract**

0eb688e79698d645df015cf2e9db5a6fe16357f1

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

Parámetros de salida: Mostra código de Smart contract referenciado.

Descripción: Muestra código ABI del Smart contract referenciado.

Ejemplo de código ABI de un Smart contract genérico:

```

    "abi": [
      {"constant": false, "inputs": [], "name": "kill", "outputs": [], "type": "function"}, {"constant": true, "inputs": [], "name": "greet", "outputs": [{"name": "", "type": "string"}], "type": "function"}, {"inputs": [{"name": "_greeting", "type": "string"}], "type": "constructor"}],
    "creation_tx_hash": "61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
    "created": "2016-07-20T01:54:50Z",
    "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1"
  }

```

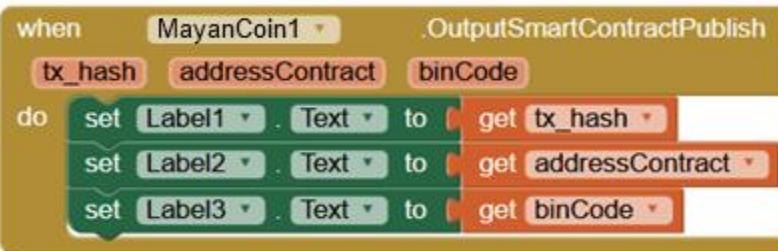
\*Antes usar el siguiente Bloque (**SmartContractPublish**) debe usar el Bloque (**SmartContractCompilerSolidity**) para revisar si el Smart contract está bien escrito.

Bloque para publicar en red Ethereum el Token ERC20 – (**SmartContractPublish**).



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

Parámetros de salida: Muestra propiedades de Smart contract referenciado. En el evento (**OutputSmartContractPublish**).



Descripción: Publica en red ethereum el Smart contract referenciado en el archivo JSON.

Ejemplo de archivo: **PubishSmartContract.json**

```

{
  "solidity": "contract mortal { \n    /* Define variable owner of the type\n     address */\n    address owner;\n    /* this function is executed at\n     initialization and sets the owner of the contract */\n    function\n     mortal() { owner = msg.sender; }\n    /* Function to recover the funds on\n     the contract */\n    function kill() { if (msg.sender == owner)\n        suicide(owner); }\n}\n\ncontract greeter is mortal {\n    /* define\n     variable greeting of the type string */\n    string greeting;\n    /*\n     this runs when the contract is executed */\n    function\n     greeter(string _greeting) public {\n        greeting = _greeting;\n    }\n    /* main\n     function */\n    function\n     greet() constant returns (string) {\n        return greeting;\n    }\n}",
  "params": ["Hello Test"],
  "publish": ["greeter"],
  "private": "3ca40...",
  "gas_limit": 500000
}

```

Como se muestra en el código anterior es el mismo código JSON usado en el ejemplo de la función de compilación al mismo código se han agregado los parámetros al final del archivo JSON:

**Params:** Parámetros implícitos del Smart contrac.

**Publish:** Nombre de cómo se publicará el Smart contrac.

**Private:** Llave primaria de la cuenta que ejecutará el Smart contract deberá tener saldo.

**Gas\_limit:** Es el saldo en WEI que se desea gastar para la publicación del Smart contract.

Ejemplo de salida cuando se ejecutó (publicando Smart contract) el Smart contract se introduce dentro de la red ethereum. Muestra la transacción realizada “**creation\_tx\_hash**” y la dirección asignada del Smart contract creado “**address**”.

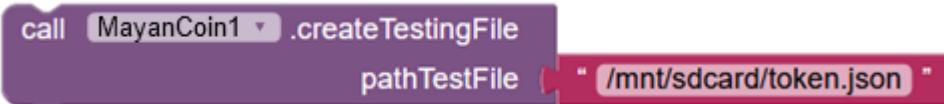
```
[
  {
    "name": "greeter",
    "solidity": "contract mortal {\n    /* Define variable owner of the\n    type address*/\n    address owner;\n    /* this function is executed at\n    initialization and sets the owner of the contract */\n    function\n    mortal() { owner = msg.sender; }\n    /* Function to recover the funds on\n    the contract */\n    function kill() { if (msg.sender == owner)\n        suicide(owner); }\n}\ncontract greeter is mortal {\n    /* define\n    variable greeting of the type string */\n    string greeting;\n    /*\n    this runs when the contract is executed */\n    function greeter(string\n    _greeting) public {\n        greeting = _greeting;\n    }\n    /* main\n    function */\n    function greet() constant returns (string) {\n        return greeting;\n    }\n}",
    "bin": "606060405260405161023e38038061023e8339810160405280510160008054600160a060020a031916331790558060016000509080519060200190828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f10609f57805160ff19168380011785555b50608e9291505b808211560cc57600081558301607d565b50505061016e806100d06000396000f35b828001600101855582156076579182015b8281115607657825182600050559160200191906001019060b0565b509056606060405260e060020a600035046341c0e1b58114610026578063cfae321714610068575b005b6100246000543373ffffffffffffffffff0908116911614156101375760005473fff0908116911614156101375760005473fff16ff5b6100c9600060609081526001805460a06020601f6002600019610100868816150201909416939093049283018190040281016040526080828152929190828280156101645780601f1061013957610100808354040283529160200191610164565b60405180806020018281038252838181518152602001915080519060200190808383829060006004602084601f0104600f02600301f150905090810190601f1680156101295780820380516001836020036101000a031916815260200191505b509250505060405180910390f35b565b820191906000526020600020905b81548152906001019060200180831161014757829003601f168201915b5050505090509056",
    "abi": [
      {
        "constant": false,
        "inputs": [],
        "name": "kill",
        "outputs": [],
        "type": "function"
      }
    ]
  }
]
```

```
{
  "constant": true,
  "inputs": [],
  "name": "greet",
  "outputs": [
    {
      "name": "",
      "type": "string"
    }
  ],
  "type": "function"
},
{
  "inputs": [
    {
      "name": "_greeting",
      "type": "string"
    }
  ],
  "type": "constructor"
},
  "gas_limit": 500000,
  "creation_tx_hash":
"61474003e56d67aba6bf148c5ec361e3a3c1ceea37fe3ace7d87759b399292f9",
  "address": "0eb688e79698d645df015cf2e9db5a6fe16357f1",
  "params": [
    "Hello Test"
  ]
},
{
  "name": "mortal",
  "solidity": "contract mortal {\n    /* Define variable owner of the\n    type address*/\n    address owner;\n    /* this function is executed at\n    initialization and sets the owner of the contract */\n    function\nmortal() { owner = msg.sender; }\n    /* Function to recover the funds on\n    the contract */\n    function kill() { if (msg.sender == owner)\n        suicide(owner); }\n}\n\ncontract greeter is mortal {\n    /* define\n    variable greeting of the type string */\n    string greeting;\n    /*\n    this runs when the contract is executed */\n    function greeter(string _greeting) public {\n        greeting = _greeting;\n    }\n    /* main\n    function */\n    function greet() constant returns (string) {\n        return greeting;\n    }\n}",
  "bin": "606060405260008054600160a060020a03191633179055605c8060226000396000f36060
60405260e060020a600035046341c0e1b58114601a575b005b60186000543373ffffffffffff
ffffffffffffffffffff90811691161415605a5760005473fffffffffffff16ff5b56",
  "abi": [
    {
      "constant": false,
      "inputs": [],
      "name": "kill",
      "outputs": [],
      "type": "function"
    },
    {

```

```
        "inputs": [],
        "type": "constructor"
    },
],
"gas_limit": 500000,
"params": [ "Hello Test"]
}
]
```

Bloque de prueba para crear archivo – ([createTestingFile](#))



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

Parámetros de salida: Se crea un archivo de prueba en la ruta referenciada.

Descripción: Este sirve para comprobar la ruta valida de creación de archivo temporal para asegurarnos que es correcto cuando usemos el Bloque ([SmartContractCreateTokenERC20v1](#)) o el Bloque ([SmartContractCreateTokenERC20v2](#)).

Bloque para obtener las tarifas de Gas Price – ([eth\\_RatesGasStationInfo](#)).



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

Parámetros de salida: Muestra propiedades de Smart contract referenciado. En el evento ([OutputEth\\_GasStationInfo](#)) los valores entregados están dados en GWEI.

El Gas Price es el valor que será pagado a los sistemas que ejecutan las transacciones en la red de Ethereum estos sistemas se les conoce comúnmente como “mineros” y los valores del Gas Price está en función de que rápido (tiempo y prioridad) se realice la transacción en la red de Ethereum.

Los valores entregados están en función de los siguientes tiempos de ejecución. Estos tiempos son aproximados y pueden variar dependiendo de cómo está en las demandas (transacciones) en la red de Ethereum.

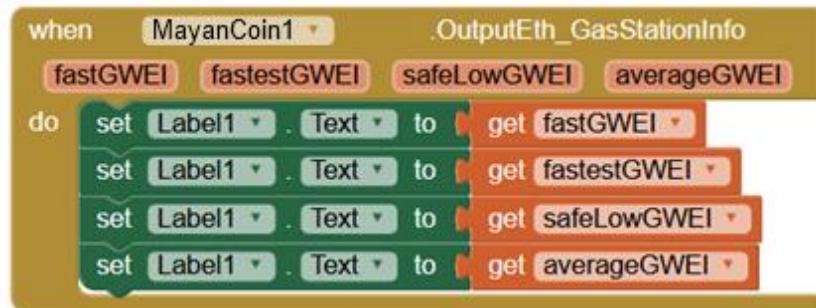
**Fast** < 2 minutos.

**Fastest** < 30 segundos.

**SafeLow** < 30 minutos.

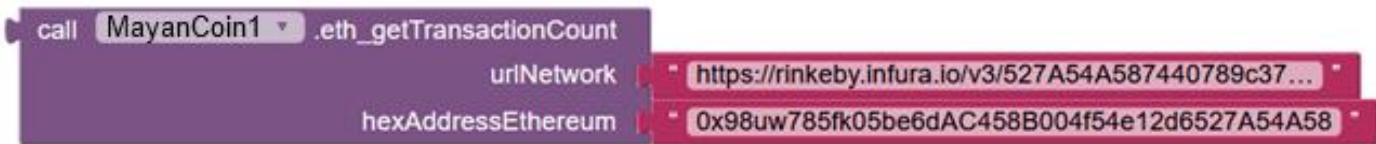
Average < 15 minutos.

Las transacciones realizadas con las Exchange Ethereum Extensión (EEE) siempre usan el Gas Price = Average.



Descripción: Obtiene el Gas Price actualizado en el momento de la consulta para crear una nueva transacción.

Bloque para obtener el número “nonce” – (eth\_getTransactionCount).



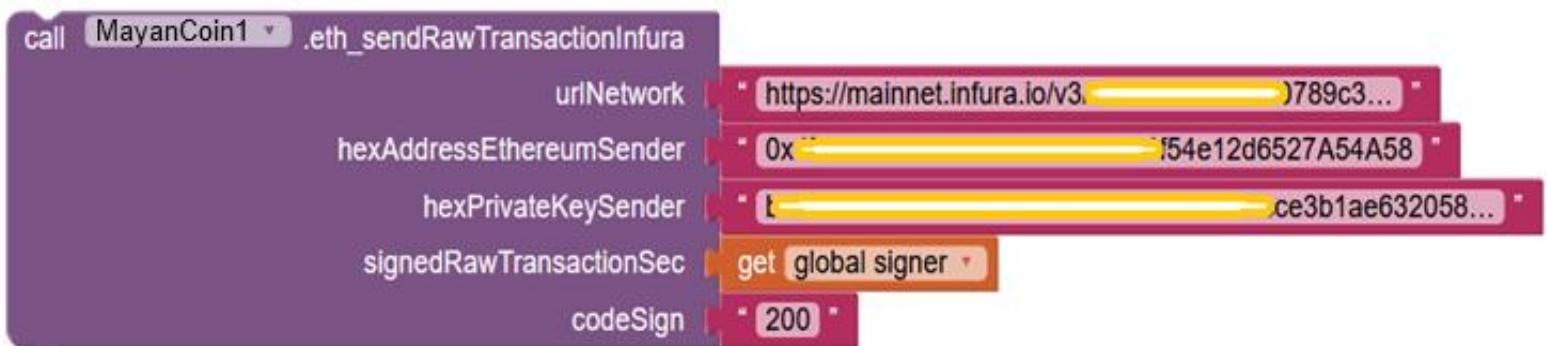
Parámetros de entrada: urlNetwork<String>, hexAddressEthereum<String>.

Parámetros de salida: Muestra en formato hexadecimal el numero consecutivo “nonce” de la dirección referenciada.

El número “nonce” es un numero incremental que lleva la contabilidad del número de transacciones que se han realizado de una dirección específica.

Descripción: Obtiene el número “nonce” de la dirección referenciada.

Bloque para enviar una transacción firmada – (eth\_SendRawTransactionInfura).

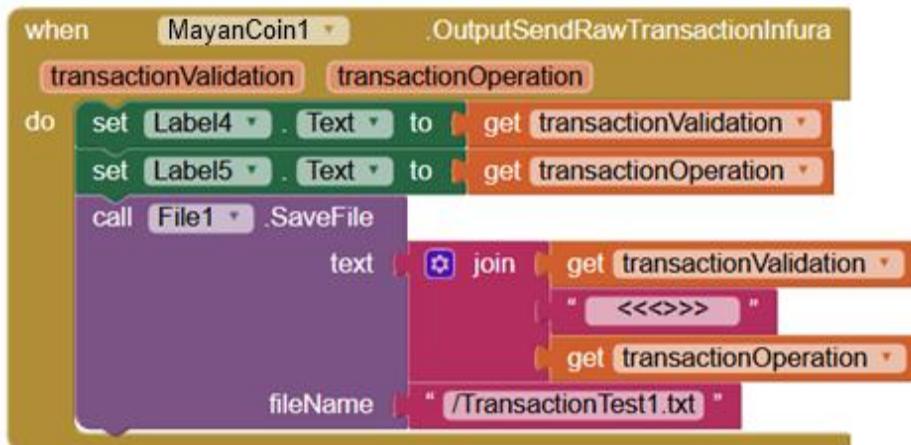


Dependencia(s) obligatoria(s): Bloque **(eth\_getTransactionCount)**, Bloque **(SignerGenericPushRawTransactionOffline)**. Revisar ejemplo del bloque **(SignerGenericPushRawTransactionOffline)**.

Parámetros de entrada: **urlNetwork <String>**, **hexAddressEthereumSender <String>**, **hexPrivateKeySender <String>**, **SignedRawTrasactionSec <String>**, **codeSign <String>**.

Parámetros de salida: Evento (**OutputSendRawTransactionInfura**).

Outputs: **transactionValidation<String>**, **transactionOperation<String>**.



Descripción: Entrega dos valores hexadecimales como resultado de las transacciones. El valor de **trasactionValidation** es la transacción efectuada en la red de Ethereum que contiene el costo implícito de Ethereum. El valor de **transactionOperation** es la transacción realizada en la red de MayanCoin con el costo de \$0.5 centavos USD por cada transacción basándose en el valor del ether que se haya tenido en el momento de la transacción, este costo es para el pago de servicios de la red de MayanCoin.org y que se invierte en el mantenimiento, soporte y creación de extensiones para el sector de criptomonedas y activos a nivel mundial.

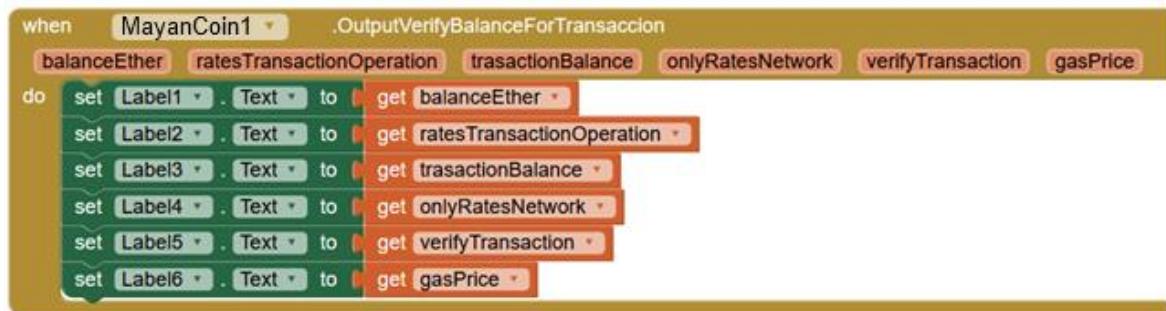
Bloque para calcular costo de una transacción standard – **(eth\_VerifiBalanceForTransaccion)**



Parámetros de entrada: **addressEthereumSender<String>**, **valueEthertoSend<String>**.

Parámetros de salida: Evento (**OutputVerifyBalanceForTransaccion**).

Outputs: **balanceEther<String>**, **ratesTransactionOperation<String>**,  
**trasactionBalance<String>**, **OnlyRatesNetwork<String>**, **verifyTransaction<String>**,  
**gasPrice<String>**.



Descripción: Entrega detalles de cual ser el costo de una transacción standard referencia a la dirección de entrada. El parámetro de salida “verifyTransaction” nos indica si la transacción se puede realizar “True” o en su caso si la dirección referenciada no cuenta con suficiente saldo nos entregara un “False”.

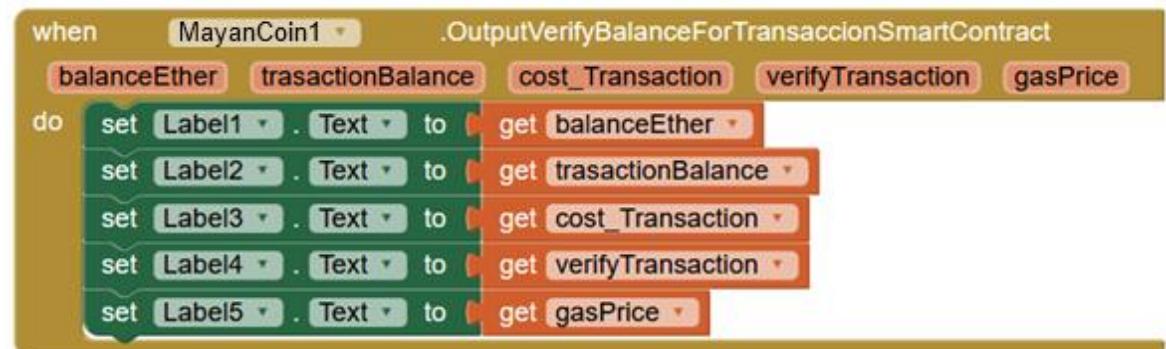
Bloque para calcular costo de una transacción standard –  
(eth\_VerifiBalanceForTransaccionSmartContract)



Parámetros de entrada: **addressEthereum<String>**, **gasLimit<String>**.

Parámetros de salida: Evento (**OutputVerifyBalanceForTransaccionSmartContract**).

Outputs: **balanceEther<String>**, **trasactionBalance<String>**, **cost\_Transaction<String>**,  
**verifyTransaction<String>**, **gasPrice<String>**.



Descripción: Entrega detalles de cual ser el costo aproximado de una transacción standard para publicar un **Smart contract**, referencia a la dirección de entrada. El parámetro de salida “verifyTransaction” nos indica si la transacción se puede realizar “True” o en su caso si la dirección referenciada no cuenta con suficiente saldo nos entregara un “False”.

**balanceEther**: Balance de la dirección referenciada esta se entrega en ethers.

**trasactionBalance**: Balance después de realizarse la transacción.

**cost\_Transaction**: Es el costo de la transacción para poder publicar el smart contrat.

**verifyTransaction**: (balanceEther menos el costo de cost\_Transaction).

**gasPrice**: Valor actual del GasPrice que usan los “Mineros”, este puede variar cada minuto.

Bloque para obtener precio Ether en moneda de un país especificado – (**eth\_getExchangeRates**).



Parámetros de entrada: **countryRates<String>**. Revisar parámetro de salida “country” en donde contiene todos los tipos de moneda de países para escoger el deseado.

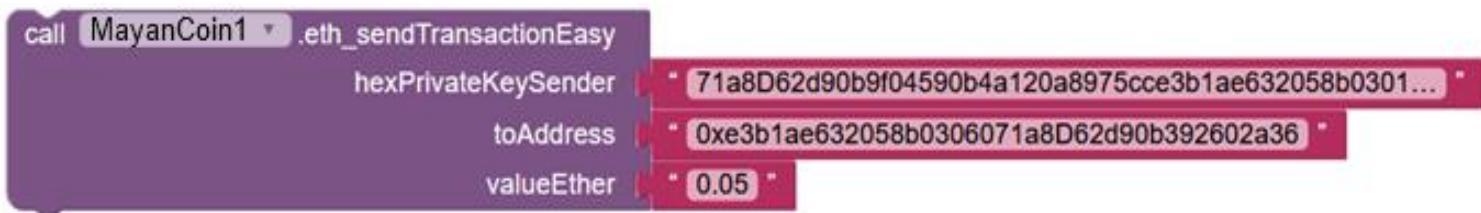
Parámetros de salida: Evento (**OutputEth\_getExchangeRates**).

Outputs: **rates<String>**, **countries<String>** salida en format JSON todas las tarifas de los países de mundo.



Descripción: Entrega el precio actual de un Ether al tipo de cambio de la moneda del país referenciado.

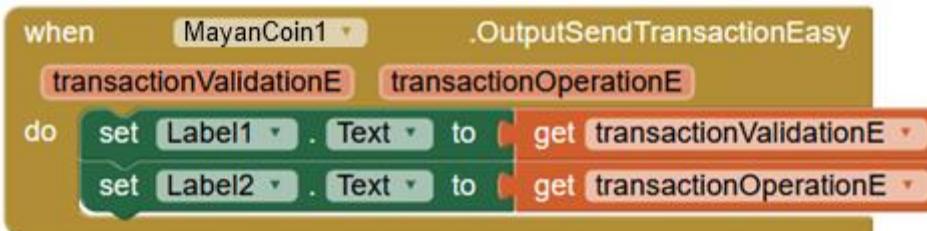
Bloque para realizar una transacción standard con los mínimos parámetros de entrada – (eth\_sendTransactionEasy).



Parámetros de entrada: hexPrivateKeySender<String>, toAddress<String>, valueEther<String>.

Parámetros de salida: Evento (OutputSendTransactionEasy).

Outputs: transactionValidation<String>, transactionOperation<String>.



Descripción: Función para realizar una transacción standard en la red de Ethereum, esta función es de uso inmediato no se necesita tener una cuenta en INFURA y se necesitan únicamente 3 parámetros de entrada, solo se necesita tener el suficiente saldo para realizar la transacción deseada.

Las transacciones son puestas en la red Ethereum usando directamente la librería oficial de Ethereum Web3j y nuestra red MayanCoin.org

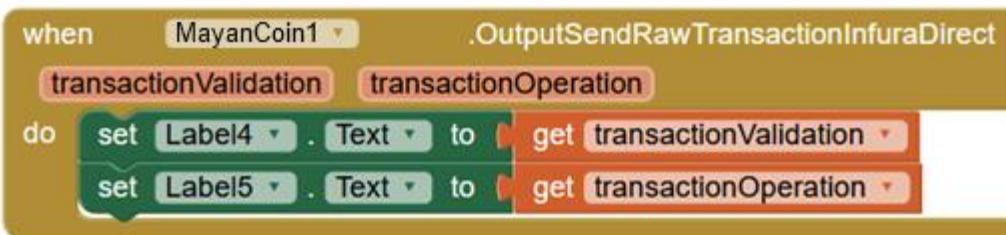
Bloque para realizar una transacción standard con los mínimos parámetros de entrada – (eth\_sendTransactionInfuraDirect).



Parámetros de entrada: `urlNetwork<String>`, `hexPrivateKeySender<String>`, `gasPrice<String>`, `gasLimit<String>`, `toAddress<String>`, `valueWEI<String>`.

Parámetros de salida: Evento (`OutputSendTransactionInfuraDirect`).

Outputs: `transactionValidation<String>`, `transactionOperation<String>`.



Descripción: Función para enviar una transacción standard que ya contiene la firma digital implícita, esta sirve para personas que ya tienen conocimientos previos de los componentes de una transacción y quieren optimizar estos parámetros según sean sus necesidades.

## 11. Calculo de costo de transacción standard y transacción Smart contract.

Para el cálculo de una transacción estándar se necesitan 3 parámetros en la red de Ethereum.

- 1.- Gas Price.
- 2.- Gas Limit.
- 3.- Valor actual de un Ether (Moneda digital de Ethereum).

**GasPrice:** Esta dado normalmente en unidades de GWEI (GigaWEI) esto corresponde a que, si 1 ether tiene 1,000,000,000,000,000 wei entonces 1 GWEI es igual a 1,000,000,000. Esta unidad sirve como pago de los sistemas que ejecutan todas las transacciones de la red Ethereum y son llamados “mineros”, se encuentra distribuidos en todo el mundo. El GasPrice no es un valor fijo y es variable y puede cambiar de minuto a minuto o segundo. Los que definen el valor de GasPrice son los “mineros” y depende de que tan saturada este la red de Ethereum.

**GasLimit:** Este valor esta dado normalmente en unidades de WEI y en una transacción standard un valor promedio por default es de 21,000 WEI aunque puede ser mayor o menor dependiendo qué tipo de transacción se quiera realizar, en una transacción standard usamos el valor por de 40,000 WEI para asegurarnos que no tengamos rechazo por tener bajo Gas Limit.

**Valor del Ether:** Este valor también es variable y se debe a diferentes parámetros de un mercado financiero global, este valor lo podemos obtener de entidades que tienen siempre actualizado el valor de Ether a nivel mundial llamados Exchange centralizados y descentralizados.

NOTA IMPORTANTE: En la Exchange Ethereum Extensión (EEE) usamos un Gas Limit mas alto (40,000) esto es debido a que en caso de no alcanzar la mínima cuota establecida por los “mineros” NO SE EFECTURA LA TRANSACCION, sin embargo la red de Ethereum, si cobrara el gasto que haya realizado en el cálculo de la transacción sin realizarla, por esta razón algunos usuarios no se explican por qué no se realiza su transacción, sin embargo se cobrara un monto o todo el GasLimit que se oferto cuando se lanzó la solicitud de la transacción, por esta razón debemos siempre tener claro cuáles serán los valores de GasLimit y Gas Price.

Se pueden consultar el GasPrice con la función **eth\_GetRatesGasStation** y el GasLimit para transacciones standard deberá ser superior a 21,000 WEI y transacciones para publicar y/o ejecutar Smart contract deberá ser mínimo de 500,000 WEI.

Costo de transacción standard.

Se define con la siguiente formula:

$$\text{Costo} = (\text{GasLimit} \times (\text{GasPrice} / (1,000,000,000,000,000,000))) \times \text{Valor Ether}.$$

Ejemplo:

Supongamos los siguientes valores:

**GasLimit** = 40,000, **GasPrice** = 45 GWEI, **Valor Ether** = 406 USD.

$$\text{Costo} = (40,000 \times (45,000,000,000 / (1,000,000,000,000,000,000))) \times 406 \text{ USD}$$

$$\text{Costo de transacción standard} = 0.0018 \text{ ether} \times 406 \text{ USD} = \$0.73 \text{ USD}$$

En el caso de la transacción de un Smart contract se necesita saber qué tipo de Smart contract se publicará ya que el costo es directamente proporcional a la carga de trabajo que realizaran los “mineros” para poder procesar el Smart contract, dicho en otras palabras, más simples es la cantidad de procesamiento en los sistemas de cómputo que manejan los “mineros”.

Para el caso de los Smart contrato un valor por default seria empezar el GasLimit con un valor de 500,000 WEI.

Un punto importante que se tiene que tomar en cuenta cuando uno propone un GasLimit no necesariamente los “mineros” tomaran toda la cantidad propuesta, es decir cuando uno envía una transacción los “mineros” calculan el esfuerzo de cómputo y sacar lo que se ocupara del GasLimit pudiendo ser menos o igual al GasLimit default que es de 21,000 WEI ofertado en algunos casos.

Todas las transacciones podrán ser consultadas en el sitio [www.etherscan.io](https://www.etherscan.io) en donde podemos consultar el detalle de cada transacción.

**Ejemplo de trasaccion standard**, en donde se envio la transaccion con un Gas Limit de 40,000 WEIs, sin embargo los “mineros” al calcular cuanto seria el costo de procesamiento solo tomaron el 52.5% es decir el valor default que son 21,000 WEIs.

The screenshot shows the Etherscan Transaction Details page for a standard Ethereum transaction. The transaction hash is 0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a. The transaction was successful and included in block 11234630 with 2 block confirmations. It occurred 52 seconds ago on Nov-11-2020 at 06:17:24 AM UTC. The transaction originated from address 0x4b7355fd05be6dac458b004f54e12d6527a54a58 and went to address 0x5d2acdb34c279aa6d1e94a77f7b18ab938fb2bb. The value transferred was 0.001084598698481562 Ether (\$0.50). The transaction fee was 0.000672 Ether (\$0.31), which is 21,000 WEI at a gas price of 0.000000032 Ether (32 Gwei). The proposed gas limit was 40,000 WEI, but the actual gas used was 21,000 (52.5%). A yellow arrow points from the transaction hash to the success status. Another yellow arrow points from the value field to the cost calculation box. Arrows also point from the gas limit and used gas fields to their respective boxes.

Field	Value	Notes
Transaction Hash	0x7dae251f21c616fb04a94112633c97e04d11c91f4d06dd9b85ed11112f02703a	
Status	Success	TransactionOperation or Transaction Validation
Block	11234630	2 Block Confirmations
Timestamp	52 secs ago (Nov-11-2020 06:17:24 AM +UTC)	Confirmed within 38 secs
From	0x4b7355fd05be6dac458b004f54e12d6527a54a58	
To	0x5d2acdb34c279aa6d1e94a77f7b18ab938fb2bb	
Value	0.001084598698481562 Ether (\$0.50)	Costo de trasaccion en Ether: $21,000 \times 0.000000032 = 0.000672$ Ether (\$0.31 USD)
Transaction Fee	0.000672 Ether (\$0.31)	
Gas Price	0.000000032 Ether (32 Gwei)	Gas Limit propuesto: 40,000 WEI
Gas Limit	40,000	
Gas Used by Transaction	21,000 (52.5%)	Gas Limit real que se usó en la transacción: 21,000 WEI.
Nonce	36	
Position	79	

Regresando a la transacción del Smart contract y tomando el GasLimit de 500,000 WEI obtenemos el siguiente costo en caso de usarlo todo.

Costo de transacción Smart contract.

Se define con la siguiente formula:

Costo = (GasLimit x (GasPrice / (1,000,000,000,000,000,000)) x Valor Ether.

Ejemplo:

Supongamos los siguientes valores:

**GasLimit** = 500,000, **GasPrice** = 45 GWEI, **Valor Ether** = 406 USD.

Costo = (500,000 x (45,000,000,000 / (1,000,000,000,000,000,000)) x 406 USD

**Costo de transacción publicar Smart contract** = 0.0225 ether x 406 USD = \$9.135 USD

Toda transacción puede ser consultada en el sitio [www.etherscan.io](http://www.etherscan.io)

**NOTA:** Para obtener el costo total de la transacción se deben sumar:

**Costo Total de Transacción** = Costo red Ethereum + Tarifa de MayanCoin.org

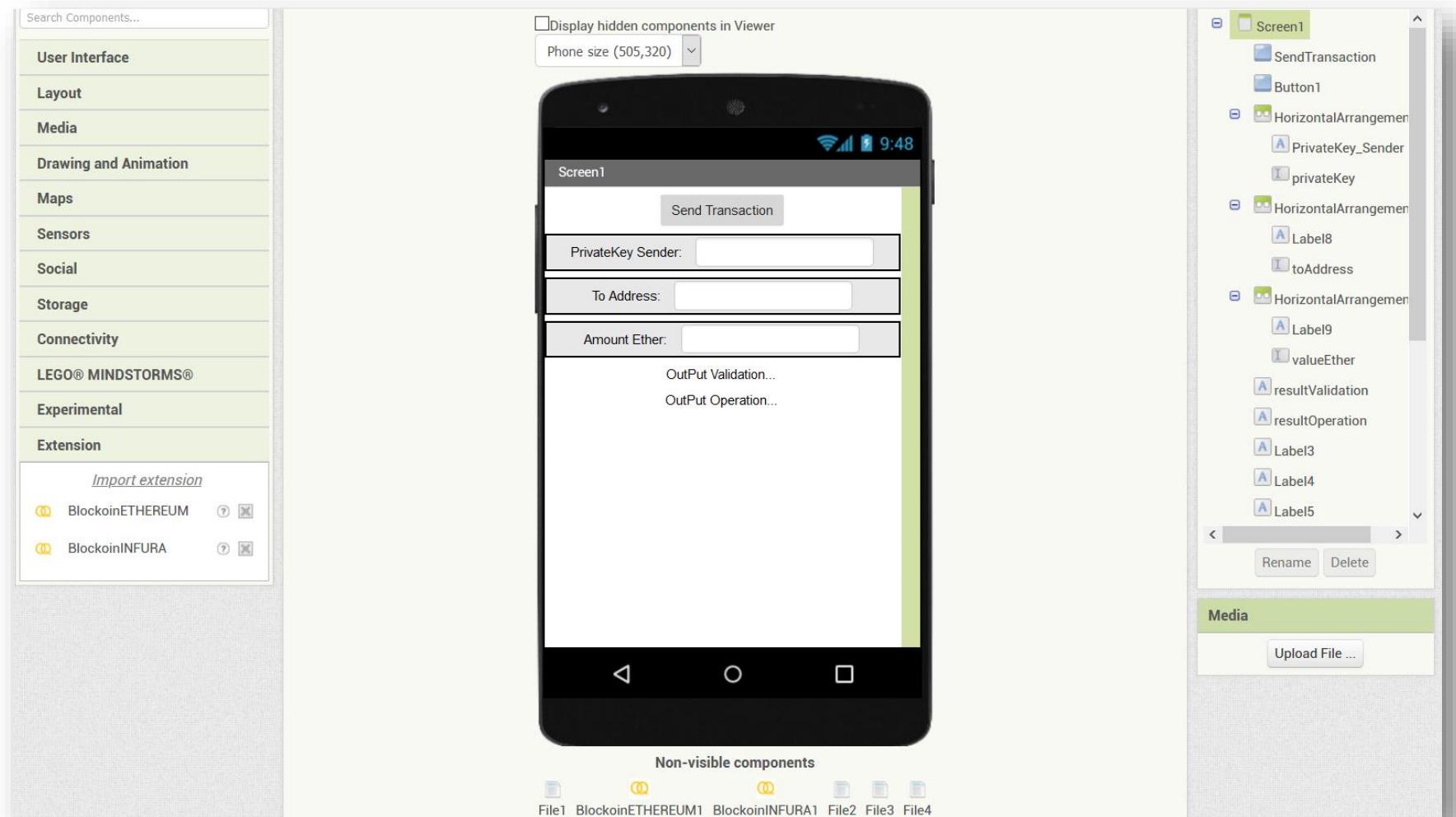
## 12. Tarifas de MayanCoin.org

Transacción standard: \$ 0.5 centavos USD + costo red Ethereum.

Transacción publicar y/o ejecutar un Smart contract: \$15 dólares USD + costo red Ethereum.

## 13. Creación tu App (Exchange) para Android en 15 minutos.

Diseño en App Inventor (Screen). – 5 minutos.



Bloques de función (eth\_SendTransactionEasy) y evento (OutPutSendTransactionEasy). – 5 minutos

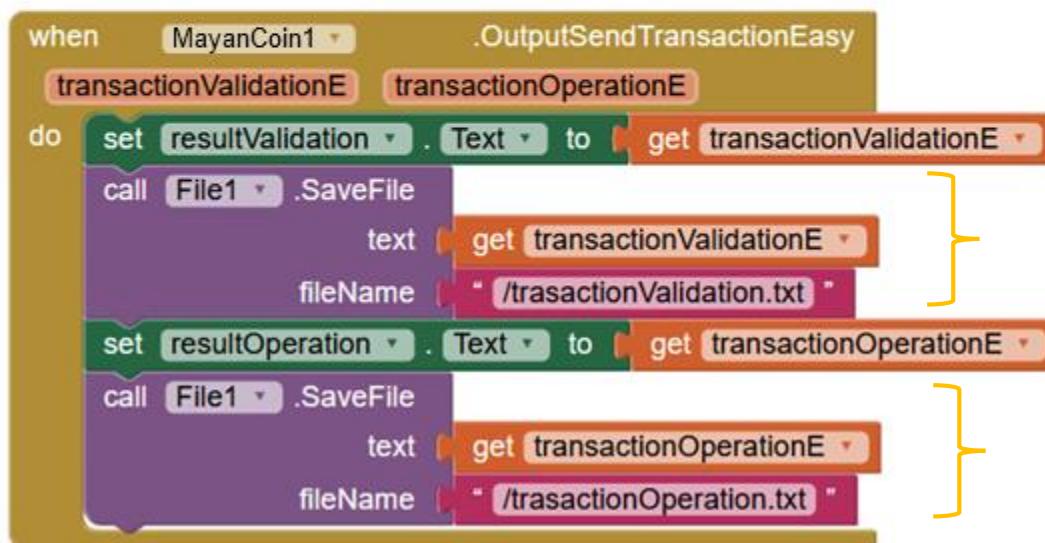


Datos de entrada:

**PrivateKey**: Llave primaria de la dirección del remitente.

**toAddress**: Dirección hexadecimal del receptor.

**valueEther**: Dar la cantidad de Ether que serán enviados.



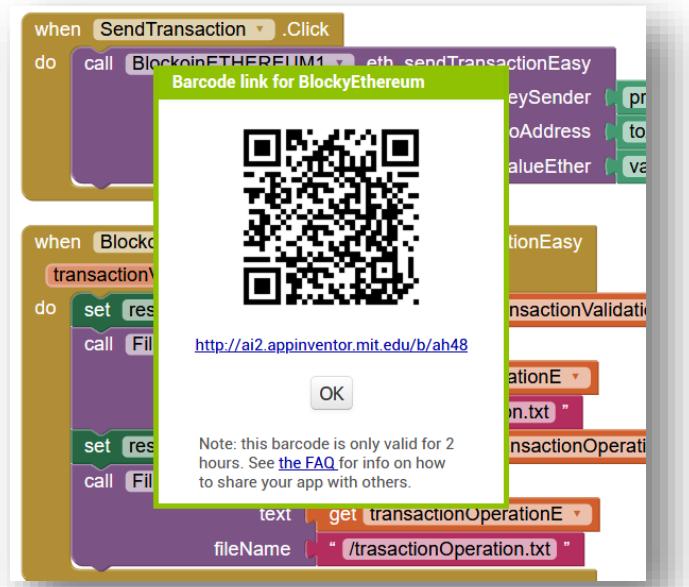
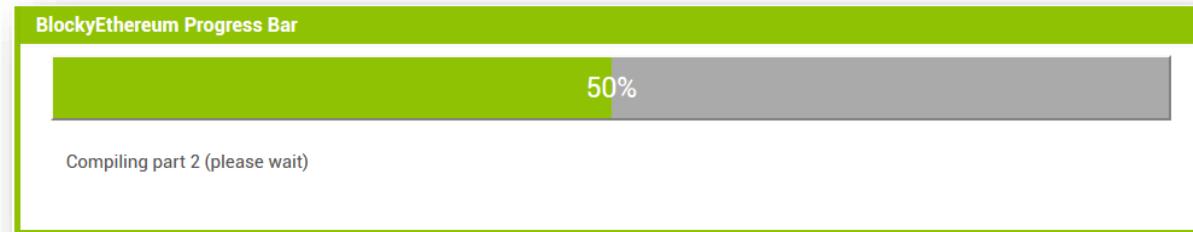
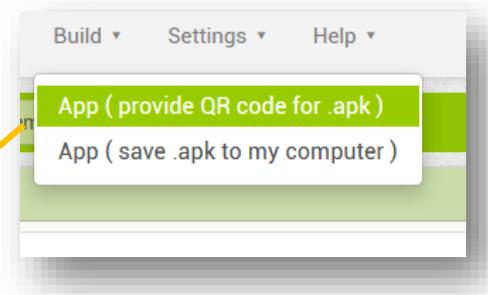
Guardar los resultados en archivos de texto:

Función File1: Archivo **trasactionValidation.txt**

Guardar los resultados en archivos de texto:

Función File2: Archivo **trasactionValidation.txt**

Compilamos, generamos archivo APK para instalarlo en el dispositivo Android. – 5 minutos



NOTA: Al ejecutar la transacción esta tardara de liberar el botón “Send Transaction” en un tiempo aproximado de 6 a 8 segundos. Debido al tiempo de conexión con la red Ethereum.

## 14. Token MayaCoin hace referencia al proyecto mundial MayanCoin.

La Token MayanCoin es un proyecto con tres directrices fundamentales.

La primera es crear la primera red de extensiones basadas en la metodología de programación visual Blockly, en la cual por su fácil e intuitivo uso la puede usar cualquier persona sin conocimientos previos de programación, las extensiones que se pueden consultar en nuestro Roadmap (White Paper) se encaminan a dos sectores fundamentales en la economía mundial, sector de criptomonedas y/o tokens y el sector de monedas (fiat) o de uso común a nivel mundial como puede ser el dólar USD, Euros EU, Libras o cualquier otra moneda de uso corriente.

La segunda directriz en el proyecto de MayanCoin tenemos comunicación directa con nuestros inversores y público mundial a través de nuestra aplicación de Mensajería soportada en la red de Telegram llamada MayaChat compatible con Telegram, disponible para bajar en el sitio:

Creamos el primer juego híbrido usando propiedades de la red de Bitcoin y Ethereum en donde la iniciativa es para apoyar, ayudar y promover los servicios de “Hospitality” a nivel mundial, iniciamos nuestro proyecto en el área de la Riviera Maya usando nuestro Business for Hospitality mayores informes en [www.MayanCoin.com](http://www.MayanCoin.com)

Características generales de la CriptoToken MayanCoin:

Nombre: MayaCoin

Símbolo: MAYA

País de Lanzamiento: Estonia

Sitio oficial: [www.MayanCoin.com](http://www.MayanCoin.com)

Empresa: MayanCoin International.

Fecha de lanzamiento: 30 de Diciembre de 2021

Creador: Team MayanCoin.



## 15. Licenciamiento y uso de software.

Licenciamiento, términos y condiciones de uso ver en el sitio de [www.MayanCoin.com](http://www.MayanCoin.com) o escribir a [info@Mayancoin.com](mailto:info@Mayancoin.com)