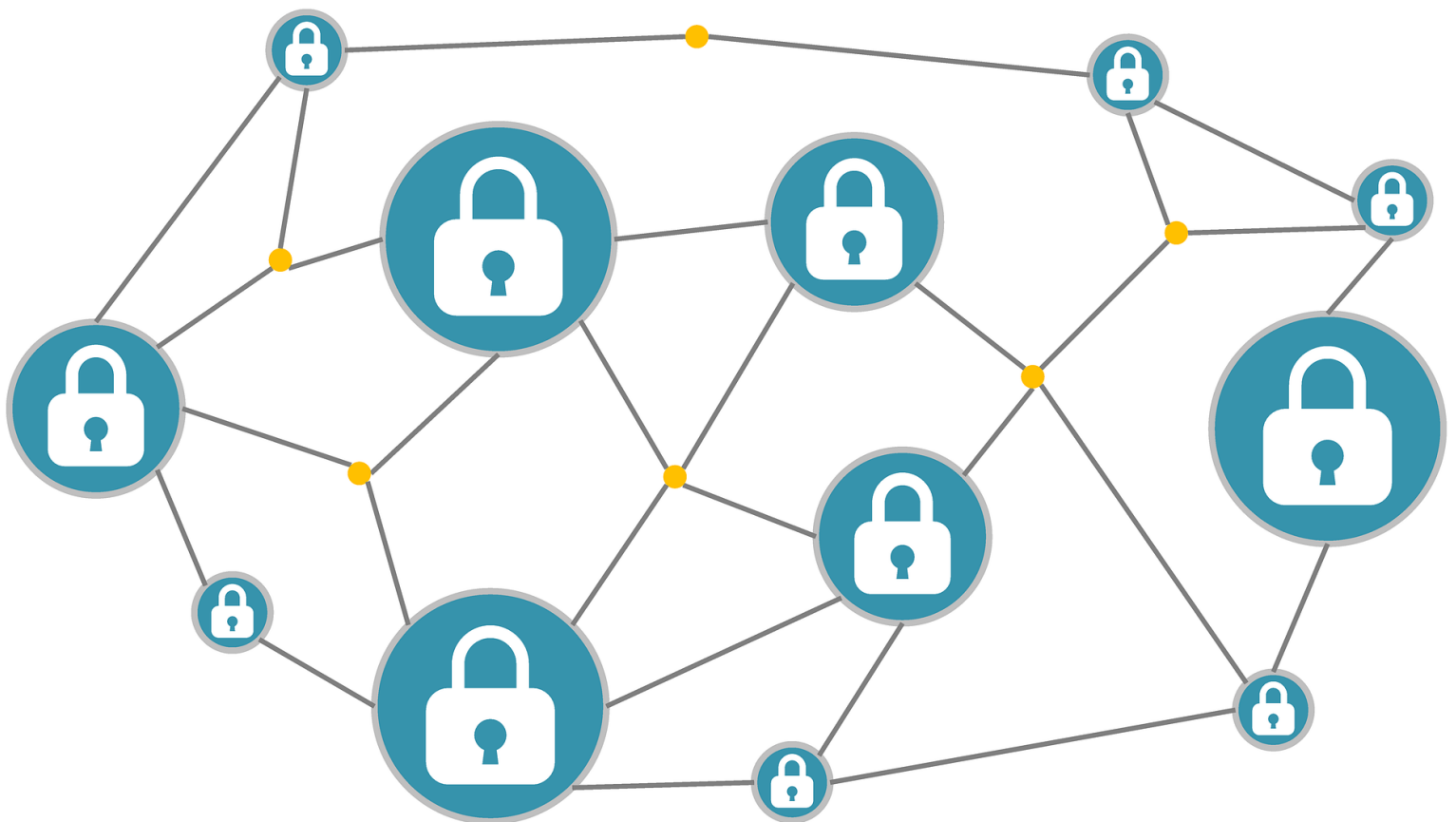


Proyecto Blockchain

Integrantes del grupo:
Miguel Ángel Huamani
Miguel Ángel Vallejo



Introducción

El proyecto consiste en desarrollar una aplicación Blockchain donde múltiples nodos, cada uno representando un proceso en un ordenador con un puerto específico, trabajarán juntos para agrupar transacciones en bloques y mantenerse sincronizados entre sí.

Cada nodo en esta red actuará como un participante activo, contribuyendo a la construcción de bloques mediante la recopilación y agrupación de transacciones. Estos bloques serán enlazados de manera secuencial para formar una cadena de bloques, donde cada bloque contiene un conjunto de transacciones y está vinculado al bloque anterior mediante su hash. Se debe de tener en cuenta que la forma en la que se van a almacenar los datos en este proyecto va a ser en formato JSON.

Blockchain.py

Clase Bloque:

Lo primero que hemos hecho ha sido definir la clase Bloque el cual es el objeto responsable de almacenar las transacciones. Los parámetros de cada bloque son los siguientes: indice, transacciones, timestamp, hash_previo y prueba. Mientras que los métodos que hemos definido dentro del bloque son calcular_hash (que devuelve el hash de cada bloque) y toDict (que devuelve la información de cada bloque en formato JSON, como si fuese un diccionario).

Clase Blockchain:

Hemos definido la clase Blockchain que será el objeto que almacene todos los bloques, y que tendrá como parámetros: la dificultad del blockchain, la lista de bloques (como list) y las transacciones no confirmadas (como list). Los métodos que hemos definido dentro de esta clase incluyen los siguientes:

- `nuevo_bloque`: crea un nuevo bloque con índice como la longitud de la lista de bloques más uno (ya que al principio contendrá el primer bloque), sin hash, con la lista de transacciones confirmadas, el timestamp, el `hash_previo` (hash del bloque anterior de la cadena) y la prueba de trabajo que inicializamos a 0.
- `nueva_transaccion`: crea una nueva transacción a partir de un origen, destino y cantidad. Dicha transacción luego la metemos en la lista de transacciones no confirmadas, para finalmente devolver el índice del bloque que almacenará la transacción.
- `primer_bloque`: con este método creamos el primer bloque de la lista con índice inicializado a 1 y con un hash previo de 1. Después integramos el bloque a la lista de bloques junto con su respectivo hash.
- `prueba_trabajo`: Algoritmo que calculará el hash del bloque hasta que encuentre un hash que empiece por la misma cantidad de 0's que aparecen en la dificultad del blockchain.
- `prueba_valida`: Método que comprueba si el hash comienza con tantos ceros como debería
- `integra_bloque`: Este método se encarga, de como su nombre indica, integrar un bloque junto con su respectivo hash (habiendo pasado la prueba), a la lista de bloques del Blockchain. Después de integrarlo nos encargamos de que se inicialice la lista de transacciones como una lista vacía.

```

159     if __name__ == '__main__':
160         blockchain = Blockchain()
161         blockchain.primer_bloque()
162         idx_transaccion = blockchain.nueva_transaccion('A', 'B', 3000)
163         hash_previo = blockchain.lista_bloques[len(blockchain.lista_bloques)-1].hash # El hash que tiene guardado el último bloque (no sería su hash real)
164         nuevo_bloque = blockchain.nuevo_bloque(hash_previo)
165         hash_prueba = blockchain.prueba_trabajo(nuevo_bloque)
166         print(blockchain.integra_bloque(nuevo_bloque, hash_prueba))
167
168

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS JUPYTER COMENTARIOS Python + ▢

PS C:\Users\mange\CompartmentMagsVirtual> & C:/Users/mange/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/mange/CompartmentMagsVirtual/Blockchain.py

True

PS C:\Users\mange\CompartmentMagsVirtual>

Blockchain_app.py

Obtener dirección (adicional):

Devuelve la dirección del nodo donde se está ejecutando dicha función.

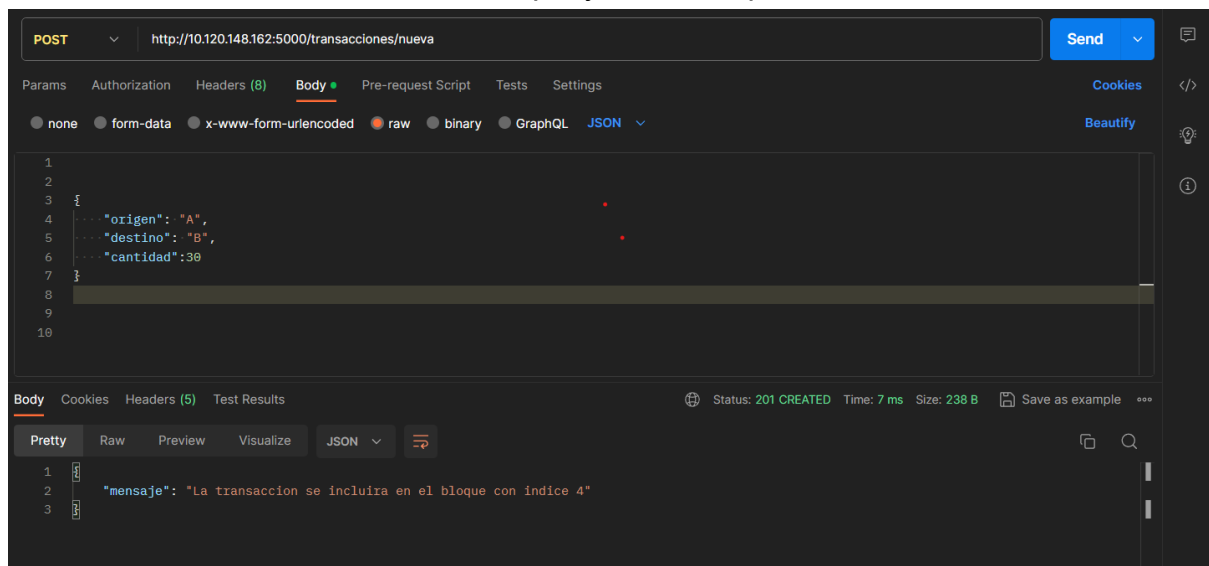
Copia de Seguridad:

Con esta función lo que hemos hecho es hacer un respaldo, es decir, una copia de seguridad del blockchain actual, es decir, de la lista de bloques completa. La copia de seguridad la hacemos abriendo un nuevo fichero cuyo nombre incluye el IP y el puerto del nodo. Es importante destacar que la copia de seguridad la realizamos cada 60 segundos con el uso de un hilo, al igual que hacemos el uso de un semáforo, para que los nodos que están en un mismo sistema ya sea virtual o físico no hagan la copia de seguridad a la vez.

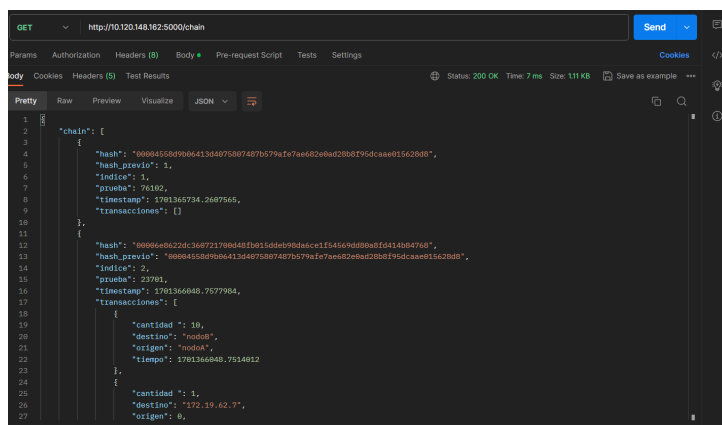
```
Blockchain_app.py
Blockchain.py
pruebas_requests.py
pruebas.ipynb
{} respaldo-nodo172.17.0.1-5.
{} respaldo-nodo172.17.0.1-5.
```

Nueva transacción:

Se envía al proceso un json con la nueva transacción, el proceso la registra la transacción en su lista de transacciones, al añadir un nuevo bloque (al minar), todas las transacciones se meten en el bloque y dicho bloque se añade al blockchain.



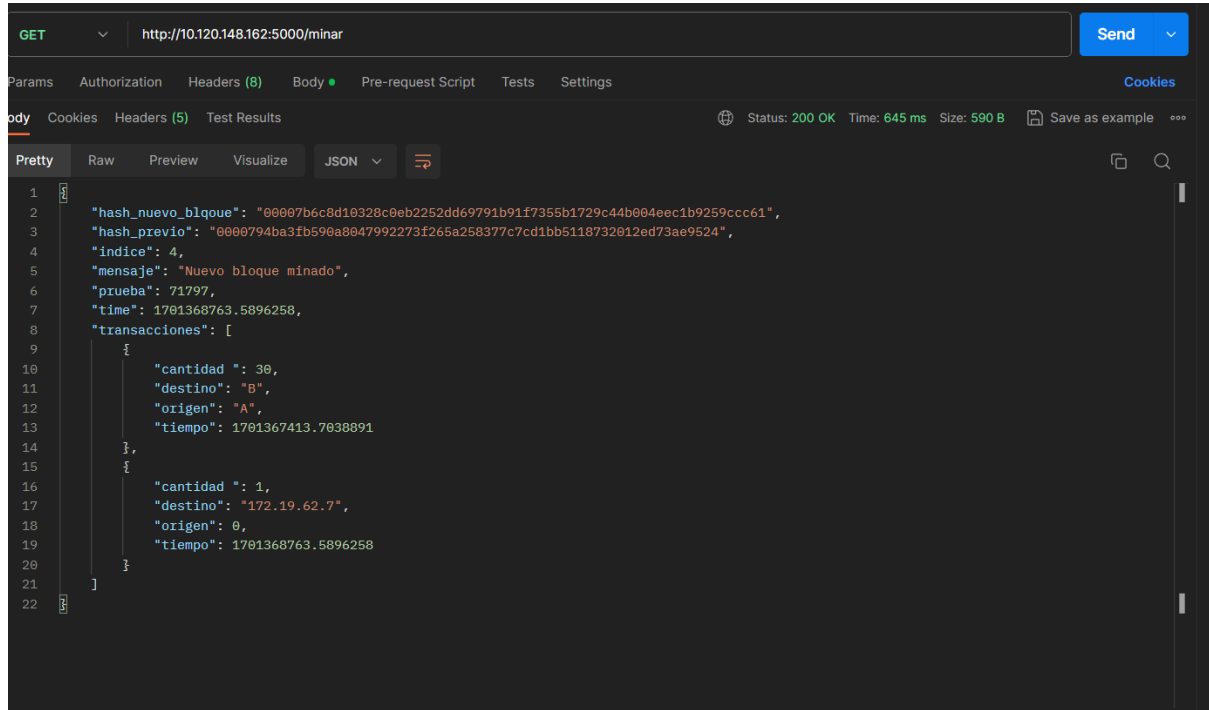
Blockchain completa:



Devuelve el la lista de blockchain actual del nodo donde se ejecuta.

Minar:

Si hay conflictos, esta función actualiza las blockchain necesarias. Si no, comprobamos si hay transacciones, si no las hay no se hace nada ya que para minar se necesitan transacciones. Si hay transacciones, se crea un nuevo bloque donde se incluyen las transacciones y una nueva transacción con destino el nodo que mina y cantidad 1.



```
1  GET http://10.120.148.162:5000/minar
2
3  Params Authorization Headers (8) Body Pre-request Script Tests Settings
4  Status: 200 OK Time: 645 ms Size: 590 B Save as example
5  Pretty Raw Preview Visualize JSON
6
7  {
8    "hash_nuevo_bloque": "00007b6c8d10328c0eb2252dd69791b91f7355b1729c44b004e0c1b9259ccc61",
9    "hash_previo": "0000794ba3fb590a8047992273f265a258377c7cd1bb5118732012ed73ae9524",
10   "indice": 4,
11   "mensaje": "Nuevo bloque minado",
12   "prueba": 71797,
13   "time": 1701368763.5896258,
14   "transacciones": [
15     {
16       "cantidad": 30,
17       "destino": "B",
18       "origen": "A",
19       "tiempo": 1701367413.7038891
20     },
21     {
22       "cantidad": 1,
23       "destino": "172.19.62.7",
24       "origen": 0,
25       "tiempo": 1701368763.5896258
26     }
27   ]
28 }
```

Resolver conflictos:

Cuándo se mina, se comprueba que la blockchain del nodo donde se mina sea la más grande. Se comprueba la blockchain del nodo donde estamos minando con la blockchain de los demás nodos. Si resulta que alguna es más grande que la blockchain del nodo actual, se actualiza la blockchain del nodo actual a dicha blockchain más grande.

Registrar Nodos completo:

Esta función recibe un diccionario con una unica clave 'direcciones_nodos', que contiene una lista de urls de los diferentes nodos que se quieren registrar.

Primero actualiza los nodos de la red en el nodo actual.

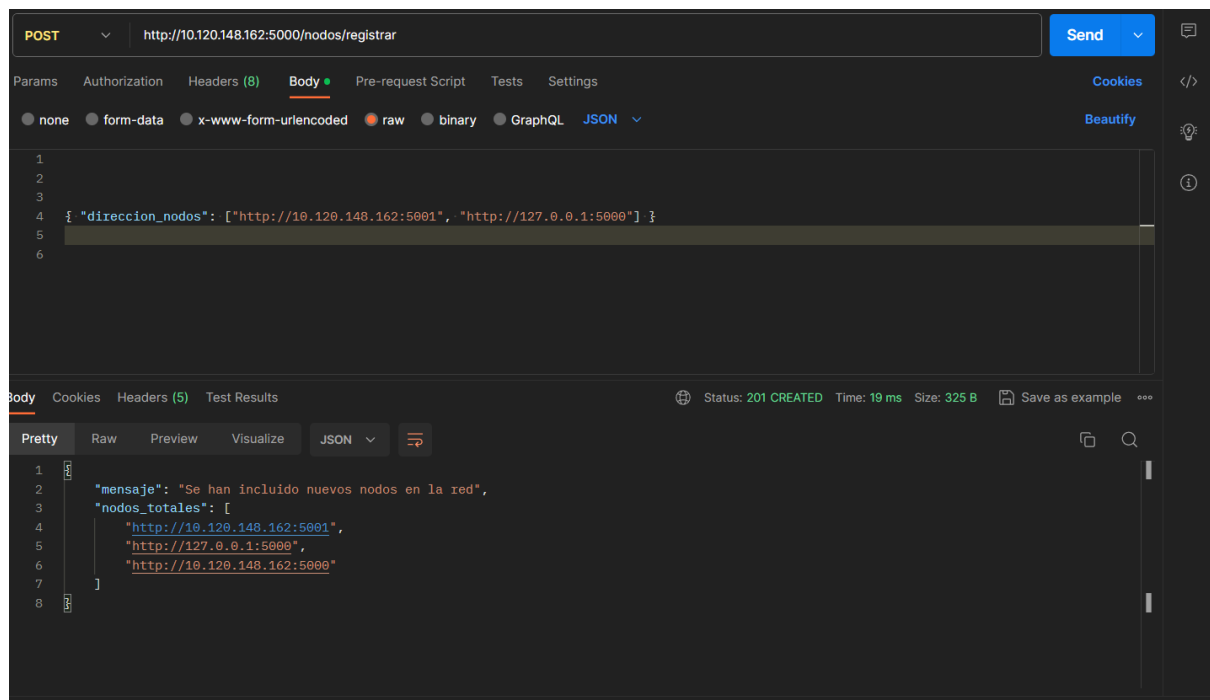
Después, convierte la blockchain del nodo donde se ejecuta a json y la manda al resto de nodos dicha blockchain.

Registrar Nodos simple:

Recibe una lista de bloques (diccionarios) en formato json, para cada diccionario (bloque) crea un objeto bloque y actualiza la blockchain del nodo donde se han enviado. Los mete uno a uno.

En esta función hay que tener cuidado, hay que poner el hash de cada bloque a None para que coincida con el hash que hay guardado en el bloque.

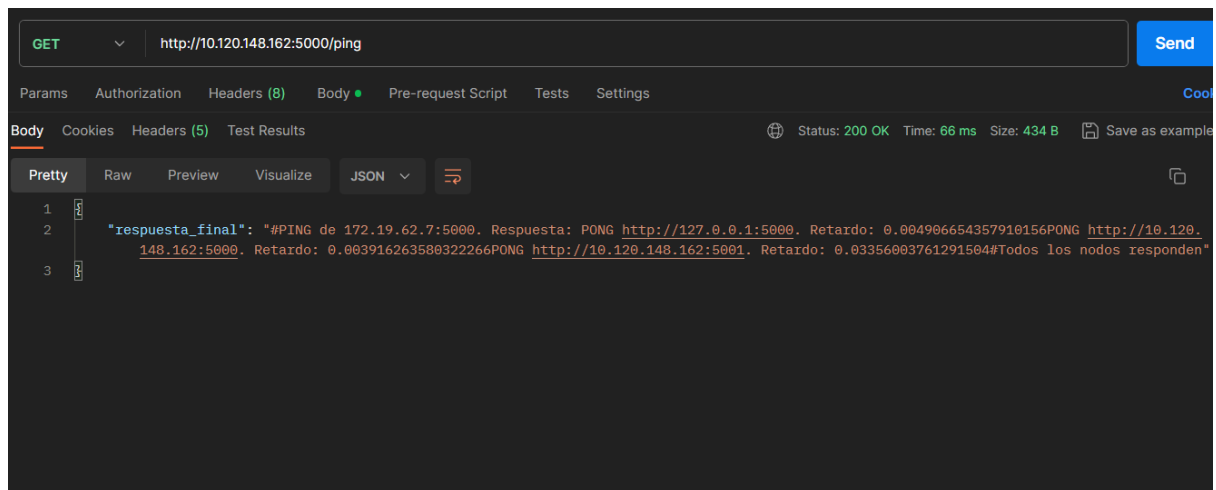
```
for bloque_dic in bloques_dic:
    indice = bloque_dic['indice']
    hash_prueba = bloque_dic['hash'] # Al modificar el atributo 'hash' en el bloque su verdadero hash cambia, por lo que si no lo inicializamo
    transacciones = bloque_dic['transacciones']
    timestamp = bloque_dic['timestamp']
    hash_previo = bloque_dic['hash_previo']
    prueba = bloque_dic['prueba']
    bloque = Blockchain.Bloque(indice, None, transacciones, timestamp, hash_previo, prueba) # el atributo hash del bloque debe ser inicializado a None
    blockchain_leida.integra_bloque(bloque, hash_prueba) # Integramos en bloque (hash_prueba = bloque.calcular_hash()) ya que hash se ha puesto
```



Tras esto, la variable global `nodos_red` en cada proceso contiene los otros nodos de la red (sus direcciones) y todos los nodos tienen la misma blockchain.

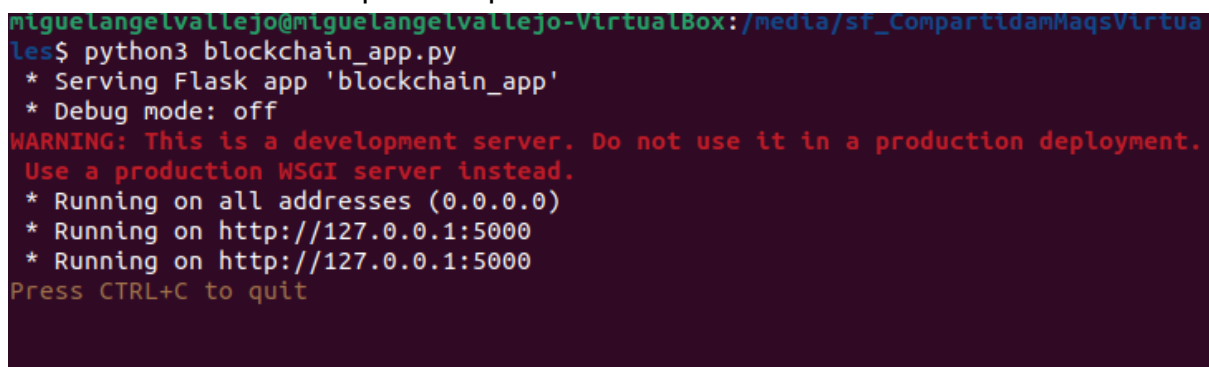
Protocolo ICMP

Lo que hemos hecho en este apartado es prácticamente verificar que el nodo principal establece conexión con el host remoto. Para lograr esto hemos definido dos servicios nuevos (ping y pong), que lo que hacen es enviar datos desde el nodo principal (el que hace el ping) a los demás nodos de la red, y estos responden con los mismos datos (haciendo el pong).



Máquinas Virtuales

En este apartado ejecutamos un nodo en una máquina virtual y los otros 2 en nuestro ordenador. Es importante poner la conexión sólo anfitrión.



Se ha conseguido tener 3 nodos en una misma red y que estén correctamente sincronizados.

Se consiguen registrar todos los nodos correctamente en un nodo principal, además de actualizar la blockchain de cada nodo en caso de conflicto. El hecho de que exista un nodo en la máquina virtual no interfiere con el resto de funciones (a

menos que se intente acceder a un nodo del sistema host a través del guest, que esto creemos que es por la conexión)

```
PS C:\Users\mange\CompartmentMqsvirtuales> python pruebas_requests.py
{"mensaje": "La transaccion se incluirea en el bloque con indice 2"}

{"hash_nuevo_bloque": "00006e8622dc360721700d48fb015dddeb98da6ce1f54569dd80a8fd414b84768", "hash_previo": "00004558d9b06413d4075807487b579afe7ae682e0ad28b8f95dcaae015628d8", "indice": 2, "mensaje": "Nuevo bloque minado", "prueba": 23701, "time": 1701366048.7577984, "transacciones": [{"cantidad": 10, "destino": "nodoB", "origen": "nodoA", "tiempo": 1701366048.7514012}, {"cantidad": 1, "destino": "172.19.62.7", "origen": 0, "tiempo": 1701366048.7577984}]}

{"chain": [{"hash": "00004558d9b06413d4075807487b579afe7ae682e0ad28b8f95dcaae015628d8", "hash_previo": 1, "indice": 1, "prueba": 76102, "timestamp": 1701365734.2607565, "transacciones": []}, {"hash": "00006e8622dc360721700d48fb015dddeb98da6ce1f54569dd80a8fd414b84768", "hash_previo": "00004558d9b06413d4075807487b579afe7ae682e0ad28b8f95dcaae015628d8", "indice": 2, "prueba": 23701, "timestamp": 1701366048.7577984, "transacciones": [{"cantidad": 10, "destino": "nodoB", "origen": "nodoA", "tiempo": 1701366048.7514012}, {"cantidad": 1, "destino": "172.19.62.7", "origen": 0, "tiempo": 1701366048.7577984}]}], "longitud": 2}

{"mensaje": "Se han incluido nuevos nodos en la red", "nodos_totales": ["http://127.0.0.1:5000", "http://10.120.148.162:5000", "http://10.120.148.162:5001"]}

{"respuesta_final": "#PING de 172.19.62.7:5000. Respuesta: PONG http://127.0.0.1:5000. Retardo: 0.004894256591796875PONG http://10.120.148.162:5000. Retardo: 0.004481754302978516PONG http://10.120.148.162:5001. Retardo: 0.0039141178131103516#Todos los nodos responden"}

{"mensaje": "Ha habido un conflicto. Esta cadena se ha actualizado con una versi\u00f3n m\u00e1s larga"}

{"mensaje": "No es posible crear un nuevo bloque. No hay transacciones"}

PS C:\Users\mange\CompartmentMqsvirtuales>
```

Este es el resultado de pruebas_request.py ejecutando como nodo principal el nodo de la máquina virtual. Lo que no se consigue es ejecutarlo en la máquina virtual, esto puede ser por que la conexión solo anfitrión es unidireccional, solo permite acceder del dispositivo host al guest pero no al revés.

Así mismo pruebas_request.py también funciona para cualquier nodo que no esté en la máquina virtual (seleccionando ese nodo como principal en la variable url_nodo).

