Course Outline

1. Cryptocurrency and Block chain
2. Delving into BlockChain
3. Bitcoin and Block chain
4. Bitcoin Mining
5. Ethereum
6. Setting up private Blockchain Environment using Ethereum Platform
7. Hyperledger
8. Setting up Development Program using Hyperledger composer
9. Create or Deploy our private Blockchain on Multi chain
10. Prospect of Blockchain

Create & Deploy Private Blockchain On Multichain

Agenda

At the end of this session you will be able to:

- Define Multichain
- Describe MultiChain Streams
- Create & deploy private blockchain
- Explain Connecting to a Blockchain
- Identify Multichain Interactive Mode
- List Native assets
- Define Transaction Metadata
- Explain Streams
- Explain Mining

Multichain

Multichain

“Let us learn about Multichain.”

Blockchains for Enterprise

Private shared database

Control pf limit + cost

Designate “miners”

No digital money

Aggregate administrator

Shroud the points of interest

Blockchain as apparatus not belief system

Multichain- introduction

- MultiChain is an off-the-rack stage for the creation and arrangement of private blockchains, either inside or between associations
- Derived from Bitcoin center programming, consequently it is perfect with Bitcoin biological community
- MultiChain underpins Windows, Linux and Mac servers
- Gives a basic API and charge line interface that makes it simple to maintain and deploy

Bitcoin to private Blockchain

Many aspects of MultiChain’s design are aimed at enabling smooth transitions between private blockchains and the bitcoin blockchain in either direction:

- MultiChain depends on a fork of Bitcoin Core, the official customer for the bitcoin network. Code changes are restricted, empowering future bitcoin upgrades to be converged in
- It utilizes bitcoin's protocol, transaction and blockchain engineering, with changes just to the handshaking procedure when two nodes initially connect. Every single other element are executed utilizing metadata and changes to the approval rules for transactions and blocks
- Its interface (commandline and API) is completely perfect with that of Bitcoin Core, with all extra usefulness gave by new orders
- It can go about as a node on the consistent bitcoin network (or other bitcoin like systems), by means of a basic convention setting in the per blockchain arrangement record
- Its multicurrency and informing highlights (see later) work comparably to the CoinSpark convention for upgrading bitcoin transactions.

Aim of Multichain

The main aim of Multichain is:

To guarantee that visibility of Blockchain's action must be kept inside the picked members

To control over which transactions are allowed

To enable more secure mining with proof of work and its associated cost

The Blockchain framework just stores transactions identified with members

The Hand-Shaking process

MultiChain uses private key cryptography property to restrict blockchain access to a list of permitted users, by expanding the "handshaking" process that occurs when two blockchain nodes connect.

Every node displays its way of life as an open address on the allowed list.

Every node confirms that alternate's address is without anyone else allowed list .

Every node sends a test message to the next gathering.

Every node sends back a mark of the test message giving their responsibility for private key relating to people in general address they exhibited.

On the off chance that any of the nodes don't have fulfilling comes about, at that point they prematurely end the distributed association.

Good MultiChain use cases

Permission blockchain
Validation by consensus, not proof of work

01

Full asset lifecycle
Issuance, transfer, exchange, escrow, reissuance, redemption, destruction

02

General storage and search
64 mb of data per transaction
Streams: key-value, identity, time series

03

MultiChain permissions

Interface with network

Send and receive transactions

Keep in touch with a stream

Issue resources

Make streams

Add block to chain

Change authorizations by agreement

MultiChain assets

- No requirement for keen contracts
- Flexible resource metadata
- Permissioned take after on issuance
- Atomic multi-resource payments
- Multi-way atomic resource trades
- Multi-signatures for security + escrow
- Buy in to resource inquiry transactions

MultiChain Streams

MultiChain Streams

“Streams can be used to implement different types of databases. Let’s learn more about multichain streams.”

MultiChain Streams

Streams provide a natural abstraction for blockchain use cases which focus on general data retrieval, time stamping and archiving

Any number of streams can be created in a MultiChain blockchain, and each stream acts as an independent append-only collection of items

Nodes choose which streams to index

Streams can be used to implement three different types of databases on a chain:

- A key-value database or document store, in the style of NoSQL.
- A time series database, which focuses on the ordering of entries.
- An identity-driven database where entries are classified according to their author

Streams Basics

Each item in a stream has the following characteristics:

One or more publishers who have digitally signed that item.

A discretionary key for helpful later recovery.

A few information, which can run from a little bit of content to numerous megabytes of crude double.

A timestamp, which is taken from the header of the block in which the thing is affirmed.

Retrieving from Streams

If a node is subscribed to a stream, information can be retrieved from that stream in a number of ways:

Retrieving items from the stream in order.

Retrieving items with a particular key.

Retrieving items signed by a particular publisher.

Listing the keys used in a stream, with item counts for each key.

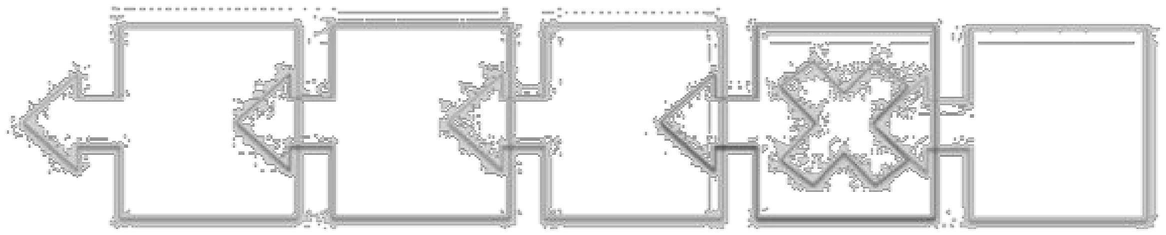
Listing the publishers in a stream, with item counts.

Consensus Model

Each block is digitally signed by creator

- Only permissioned parties can "mine"

Mining diversity for distributed consensus



No proof-of-work or digital money

Multichain Flexibility

45+ blockchain parameters

- Block size/time, permissioning, admin consensus, mining, optional native currency

Authorizations can change after some time

Resources: reissuance and devastation

Streams: hubs take after their interests

Custom metadata all around

Bound together JSON-RPC API for applications

Deployment Options

Environment agnostic

- Self-hosted in data center
- Public or private cloud
- Accessed as a service

Nodes added simply and quickly

API cleanly separates app from node

Shared administration model

- Smooth governance transitions

Security in Multichain

Forked from Bitcoin Core
- $40B+ and more than 8 years presentation to web

Full multisignature bolster

Outer key administration
- Bitcoin equipment security modules

Agreement over information not execution
- Avoids Ethereum-style hard forks
- No need for fees to stop runaway

Speed & Scalability

Millions of addresses, assets, streams

Unlimited transactions / stream items

1000 tps (1.0 bets 2 + mid-range server)

Includes signature verification + transaction processing i.e. real Byzantine tolerance

Unlimited nodes in network

Block time as low as 2 seconds

Process of Mining in Multichain Technology

The Multichain defines miners to an identifiable set of entities. It introduces a criterion known a mining diversity that binds with 0<= mining diversity <=1

The efficacy of block is verifiable by performing the following:

- Apply the changes in permissions set by transactions and the block respectively
- Count the total approved miners set after these variations in the block
- Multiply the number of miners by mining diversity and round up to attain left spacing

This put the round-robin plan for impact in which the allowed mineworkers must make obstructs in turn in request to produce a legitimate blockchain

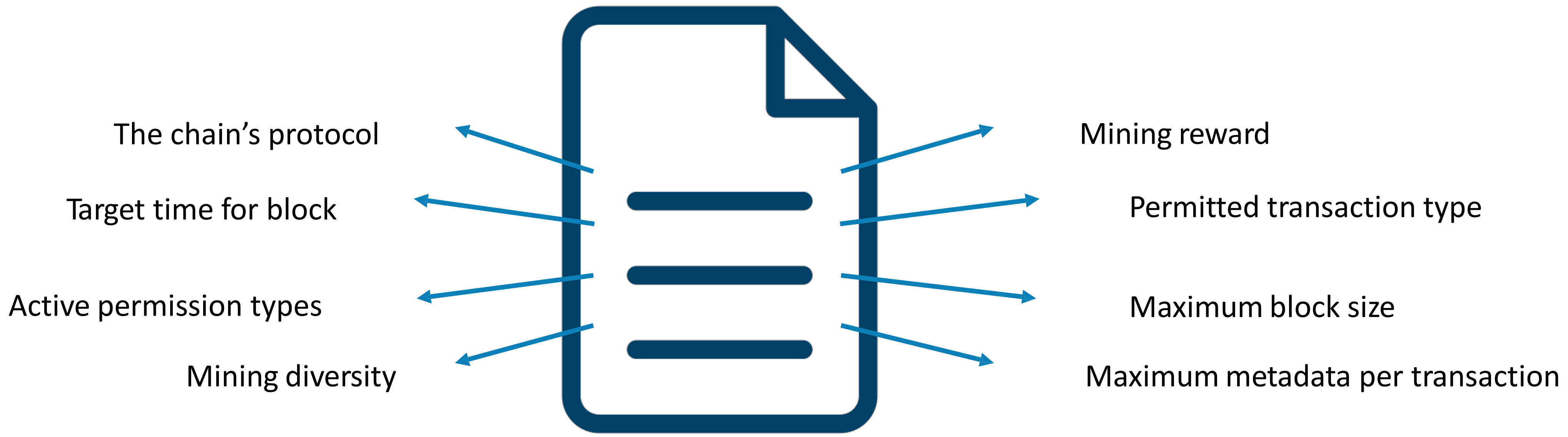
The mineworkers make obstructs in a turn in this calendar to produce a precise Blockchain

The mining decent variety measure builds up the unbending nature of the plan

The "one" esteem outlines that the pivot incorporates each allowed digger. While zero limitations implies non limitations by any means

The params.dat file

You can set these value in params.dat file. The params.dat file contains the whole configuration like:



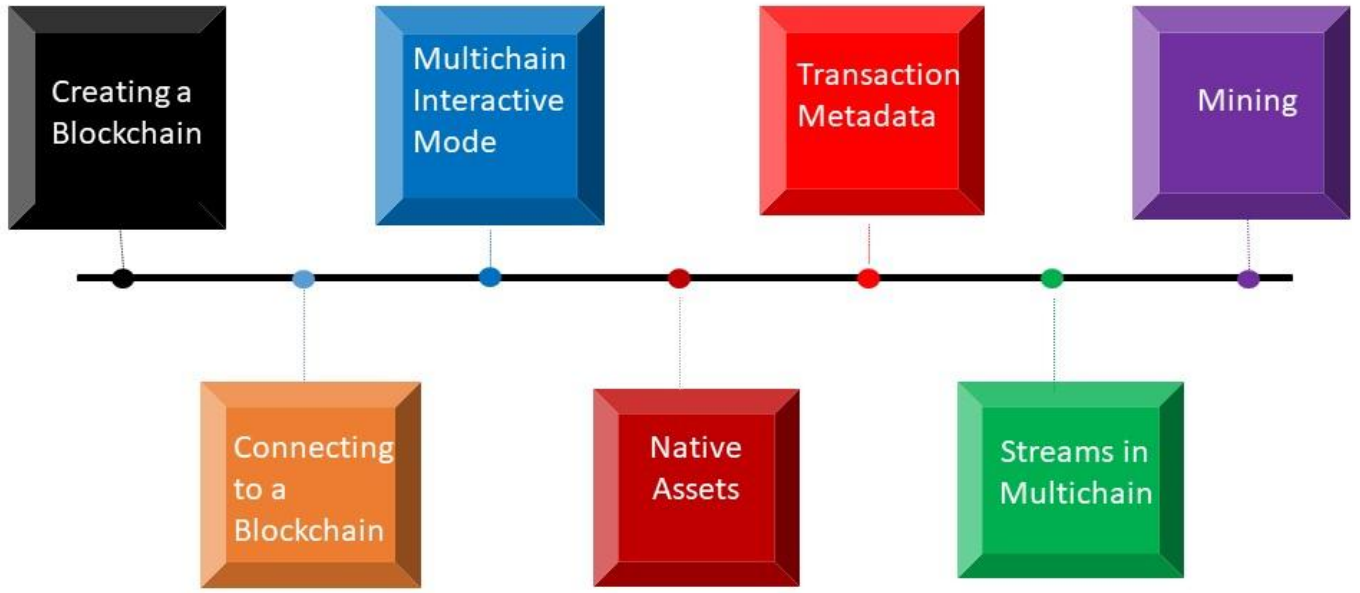
Create & deploy private blockchain

Steps to create & deploy private blockchain

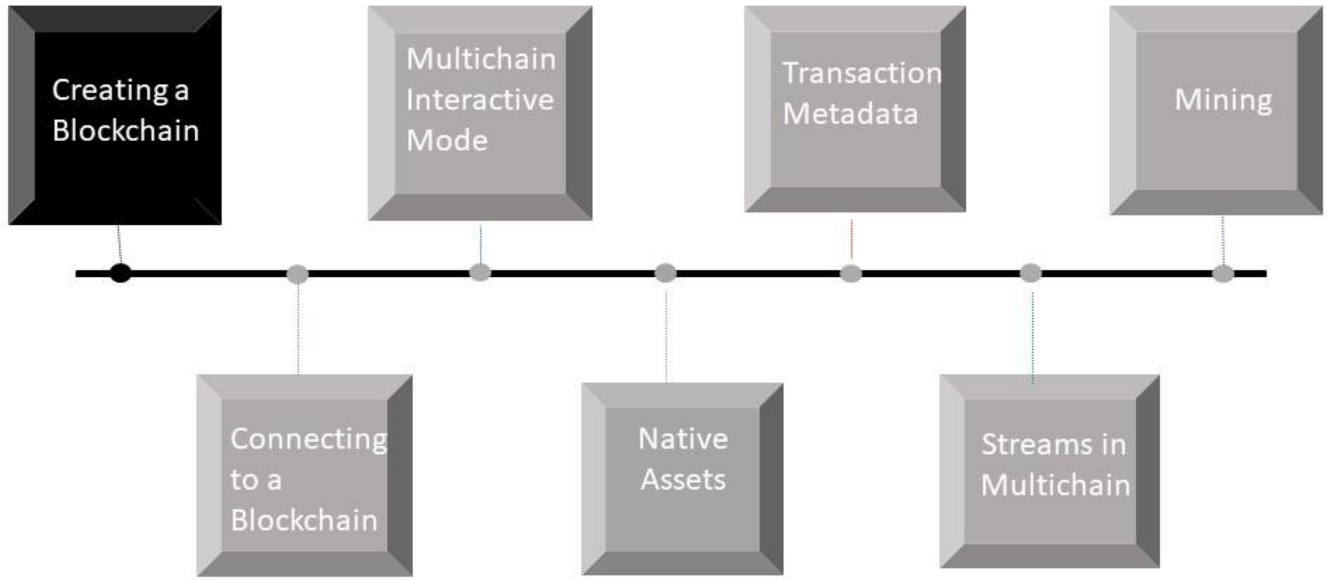
“Now, we are going to learn steps to create & deploy private blockchain.”

Steps to create & deploy private blockchain

Following are the steps to create & deploy private blockchain:

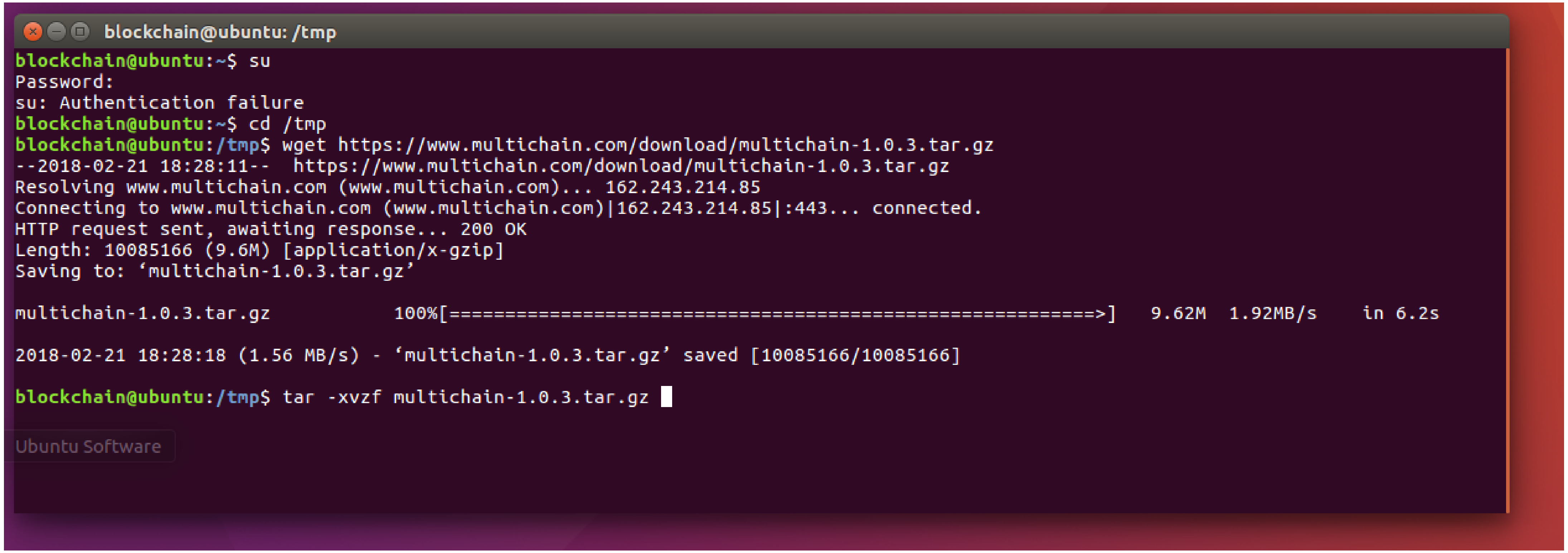


Step 1: Creating a blockchain



Download and Install MultiChain

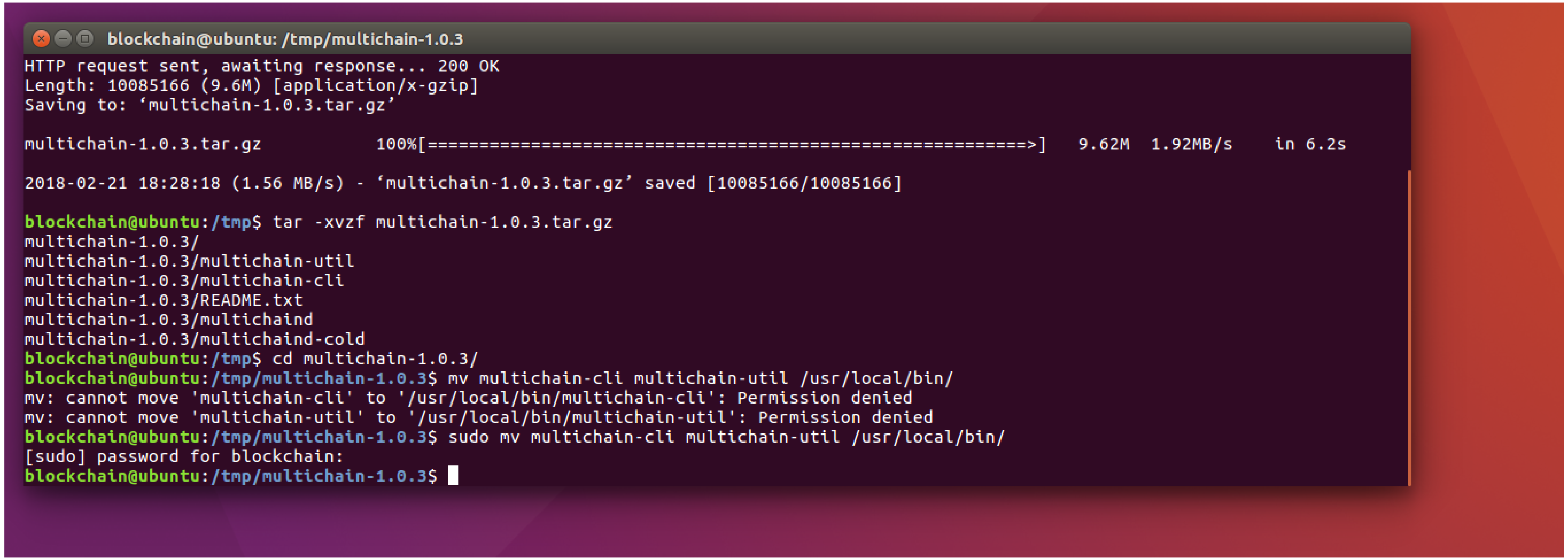
```
su (enter root password)
cd /tmp
wget https://www.multichain.com/download/multichain-1.0.3.tar.gz
```



```
blockchain@ubuntu: /tmp
blockchain@ubuntu:~$ su
Password:
su: Authentication failure
blockchain@ubuntu:~$ cd /tmp
blockchain@ubuntu:/tmp$ wget https://www.multichain.com/download/multichain-1.0.3.tar.gz
--2018-02-21 18:28:11--  https://www.multichain.com/download/multichain-1.0.3.tar.gz
Resolving www.multichain.com (www.multichain.com)... 162.243.214.85
Connecting to www.multichain.com (www.multichain.com)|162.243.214.85|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10085166 (9.6M) [application/x-gzip]
Saving to: ‘multichain-1.0.3.tar.gz’

multichain-1.0.3.tar.gz           100%[===========================================================>]   9.62M  1.92MB/s    in 6.2s

2018-02-21 18:28:18 (1.56 MB/s) - ‘multichain-1.0.3.tar.gz’ saved [10085166/10085166]

blockchain@ubuntu:/tmp$ tar -xvzf multichain-1.0.3.tar.gz
```

Download and Install MultiChain

```
tar -xvzf multichain-1.0.3.tar.gz
cd multichain-1.0.3
mv multichaind multichain-cli multichain-util /usr/local/bin
```



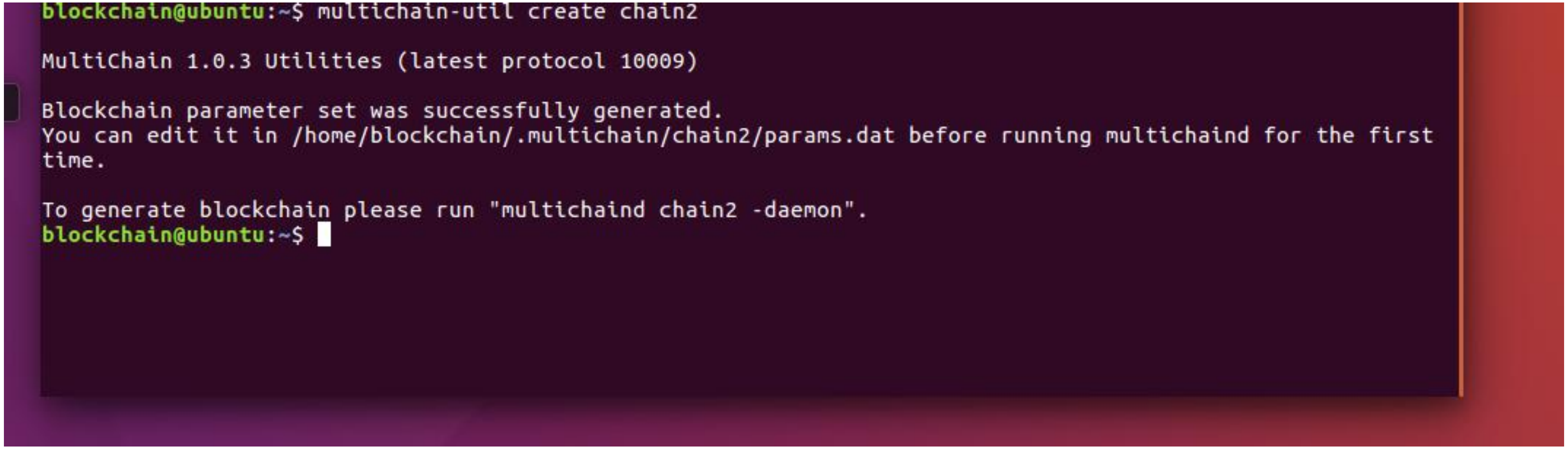
blockchain@ubuntu: /tmp/multichain-1.0.3

```
HTTP request sent, awaiting response... 200 OK
Length: 10085166 (9.6M) [application/x-gzip]
Saving to: ‘multichain-1.0.3.tar.gz’

multichain-1.0.3.tar.gz          100%[===========================================================>]   9.62M  1.92MB/s    in 6.2s

2018-02-21 18:28:18 (1.56 MB/s) - ‘multichain-1.0.3.tar.gz’ saved [10085166/10085166]

blockchain@ubuntu:/tmp$ tar -xvzf multichain-1.0.3.tar.gz
multichain-1.0.3/
multichain-1.0.3/multichain-util
multichain-1.0.3/multichain-cli
multichain-1.0.3/README.txt
multichain-1.0.3/multichaind
multichain-1.0.3/multichaind-cold
blockchain@ubuntu:/tmp$ cd multichain-1.0.3/
blockchain@ubuntu:/tmp/multichain-1.0.3$ mv multichain-cli multichain-util /usr/local/bin/
mv: cannot move 'multichain-cli' to '/usr/local/bin/multichain-cli': Permission denied
mv: cannot move 'multichain-util' to '/usr/local/bin/multichain-util': Permission denied
blockchain@ubuntu:/tmp/multichain-1.0.3$ sudo mv multichain-cli multichain-util /usr/local/bin/
[sudo] password for blockchain:
blockchain@ubuntu:/tmp/multichain-1.0.3$
```

To Create a blockchain

First we will create a new blockchain named chain1 (you can give the name of your choice).

On the first server, run this command: `multichain-util create chain2`



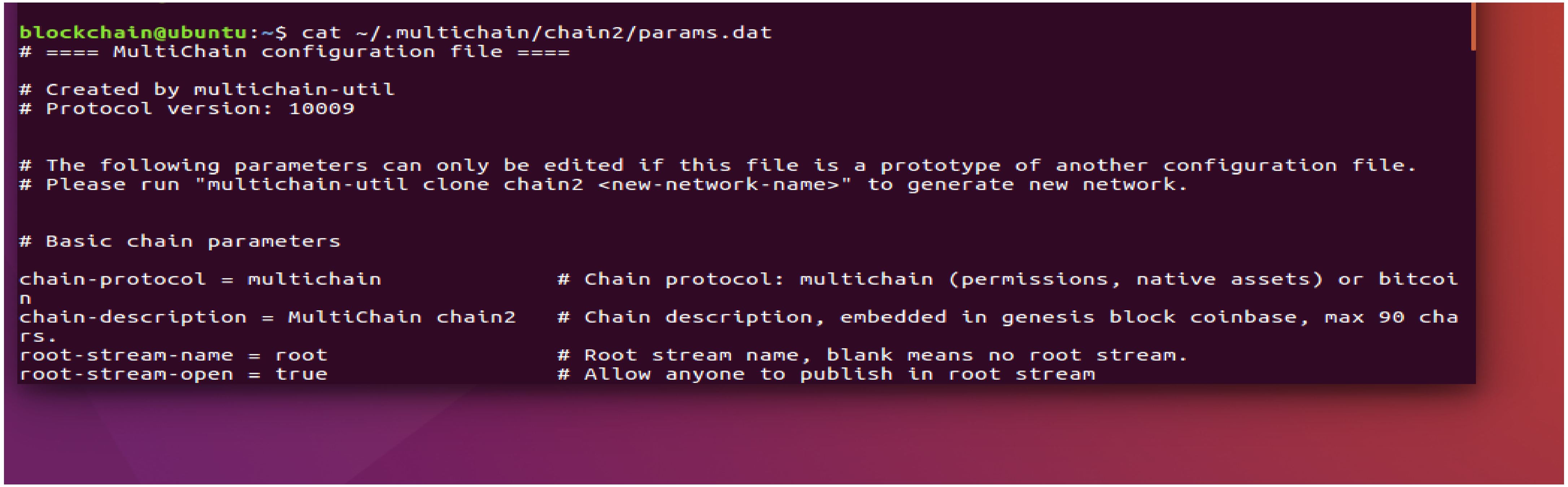
```
blockchain@ubuntu:~$ multichain-util create chain2

MultiChain 1.0.3 Utilities (latest protocol 10009)

Blockchain parameter set was successfully generated.
You can edit it in /home/blockchain/.multichain/chain2/params.dat before running multichaind for the first
time.

To generate blockchain please run "multichaind chain2 -daemon".
blockchain@ubuntu:~$
```

View default settings

To view the blockchain's default settings run the following commands:

```
cat ~/.multichain/chain1/params.dat
```

(Above command will list the parameters and settings of the blockchain)



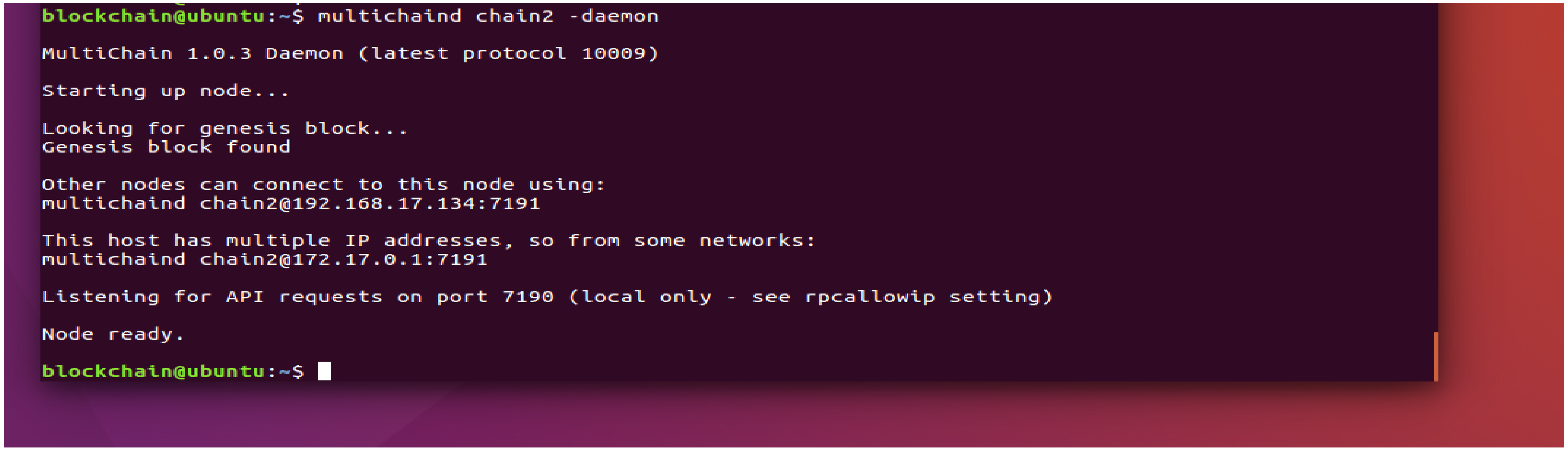
```
blockchain@ubuntu:~$ cat ~/.multichain/chain2/params.dat
# ==== MultiChain configuration file ====

# Created by multichain-util
# Protocol version: 10009


# The following parameters can only be edited if this file is a prototype of another configuration file.
# Please run "multichain-util clone chain2 <new-network-name>" to generate new network.


# Basic chain parameters

chain-protocol = multichain              # Chain protocol: multichain (permissions, native assets) or bitcoi
n
chain-description = MultiChain chain2    # Chain description, embedded in genesis block coinbase, max 90 cha
rs.
root-stream-name = root                  # Root stream name, blank means no root stream.
root-stream-open = true                  # Allow anyone to publish in root stream
```

***Note:** List shown here is just a specimen. As the list was too long to display on the ppt*

Initializing the blockchain

Run the following command to initialize the blockchain, including mining the genesis block:

```
multichaind chain1 -daemon
```



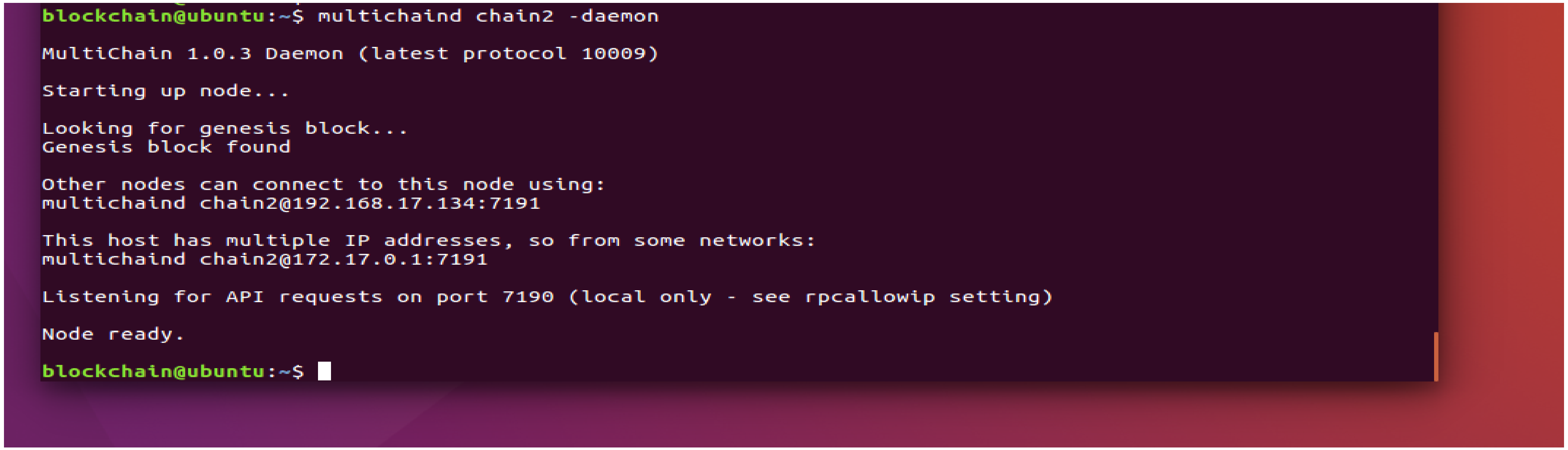
```
blockchain@ubuntu:~$ multichaind chain2 -daemon

MultiChain 1.0.3 Daemon (latest protocol 10009)

Starting up node...

Looking for genesis block...
Genesis block found

Other nodes can connect to this node using:
multichaind chain2@192.168.17.134:7191

This host has multiple IP addresses, so from some networks:
multichaind chain2@172.17.0.1:7191

Listening for API requests on port 7190 (local only - see rpcallowip setting)

Node ready.

blockchain@ubuntu:~$
```

Note: *You should be told that the server has started and then after a few seconds, that the genesis block was found. You should also be given the node address that others can use to connect to this chain*

Initializing the blockchain

Run the following command to initialize the blockchain, including mining the genesis block:

```
multichaind chain1 -daemon
```



```
blockchain@ubuntu:~$ multichaind chain2 -daemon

MultiChain 1.0.3 Daemon (latest protocol 10009)

Starting up node...

Looking for genesis block...
Genesis block found

Other nodes can connect to this node using:
multichaind chain2@192.168.17.134:7191

This host has multiple IP addresses, so from some networks:
multichaind chain2@172.17.0.1:7191

Listening for API requests on port 7190 (local only - see rpcallowip setting)

Node ready.

blockchain@ubuntu:~$
```

Note: *You should be told that the server has started and then after a few seconds, that the genesis block was found. You should also be given the node address that others can use to connect to this chain*

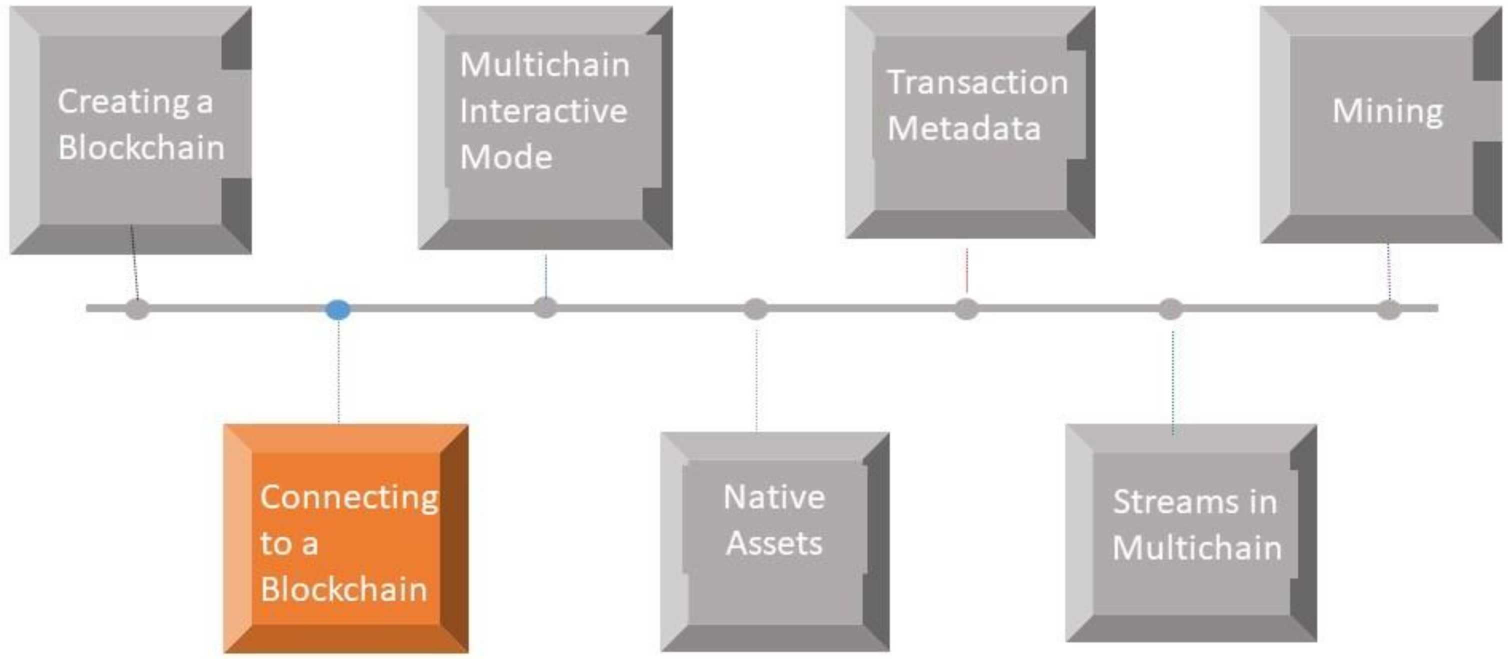
Connecting to a Blockchain

Connecting to a Blockchain

“Let’s learn about Connecting to a Blockchain

.”

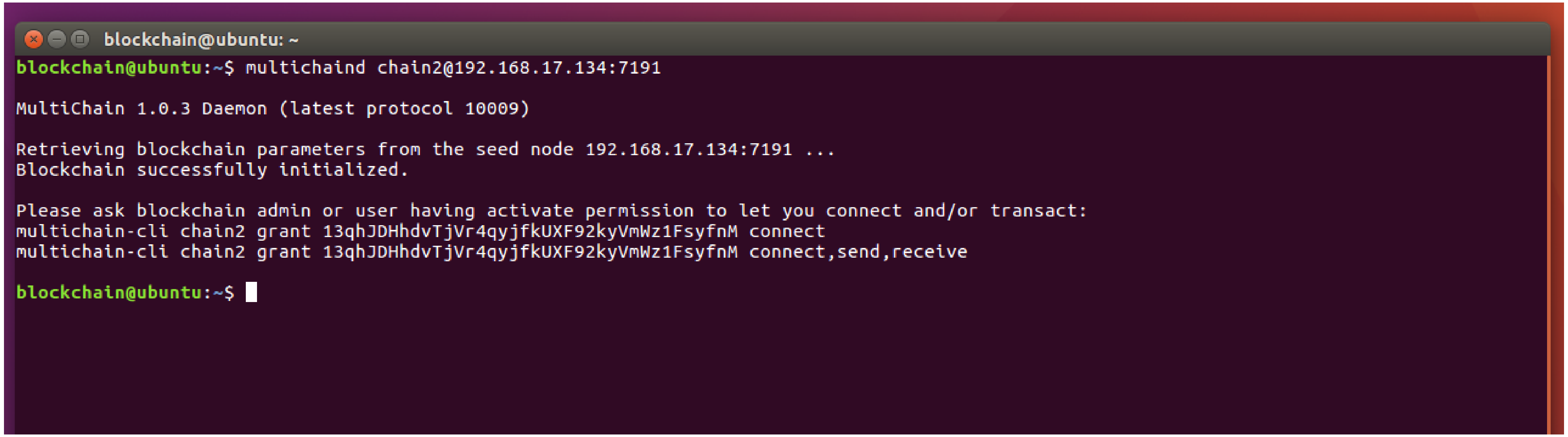
Step 2: Connecting to a Blockchain



Connecting to a Blockchain from the second server

Now we'll connect to this blockchain from elsewhere. On the second server, run the following:

```
multichaind chain1@[ip-address]:[port
```



```
blockchain@ubuntu: ~
blockchain@ubuntu:~$ multichaind chain2@192.168.17.134:7191

MultiChain 1.0.3 Daemon (latest protocol 10009)

Retrieving blockchain parameters from the seed node 192.168.17.134:7191 ...
Blockchain successfully initialized.

Please ask blockchain admin or user having activate permission to let you connect and/or transact:
multichain-cli chain2 grant 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM connect
multichain-cli chain2 grant 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM connect,send,receive

blockchain@ubuntu:~$ 
```

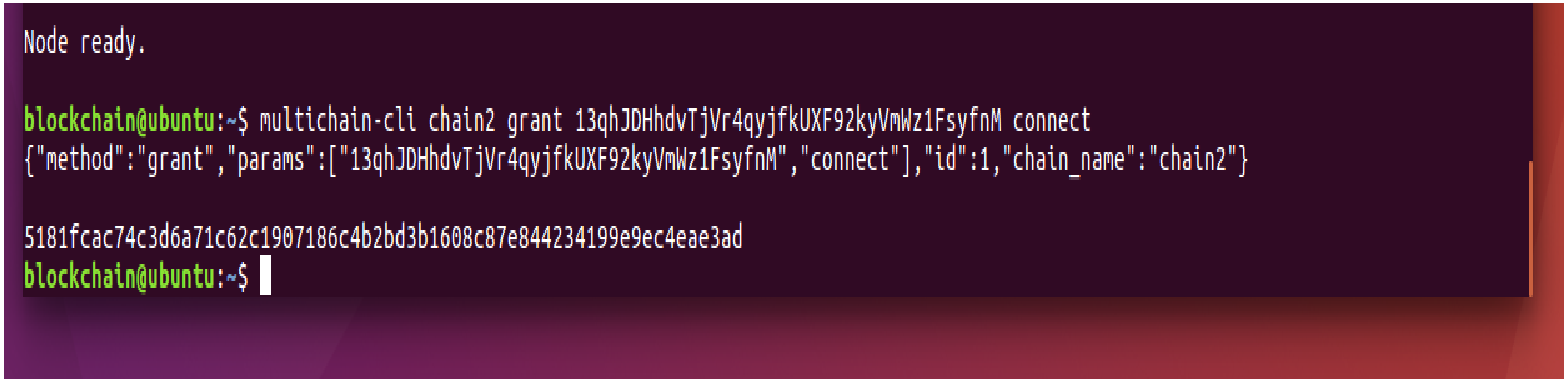
***Note:** You should be told that the blockchain was successfully initialized, but you do not have permission to connect. You should also be shown a message containing an address in this node's wallet.*

Adding Connection Permission for the address

Add connection permissions for this address:

multichain-cli chain1 grant 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnMconnect

Above command will grant permission to connect to the specified address



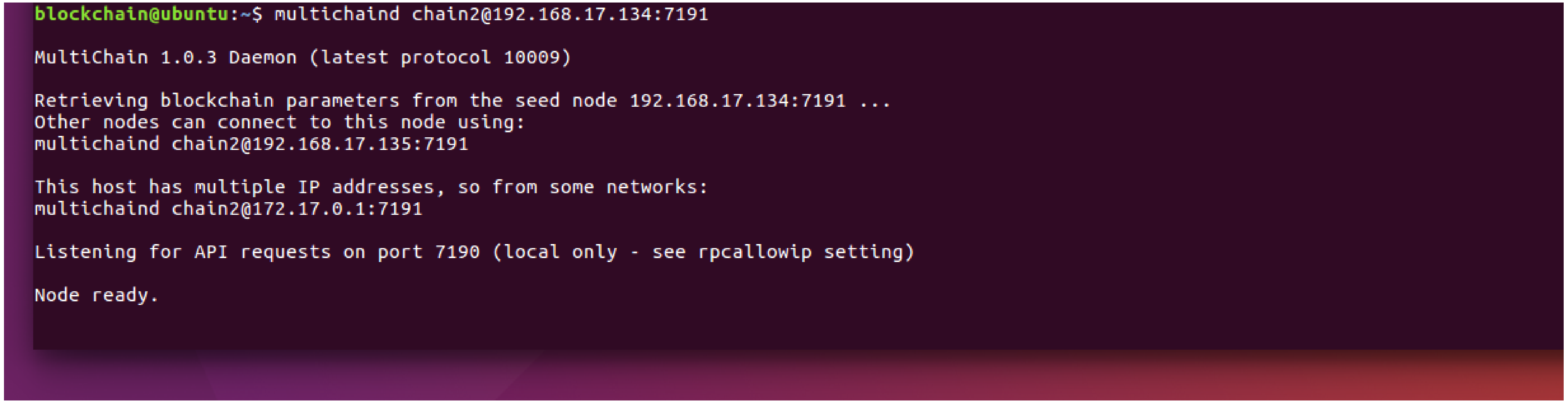
```
Node ready.

blockchain@ubuntu:~$ multichain-cli chain2 grant 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM connect
{"method":"grant","params":["13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM","connect"],"id":1,"chain_name":"chain2"}

5181fcac74c3d6a71c62c1907186c4b2bd3b1608c87e844234199e9ec4eae3ad
blockchain@ubuntu:~$
```

***Note:** The address here is unique for this particular blockchain. It will be different when you run your Blockchain*

Reconnecting from the second server

Now try reconnecting again from the second server:

multichaind chain1 -daemon



```
blockchain@ubuntu:~$ multichaind chain2@192.168.17.134:7191

MultiChain 1.0.3 Daemon (latest protocol 10009)

Retrieving blockchain parameters from the seed node 192.168.17.134:7191 ...
Other nodes can connect to this node using:
multichaind chain2@192.168.17.135:7191

This host has multiple IP addresses, so from some networks:
multichaind chain2@172.17.0.1:7191

Listening for API requests on port 7190 (local only - see rpcallowip setting)

Node ready.
```

Note: *You should be shown a message that the node was started, and it should display this second node's address*

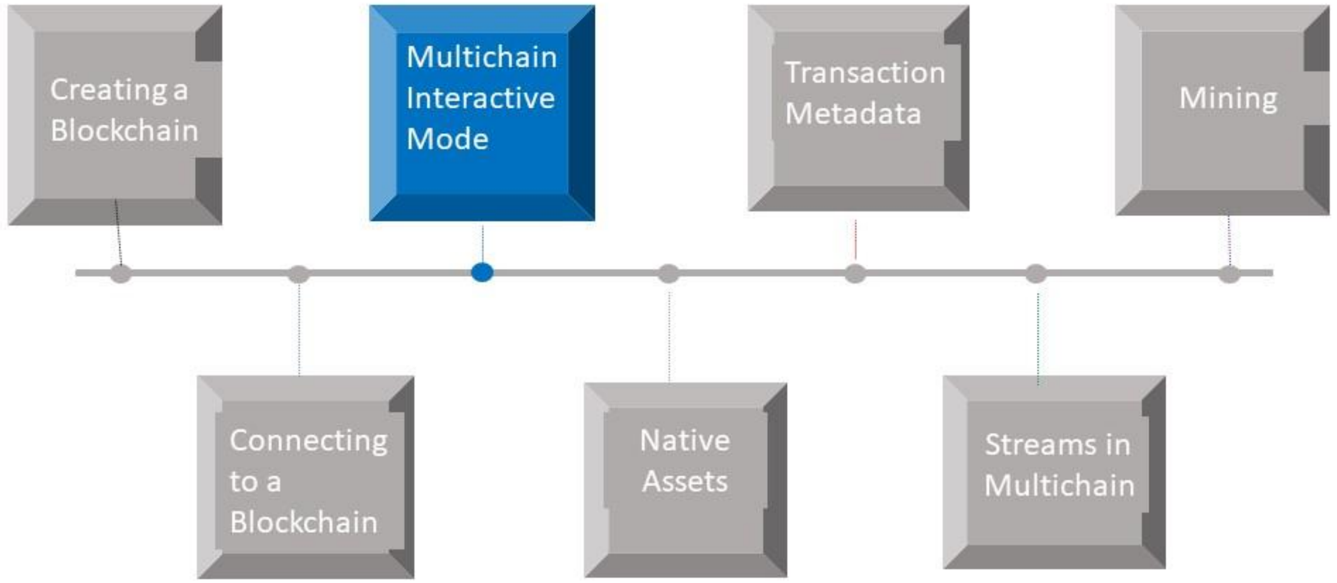
Multichain Interactive Mode

Multichain Interactive Mode

“Let's learn about Multichain Interactive Mode.”

Step 3: Multichain Interactive Mode

Before we proceed, let's enter interactive mode so we can issue commands without typing multichain-cli chain1 every time

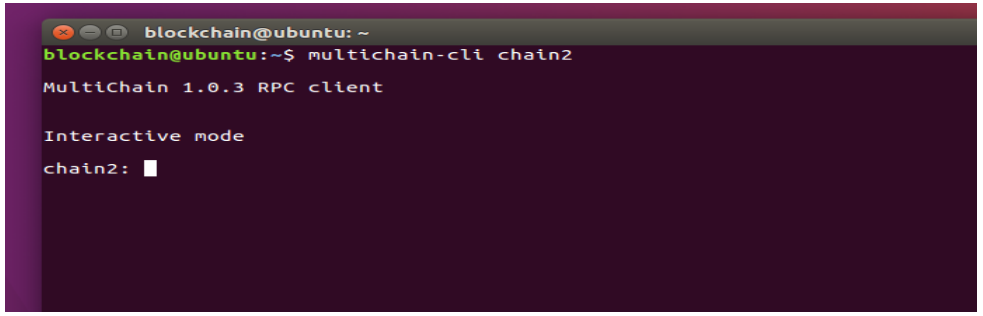


Switching to interactive mode

To enter the multichain interactive mode, on both servers run the following command:

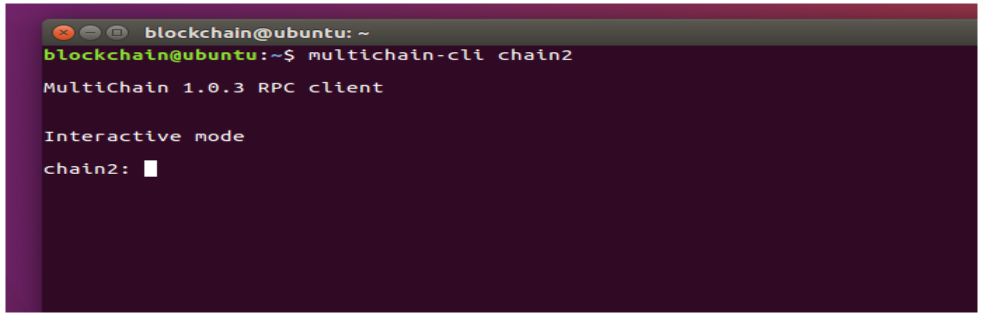
```
multichain-cli chain1
```

Server 1:



```
blockchain@ubuntu: ~
blockchain@ubuntu:~$ multichain-cli chain2
MultiChain 1.0.3 RPC client

Interactive mode
chain2:
```

Server 2:



```
blockchain@ubuntu: ~
blockchain@ubuntu:~$ multichain-cli chain2
MultiChain 1.0.3 RPC client

Interactive mode
chain2:
```

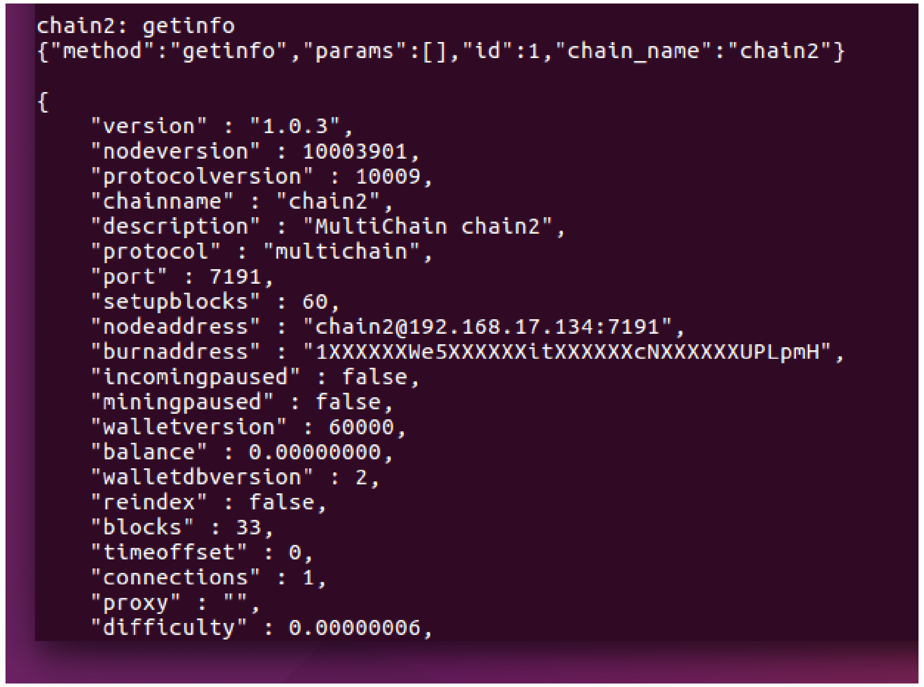
Note: *Now that the blockchain is working on two nodes, you can run the commands in this section on either or both*

Getting the general information

To get general information of the blockchain execute the following :

getinfo

Above command will provide the general information about the blockchain



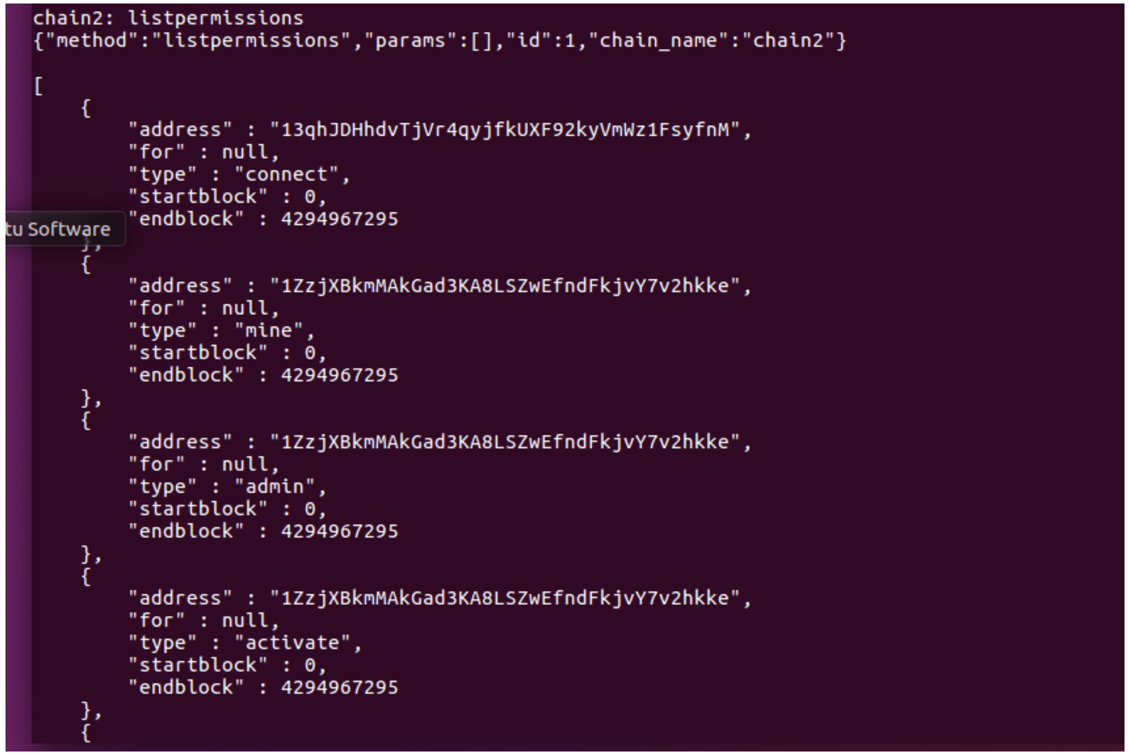
```
chain2: getinfo
{"method":"getinfo","params":[],"id":1,"chain_name":"chain2"}

{
    "version" : "1.0.3",
    "nodeversion" : 10003901,
    "protocolversion" : 10009,
    "chainname" : "chain2",
    "description" : "MultiChain chain2",
    "protocol" : "multichain",
    "port" : 7191,
    "setupblocks" : 60,
    "nodeaddress" : "chain2@192.168.17.134:7191",
    "burnaddress" : "1XXXXXXWe5XXXXXXitXXXXXXcNXXXXXXUPLpmH",
    "incomingpaused" : false,
    "miningpaused" : false,
    "walletversion" : 60000,
    "balance" : 0.00000000,
    "walletdbversion" : 2,
    "reindex" : false,
    "blocks" : 33,
    "timeoffset" : 0,
    "connections" : 1,
    "proxy" : "",
    "difficulty" : 0.00000006,
```

Permissions currently assigned

To Show all permissions currently assigned execute the following:

listpermissions

Above command will list all the permissions currently assigned to the address



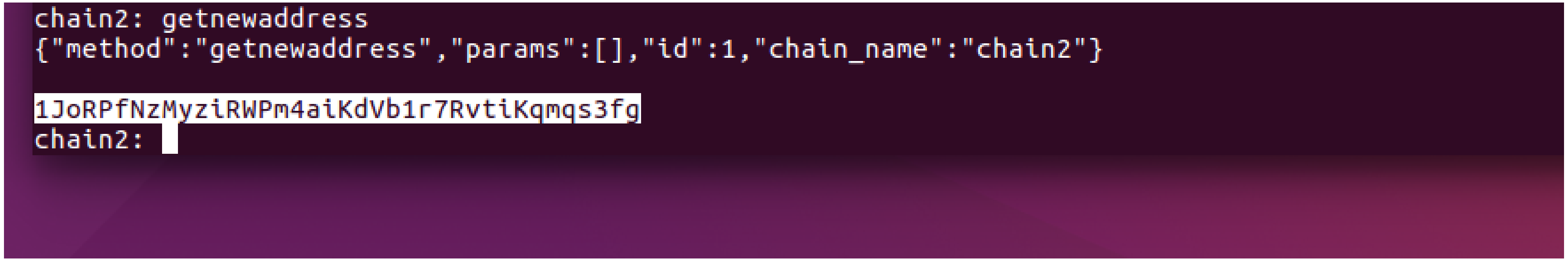
```
chain2: listpermissions
{"method":"listpermissions","params":[],"id":1,"chain_name":"chain2"}

[
    {
        "address" : "13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM",
        "for" : null,
        "type" : "connect",
        "startblock" : 0,
        "endblock" : 4294967295
    },
    {
        "address" : "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke",
        "for" : null,
        "type" : "mine",
        "startblock" : 0,
        "endblock" : 4294967295
    },
    {
        "address" : "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke",
        "for" : null,
        "type" : "admin",
        "startblock" : 0,
        "endblock" : 4294967295
    },
    {
        "address" : "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke",
        "for" : null,
        "type" : "activate",
        "startblock" : 0,
        "endblock" : 4294967295
    },
    {
```

Creating new Address in the wallet

To create a new address in the wallet execute the following:

getnewaddress

Above command will create a new address in the wallet



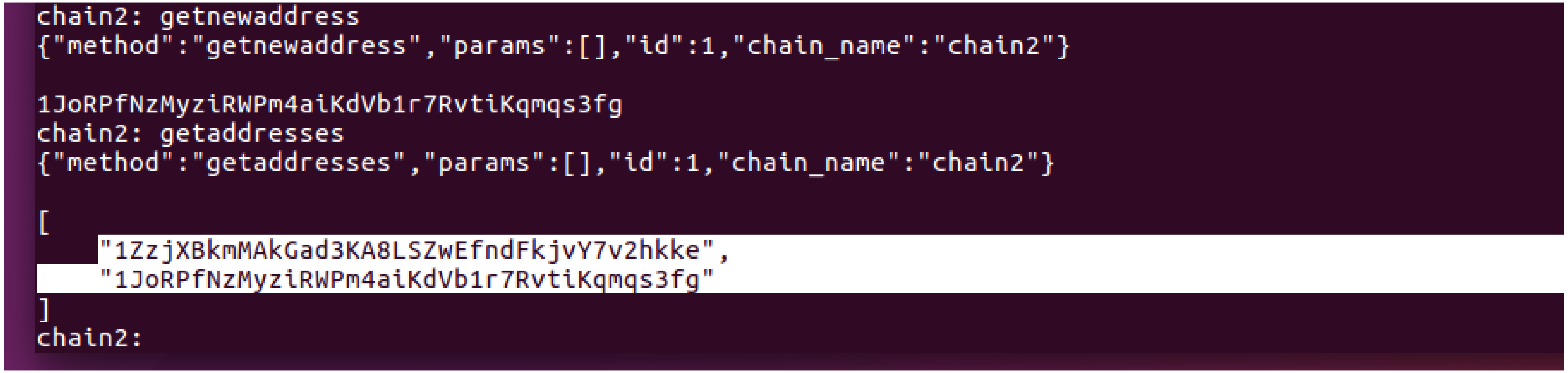
```
chain2: getnewaddress
{"method":"getnewaddress","params":[],"id":1,"chain_name":"chain2"}

1JoRPfNzMyziRWPm4aiKdVb1r7RvtiKqmqs3fg
chain2: 
```

List all addresses

To list all addresses in the wallet execute the following:

```
getaddresses
```

Above command will list all the addresses in the wallet



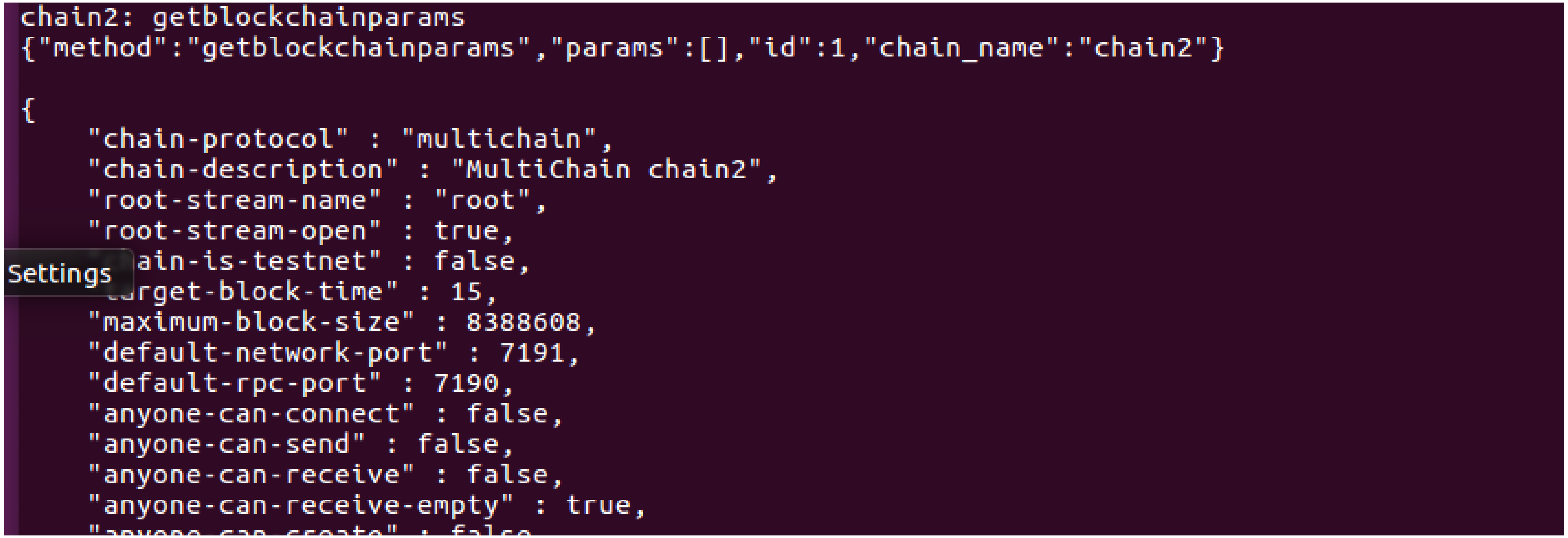
```
chain2: getnewaddress
{"method":"getnewaddress","params":[],"id":1,"chain_name":"chain2"}

1JoRPfNzMyziRWPm4aiKdVb1r7RvtiKqmqs3fg
chain2: getaddresses
{"method":"getaddresses","params":[],"id":1,"chain_name":"chain2"}

[
    "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke",
    "1JoRPfNzMyziRWPm4aiKdVb1r7RvtiKqmqs3fg"
]
chain2:
```

Getting blockchain parameters

Getting blockchain parameters

getblockchainparams

Above command will display the the parameters of this blockchain



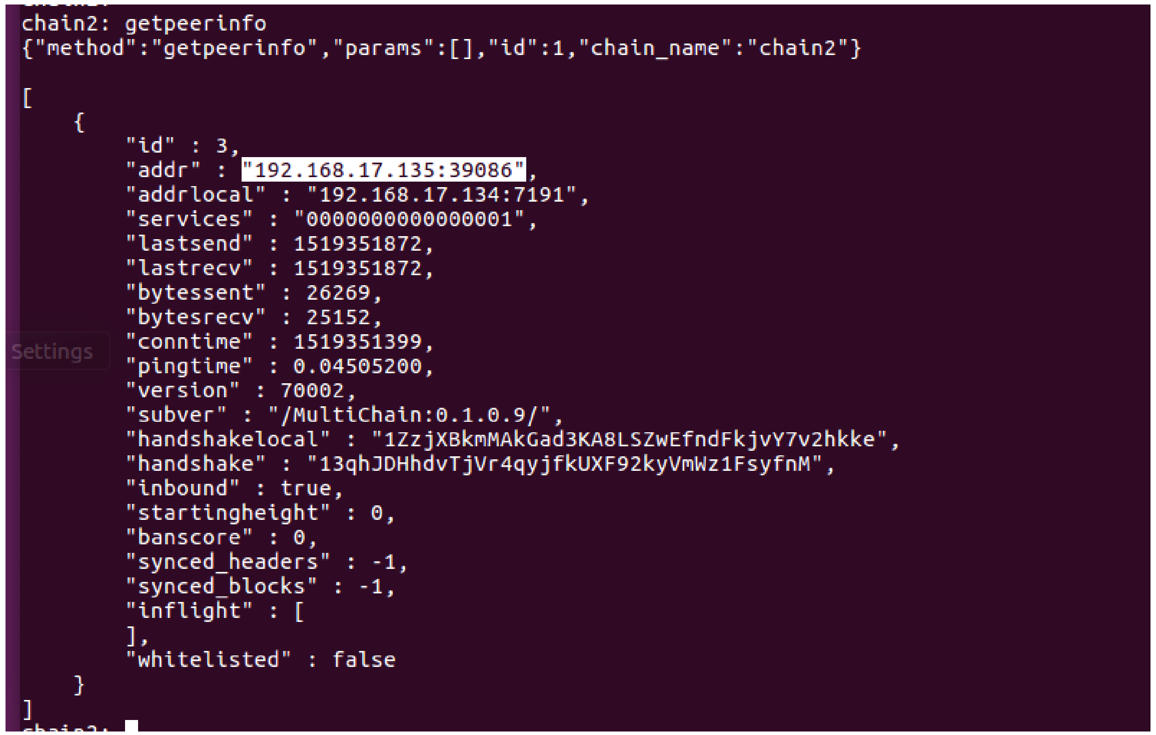
```
chain2: getblockchainparams
{"method":"getblockchainparams","params":[],"id":1,"chain_name":"chain2"}

{
    "chain-protocol" : "multichain",
    "chain-description" : "MultiChain chain2",
    "root-stream-name" : "root",
    "root-stream-open" : true,
       ain-is-testnet" : false,
       rget-block-time" : 15,
    "maximum-block-size" : 8388608,
    "default-network-port" : 7191,
    "default-rpc-port" : 7190,
    "anyone-can-connect" : false,
    "anyone-can-send" : false,
    "anyone-can-receive" : false,
    "anyone-can-receive-empty" : true,
    "anyone-can-create" : false
```

Getting a list of connected peers

For each node, get a list of connected peers by executing the following:

getpeerinfo

Above command will list the connected peers



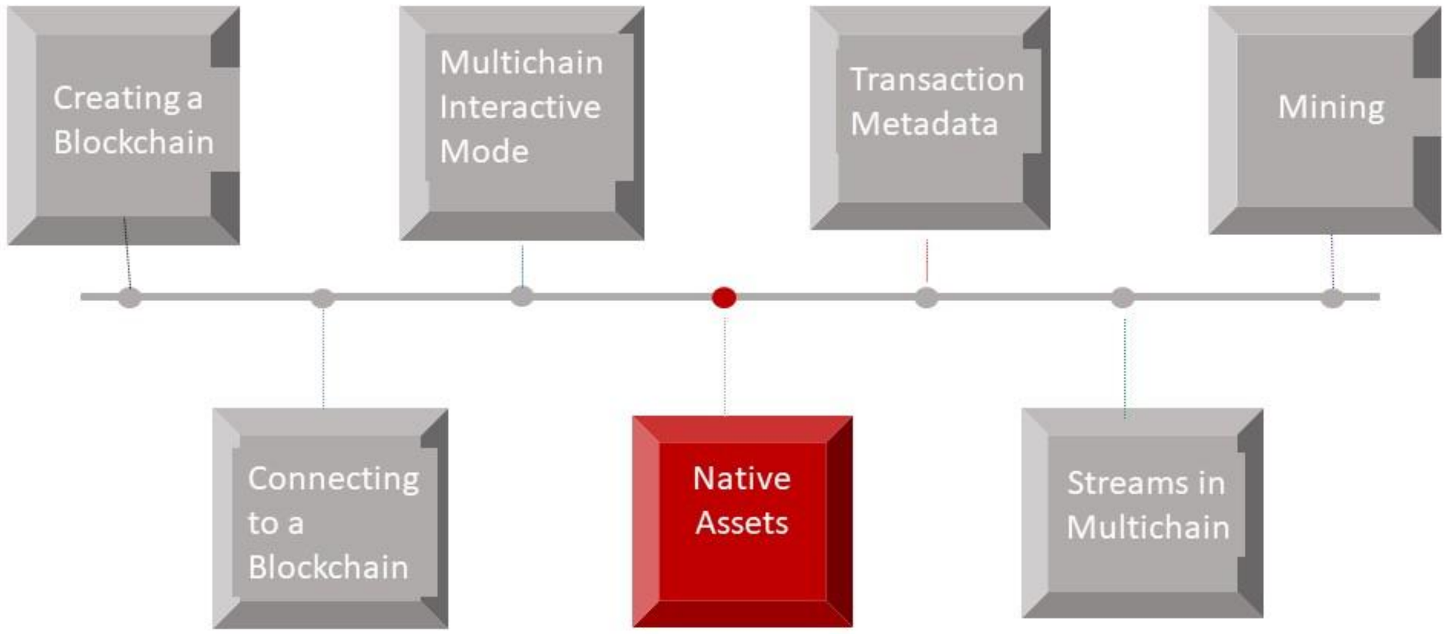
```
chain2: getpeerinfo
{"method":"getpeerinfo","params":[],"id":1,"chain_name":"chain2"}

[
    {
        "id" : 3,
        "addr" : "192.168.17.135:39086",
        "addrlocal" : "192.168.17.134:7191",
        "services" : "0000000000000001",
        "lastsend" : 1519351872,
        "lastrecv" : 1519351872,
        "bytessent" : 26269,
        "bytesrecv" : 25152,
        "conntime" : 1519351399,
        "pingtime" : 0.04505200,
        "version" : 70002,
        "subver" : "/MultiChain:0.1.0.9/",
        "handshakelocal" : "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke",
        "handshake" : "13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM",
        "inbound" : true,
        "startingheight" : 0,
        "banscore" : 0,
        "synced_headers" : -1,
        "synced_blocks" : -1,
        "inflight" : [
        ],
        "whitelisted" : false
    }
]
```

Native assets

Native assets

“Let's learn about Native assets.”

Step 4: Native assets

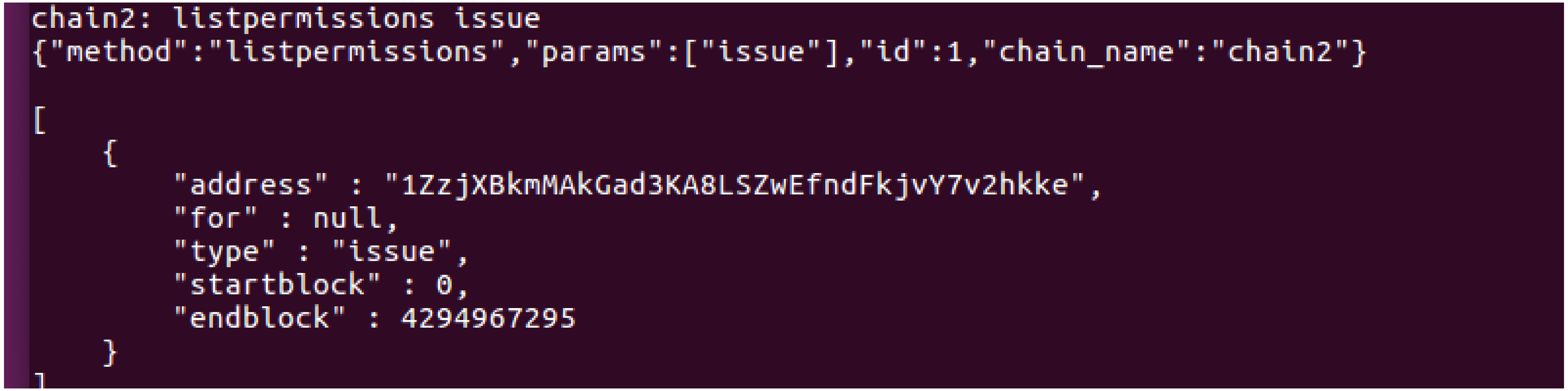


Getting permissions to create Assets

Now we are going to create a new asset and send it between nodes
On the first server, get the address that has the permission to create assets

```
listpermissions issue
```

Above command will list the address that has the permission to create assets



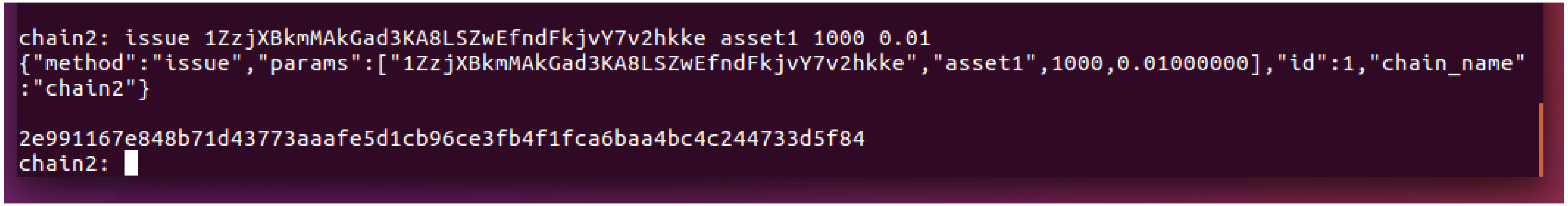
```
chain2: listpermissions issue
{"method":"listpermissions","params":["issue"],"id":1,"chain_name":"chain2"}

[
    {
        "address" : "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke",
        "for" : null,
        "type" : "issue",
        "startblock" : 0,
        "endblock" : 4294967295
    }
]
```

Creating new Asset

Now we'll create a new asset on this node with 1000 units, each of which can be subdivided into 100 parts, sending it to itself
To create a new asset execute the following:

```
issue 1JXy9Va15ikDf4xmE8XGu74ZuxfAnz3Ug6pq4J asset1 1000 0.01
```

Above command will issue asset1 to the specified address



```
chain2: issue 1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke asset1 1000 0.01
{"method":"issue","params":["1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke","asset1",1000,0.01000000],"id":1,"chain_name"
:"chain2"}

2e991167e848b71d43773aaafe5d1cb96ce3fb4f1fca6baa4bc4c244733d5f84
chain2:
```

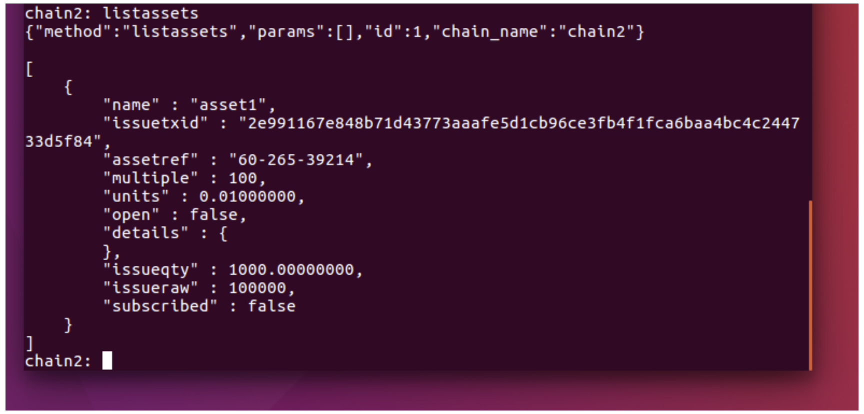
Verifying the Asset

On both servers, verify that the asset named asset1 is listed:

```
listassets
```

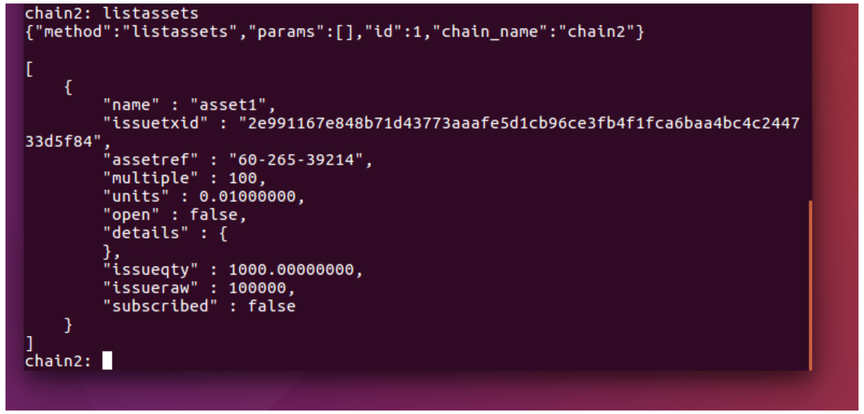
The asset can be seen listed in both the servers

Server 1:



```
chain2: listassets
{"method":"listassets","params":[],"id":1,"chain_name":"chain2"}

[
    {
        "name" : "asset1",
        "issuetxid" : "2e991167e848b71d43773aaafe5d1cb96ce3fb4f1fca6baa4bc4c2447
33d5f84",
        "assetref" : "60-265-39214",
        "multiple" : 100,
        "units" : 0.01000000,
        "open" : false,
        "details" : {
        },
        "issueqty" : 1000.00000000,
        "issueraw" : 100000,
        "subscribed" : false
    }
]
chain2: 
```

Server 2:



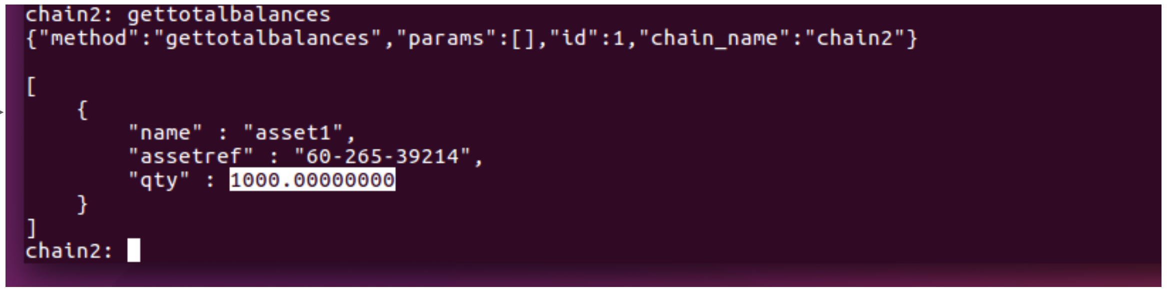
```
chain2: listassets
{"method":"listassets","params":[],"id":1,"chain_name":"chain2"}

[
    {
        "name" : "asset1",
        "issuetxid" : "2e991167e848b71d43773aaafe5d1cb96ce3fb4f1fca6baa4bc4c2447
33d5f84",
        "assetref" : "60-265-39214",
        "multiple" : 100,
        "units" : 0.01000000,
        "open" : false,
        "details" : {
        },
        "issueqty" : 1000.00000000,
        "issueraw" : 100000,
        "subscribed" : false
    }
]
chain2: 
```

Checking Asset balances

Now check the asset balances on each server. The first should show a balance of 1000, and the second should show no assets at all

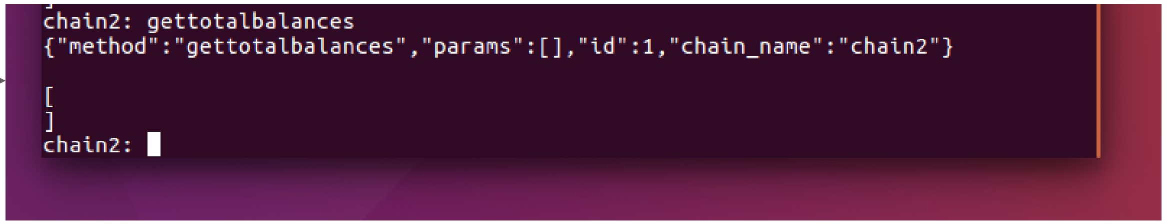
The first server shows the asset balance of 1000, whereas the second server has no asset at all. Hence verified.

Server 1



```
chain2: gettotalbalances
{"method":"gettotalbalances","params":[],"id":1,"chain_name":"chain2"}

[
    {
        "name" : "asset1",
        "assetref" : "60-265-39214",
        "qty" : 1000.00000000
    }
]
chain2: 
```

Server 2



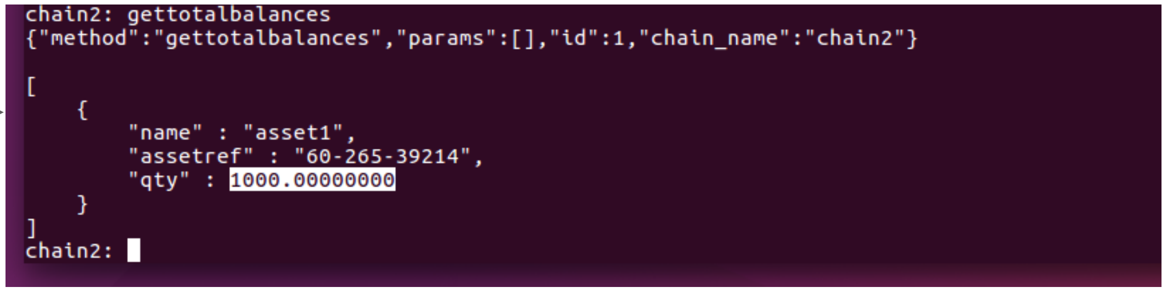
```
chain2: gettotalbalances
{"method":"gettotalbalances","params":[],"id":1,"chain_name":"chain2"}

[
]
chain2: 
```

Sending the Asset

On the first server, now try sending 100 units of the asset to the second server's wallet
To send the units, execute the following command:

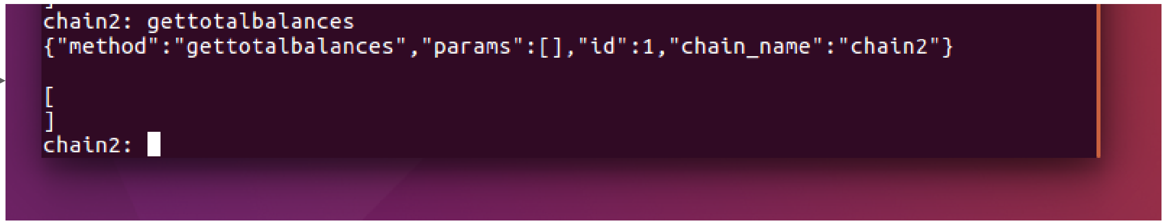
The first server shows the asset balance of 1000, whereas the second server has no asset at all. Hence verified.

Server 1



```
chain2: gettotalbalances
{"method":"gettotalbalances","params":[],"id":1,"chain_name":"chain2"}

[
    {
        "name" : "asset1",
        "assetref" : "60-265-39214",
        "qty" : 1000.00000000
    }
]
chain2: 
```

Server 2



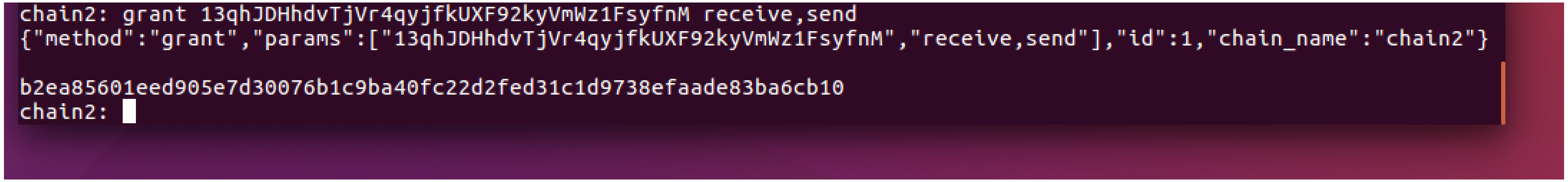
```
chain2: gettotalbalances
{"method":"gettotalbalances","params":[],"id":1,"chain_name":"chain2"}

[
]
chain2: 
```

Adding permissions

Beforehand, you probably observed a blunder that the address does not have get authorizations. So it's a great opportunity to include get and send authorizations
To allow authorizations to the address, execute the accompanying order :

```
grant 13aNooypwXPW5NzuxvzGEDtEG6Hxtq8LhpByd receive, send
```

Above command will grant permission to the address to receive or send the asset



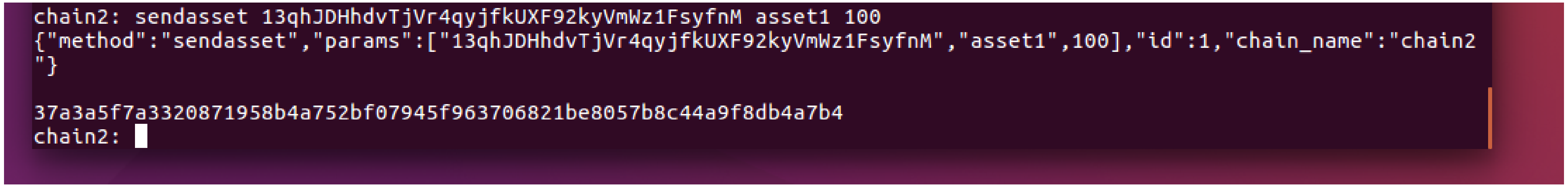
```
chain2: grant 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM receive,send
{"method":"grant","params":["13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM","receive,send"],"id":1,"chain_name":"chain2"}

b2ea85601eed905e7d30076b1c9ba40fc22d2fed31c1d9738efaade83ba6cb10
chain2: 
```

Resending the asset

Now try sending the asset again, and it should go through:

```
sendasset 13aNooypwXPW5NzuxvzGEDtEG6Hxtq8LhpByd asset1 100
```

Above command will now send the asset to the particular address



```
chain2: sendasset 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM asset1 100
{"method":"sendasset","params":["13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM","asset1",100],"id":1,"chain_name":"chain2
"}

37a3a5f7a3320871958b4a752bf07945f963706821be8057b8c44a9f8db4a7b4
chain2: 
```

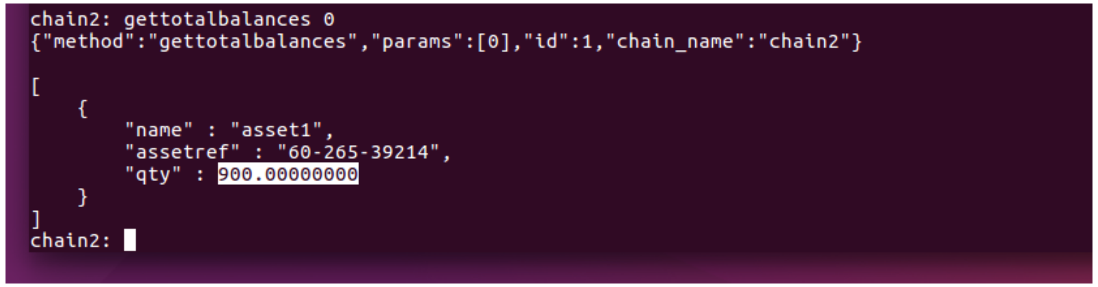
Checking Asset balances

Now check the asset balances on each server, including transactions with zero confirmations. They should be 900 and 100 respectively
To check asset balances, execute the following command:

```
gettotalbalances 0
```

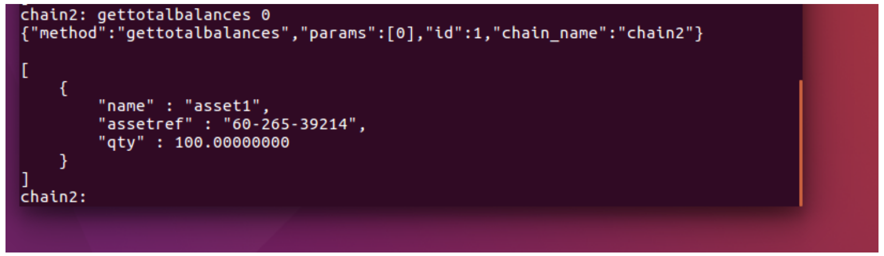
You can see that server 1 asset balance is now 900 and , server 2 has asset balance of 100

Server 1:



```
chain2: gettotalbalances 0
{"method":"gettotalbalances","params":[0],"id":1,"chain_name":"chain2"}

[
    {
        "name" : "asset1",
        "assetref" : "60-265-39214",
        "qty" : 900.00000000
    }
]
chain2:
```

Server 2:



```
chain2: gettotalbalances 0
{"method":"gettotalbalances","params":[0],"id":1,"chain_name":"chain2"}

[
    {
        "name" : "asset1",
        "assetref" : "60-265-39214",
        "qty" : 100.00000000
    }
]
chain2:
```

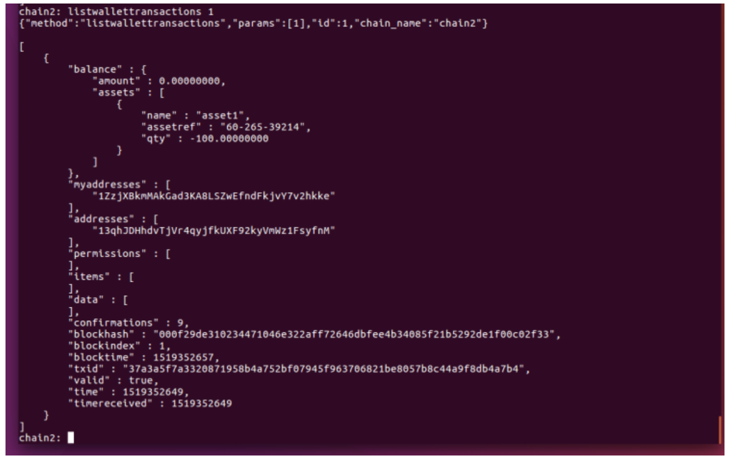
View transaction

You can likewise see the transaction on every node and perceive how it influenced their equalizations

To view the transaction, execute the following command on each node

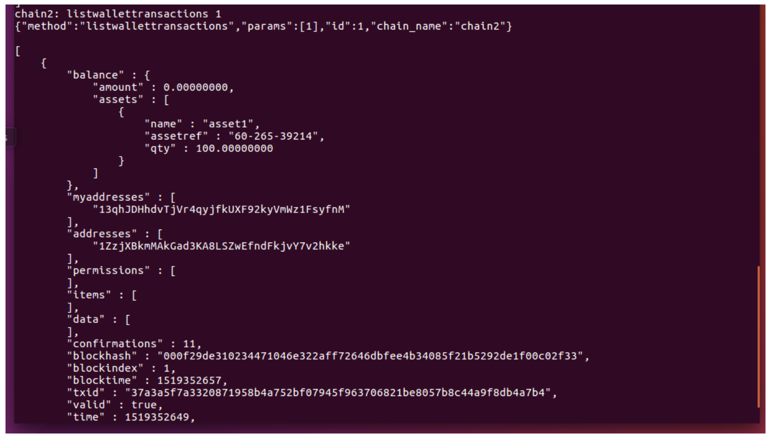
```
listwallettransactions 1
```

Server 1:



```
chain2: listwallettransactions 1
{"method":"listwallettransactions","params":[1],"id":1,"chain_name":"chain2"}

[
    {
        "balance" : {
            "amount" : 0.00000000,
            "assets" : [
                {
                    "name" : "asset1",
                    "assetref" : "60-265-39214",
                    "qty" : -100.00000000
                }
            ]
        },
        "myaddresses" : [
            "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke"
        ],
        "addresses" : [
            "13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM"
        ],
        "permissions" : [
        ],
        "items" : [
        ],
        "data" : [
        ],
        "confirmations" : 9,
        "blockhash" : "000f29de310234471046e322aff72646dbfee4b34085f21b5292de1f00c02f33",
        "blockindex" : 1,
        "blocktime" : 1519352657,
        "txid" : "37a3a5f7a3320871958b4a752bf07945f963706821be8057b8c44a9f8db4a7b4",
        "valid" : true,
        "time" : 1519352649,
        "timereceived" : 1519352649
    }
]
chain2: 
```

Server 2:



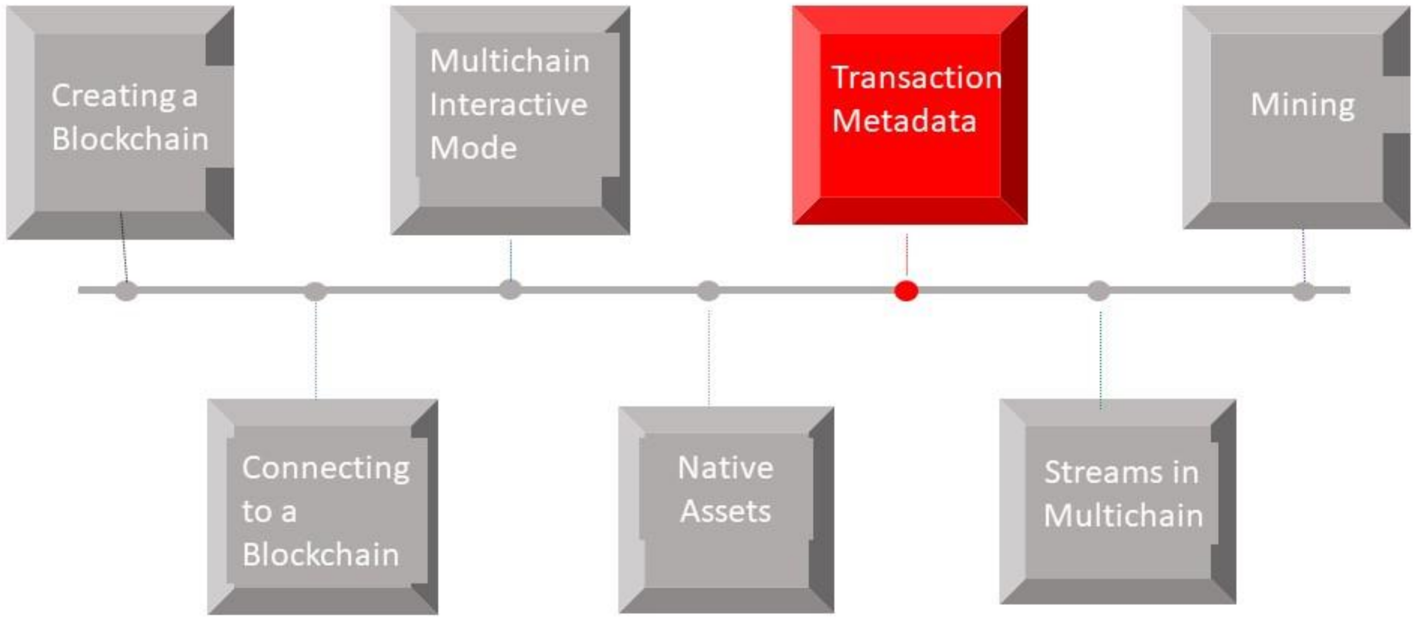
```
chain2: listwallettransactions 1
{"method":"listwallettransactions","params":[1],"id":1,"chain_name":"chain2"}

[
    {
        "balance" : {
            "amount" : 0.00000000,
            "assets" : [
                {
                    "name" : "asset1",
                    "assetref" : "60-265-39214",
                    "qty" : 100.00000000
                }
            ]
        },
        "myaddresses" : [
            "13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM"
        ],
        "addresses" : [
            "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke"
        ],
        "permissions" : [
        ],
        "items" : [
        ],
        "data" : [
        ],
        "confirmations" : 11,
        "blockhash" : "000f29de310234471046e322aff72646dbfee4b34085f21b5292de1f00c02f33",
        "blockindex" : 1,
        "blocktime" : 1519352657,
        "txid" : "37a3a5f7a3320871958b4a752bf07945f963706821be8057b8c44a9f8db4a7b4",
        "valid" : true,
        "time" : 1519352649,
```

Transaction Metadata

Transaction Metadata

“Let's learn about Transaction Metadata.”

Step 5: Transaction Metadata



Checking Asset balances

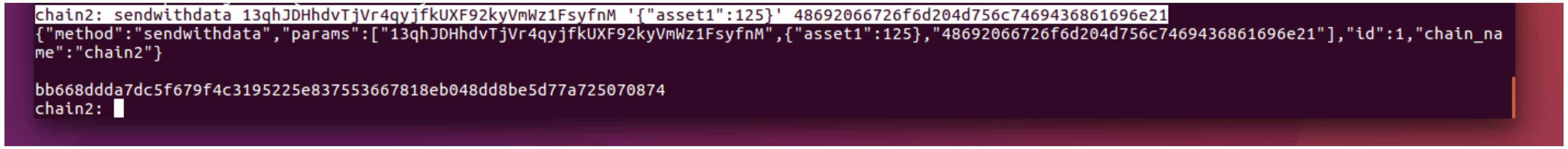
To create a transaction with some metadata, on first server, execute the following command

```
sendwithdata  13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM  '{"asset1":125}'
```

```
48692066726f6d204d756c7469436861696e21
```

$\underbrace{\hphantom{48692066726f6d204d756c7469436861696e21}}$

This is just a metadata and it can be anything of your choice



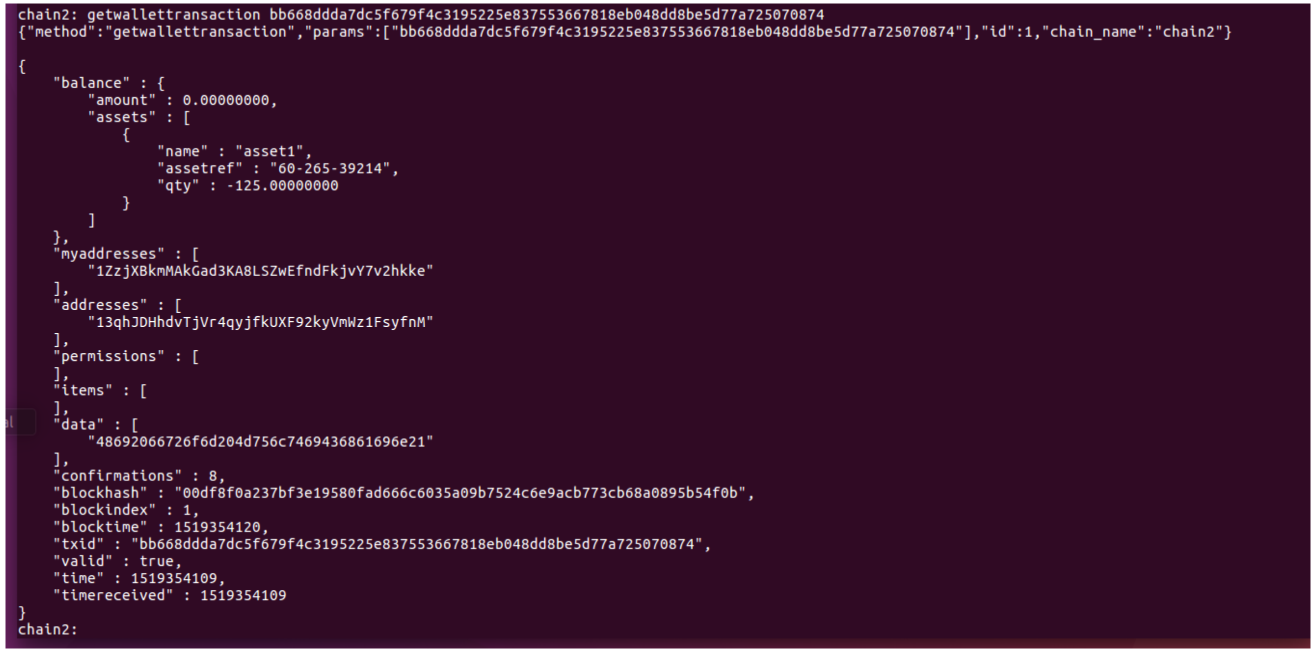
```
chain2: sendwithdata 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM '{"asset1":125}' 48692066726f6d204d756c7469436861696e21
{"method":"sendwithdata","params":["13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM",{"asset1":125},"48692066726f6d204d756c7469436861696e21"],"id":1,"chain_na
me":"chain2"}

bb668ddda7dc5f679f4c3195225e837553667818eb048dd8be5d77a725070874
chain2: 
```

Examining the transaction

Now the previous transaction can be examined on the second server
To examine the server, execute the following command:

```
getwallettransaction 6ƭƭƭ7eƭa.....66f67
```

This is the transaction ID



```
chain2: getwallettransaction bb668ddda7dc5f679f4c3195225e837553667818eb048dd8be5d77a725070874
{"method":"getwallettransaction","params":["bb668ddda7dc5f679f4c3195225e837553667818eb048dd8be5d77a725070874"],"id":1,"chain_name":"chain2"}

{
    "balance" : {
        "amount" : 0.00000000,
        "assets" : [
            {
                "name" : "asset1",
                "assetref" : "60-265-39214",
                "qty" : -125.00000000
            }
        ]
    },
    "myaddresses" : [
        "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke"
    ],
    "addresses" : [
        "13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM"
    ],
    "permissions" : [
    ],
    "items" : [
    ],
    "data" : [
        "48692066726f6d204d756c7469436861696e21"
    ],
    "confirmations" : 8,
    "blockhash" : "00df8f0a237bf3e19580fad666c6035a09b7524c6e9acb773cb68a0895b54f0b",
    "blockindex" : 1,
    "blocktime" : 1519354120,
    "txid" : "bb668ddda7dc5f679f4c3195225e837553667818eb048dd8be5d77a725070874",
    "valid" : true,
    "time" : 1519354109,
    "timereceived" : 1519354109
}
chain2:
```

In the **balance field** you can see the incoming 125 units of asset1 and the **data field** containing the hexadecimal metadata that was added

Streams

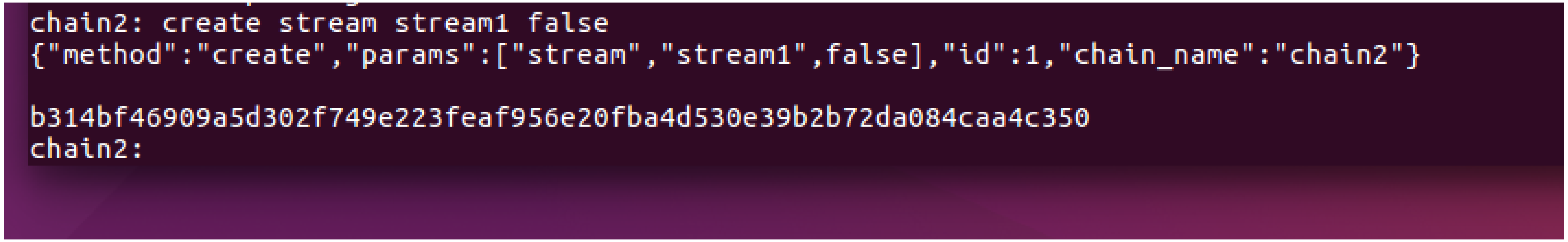
Streams

“Let's learn about streams.”

Examining the transaction

To create stream , execute the following command on the first server:

create stream stream1 false



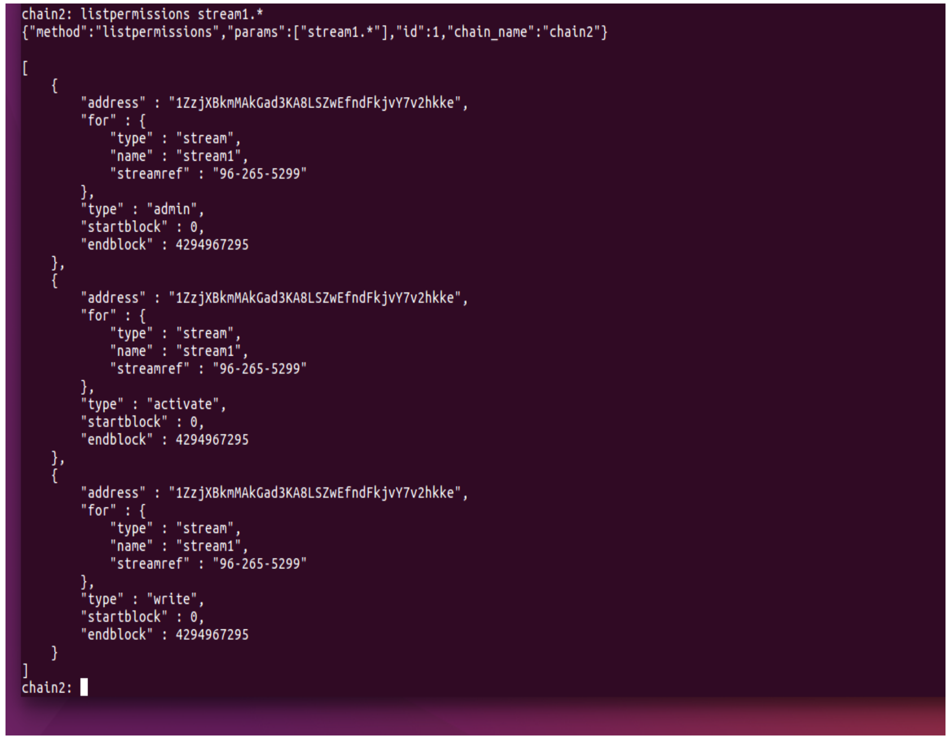
```
chain2: create stream stream1 false
{"method":"create","params":["stream","stream1",false],"id":1,"chain_name":"chain2"}

b314bf46909a5d302f749e223feaf956e20fba4d530e39b2b72da084caa4c350
chain2:
```

***Note:** The false means the stream can only be written to by those with explicit permissions*

Permissions of the Stream

Let’s see the permissions of the stream
To see the permissions, execute the following command:

```
listpermissions stream1.*
```



```
chain2: listpermissions stream1.*
{"method":"listpermissions","params":["stream1.*"],"id":1,"chain_name":"chain2"}

[
    {
        "address" : "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke",
        "for" : {
            "type" : "stream",
            "name" : "stream1",
            "streamref" : "96-265-5299"
        },
        "type" : "admin",
        "startblock" : 0,
        "endblock" : 4294967295
    },
    {
        "address" : "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke",
        "for" : {
            "type" : "stream",
            "name" : "stream1",
            "streamref" : "96-265-5299"
        },
        "type" : "activate",
        "startblock" : 0,
        "endblock" : 4294967295
    },
    {
        "address" : "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke",
        "for" : {
            "type" : "stream",
            "name" : "stream1",
            "streamref" : "96-265-5299"
        },
        "type" : "write",
        "startblock" : 0,
        "endblock" : 4294967295
    }
]
chain2: 
```

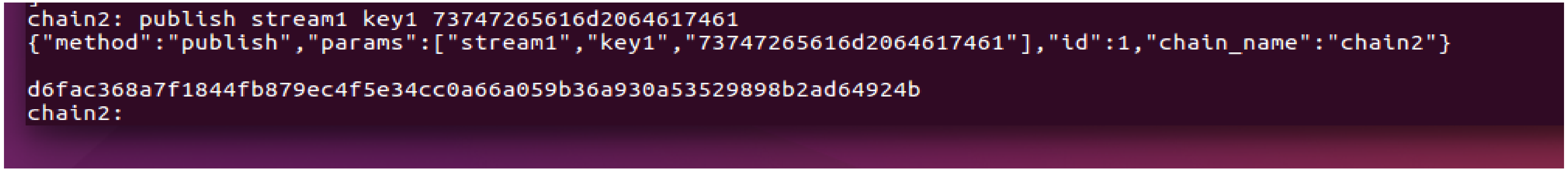
***Note:** So far now only first server has the ability to write to the stream, as well as administrate it*

Publishing to the stream

Let's publish something to the stream using the key
To publish to the stream with the key, execute the following:

```
publish stream1 key1 73747265616d2064617461
```

This is just a random data. It can be anything of your choice



```
chain2: publish stream1 key1 73747265616d2064617461
{"method":"publish","params":["stream1","key1","73747265616d2064617461"],"id":1,"chain_name":"chain2"}

d6fac368a7f1844fb879ec4f5e34cc0a66a059b36a930a53529898b2ad64924b
chain2:
```

***Note:** So far now only first server has the ability to write to the stream, as well as administrate it*

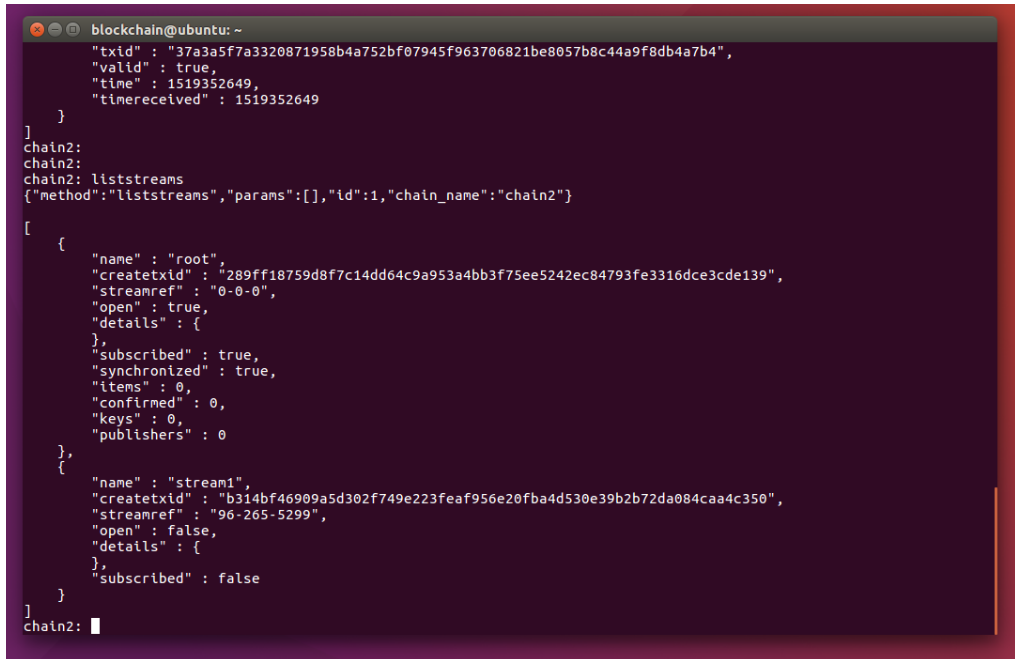
Checking the visibility of the second server

Now let’s see that the stream is visible on another node

On the second server execute the following command:

```
liststreams
```

Above command will list the streams



```
blockchain@ubuntu: ~
        "txid" : "37a3a5f7a3320871958b4a752bf07945f963706821be8057b8c44a9f8db4a7b4",
        "valid" : true,
        "time" : 1519352649,
        "timereceived" : 1519352649
    }
]
chain2:
chain2:
chain2: liststreams
{"method":"liststreams","params":[],"id":1,"chain_name":"chain2"}

[
    {
        "name" : "root",
        "createtxid" : "289ff18759d8f7c14dd64c9a953a4bb3f75ee5242ec84793fe3316dce3cde139",
        "streamref" : "0-0-0",
        "open" : true,
        "details" : {
        },
        "subscribed" : true,
        "synchronized" : true,
        "items" : 0,
        "confirmed" : 0,
        "keys" : 0,
        "publishers" : 0
    },
    {
        "name" : "stream1",
        "createtxid" : "b314bf46909a5d302f749e223feaf956e20fba4d530e39b2b72da084caa4c350",
        "streamref" : "96-265-5299",
        "open" : false,
        "details" : {
        },
        "subscribed" : false
    }
]
chain2: 
```

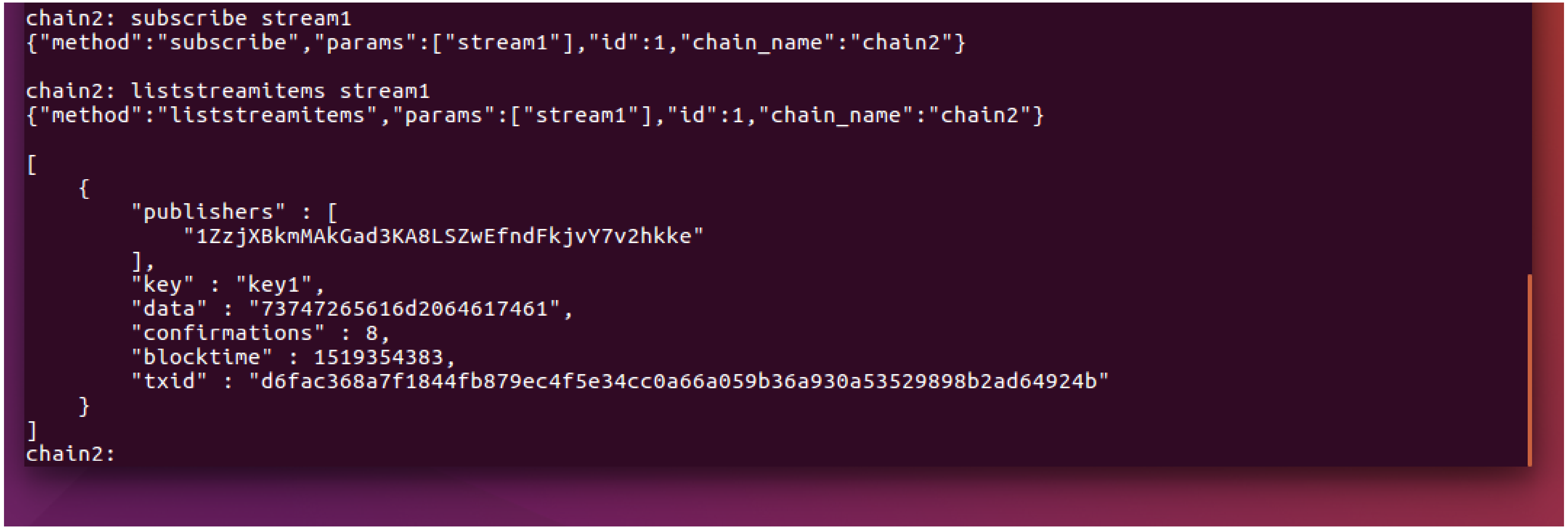
***Note:** Here, the root stream was in the system by default.*

Second server to subscribe to the stream

Now we want the second server to subscribe to the stream, then view its contents
To subscribe to the stream and view its content, execute the following command

```
subscribe stream1
```

```
liststreamitems stream1
```



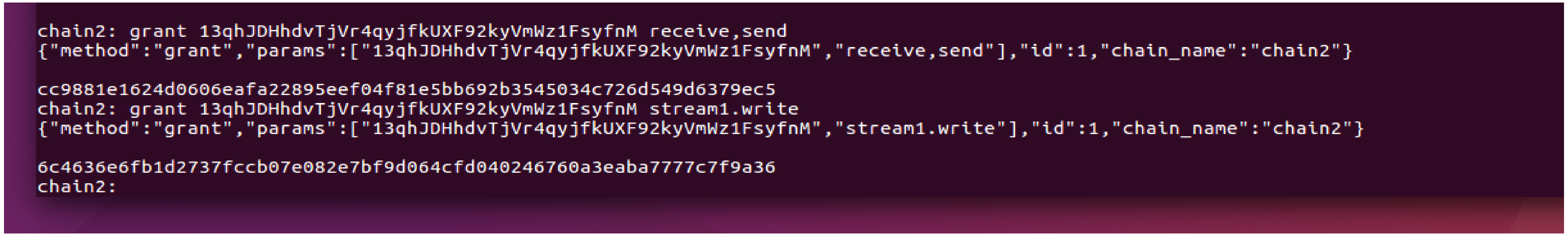
```
chain2: subscribe stream1
{"method":"subscribe","params":["stream1"],"id":1,"chain_name":"chain2"}

chain2: liststreamitems stream1
{"method":"liststreamitems","params":["stream1"],"id":1,"chain_name":"chain2"}

[
    {
        "publishers" : [
            "1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke"
        ],
        "key" : "key1",
        "data" : "73747265616d2064617461",
        "confirmations" : 8,
        "blocktime" : 1519354383,
        "txid" : "d6fac368a7f1844fb879ec4f5e34cc0a66a059b36a930a53529898b2ad64924b"
    }
]
chain2:
```

Allowing second server to publish to the stream

Now we want the second server to be allowed to publish to the stream
To publish to a stream, first we have to provide permission to the address, execute the following on the first server:

```
grant 13aNooypwXPW5NzuxvzGEDtEG6Hxtq8LhpByd receive,send
```

```
grant 13aNooypwXPW5NzuxvzGEDtEG6Hxtq8LhpByd stream1.write
```



```
chain2: grant 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM receive,send
{"method":"grant","params":["13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM","receive,send"],"id":1,"chain_name":"chain2"}

cc9881e1624d0606eafa22895eef04f81e5bb692b3545034c726d549d6379ec5
chain2: grant 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM stream1.write
{"method":"grant","params":["13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM","stream1.write"],"id":1,"chain_name":"chain2"}

6c4636e6fb1d2737fccb07e082e7bf9d064cfd040246760a3eaba7777c7f9a36
chain2:
```

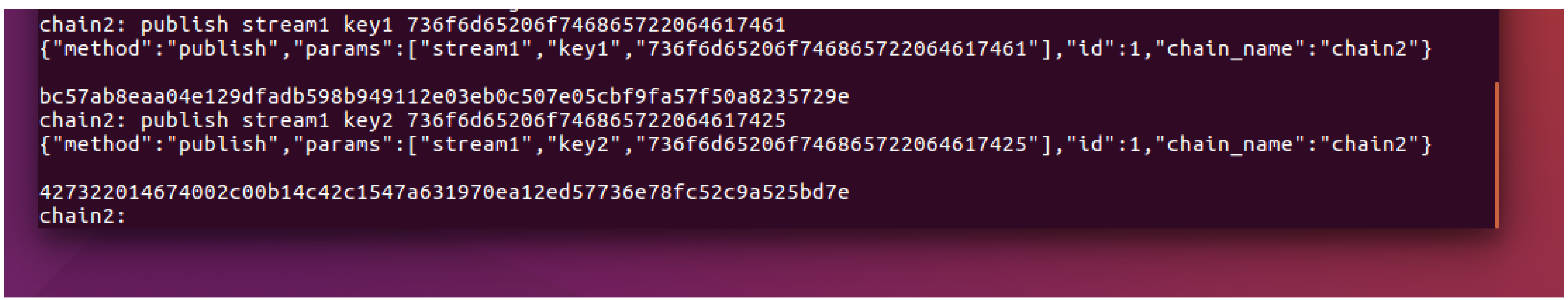
Note: *The address needs both general send/receive permissions for the blockchain, as well as permission to write to this specific stream*

Publishing items on the second server

Now let's publish a couple of items on the second server:

```
publish stream1 key1 736f6d65206f746865722064617461
```

```
publish stream1 key2 53747265616d732052756c6521
```



```
chain2: publish stream1 key1 736f6d65206f746865722064617461
{"method":"publish","params":["stream1","key1","736f6d65206f746865722064617461"],"id":1,"chain_name":"chain2"}

bc57ab8eaa04e129dfadb598b949112e03eb0c507e05cbf9fa57f50a8235729e
chain2: publish stream1 key2 736f6d65206f746865722064617425
{"method":"publish","params":["stream1","key2","736f6d65206f746865722064617425"],"id":1,"chain_name":"chain2"}

427322014674002c00b14c42c1547a631970ea12ed57736e78fc52c9a525bd7e
chain2:
```

Mining

Mining

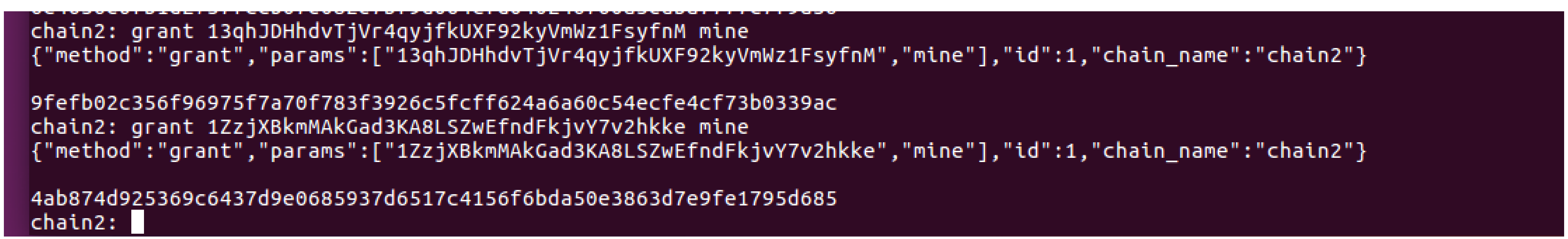
“Let’s learn more about mining.”

Start mining

To start the mining process, On the *first server* run the following:

```
grant 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM  mine
```

```
grant 1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke mine
```



```
chain2: grant 13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM mine
{"method":"grant","params":["13qhJDHhdvTjVr4qyjfkUXF92kyVmWz1FsyfnM","mine"],"id":1,"chain_name":"chain2"}

9fefb02c356f96975f7a70f783f3926c5fcff624a6a60c54ecfe4cf73b0339ac
chain2: grant 1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke mine
{"method":"grant","params":["1ZzjXBkmMAkGad3KA8LSZwEfndFkjvY7v2hkke","mine"],"id":1,"chain_name":"chain2"}

4ab874d925369c6437d9e0685937d6517c4156f6bda50e3863d7e9fe1795d685
chain2: 
```

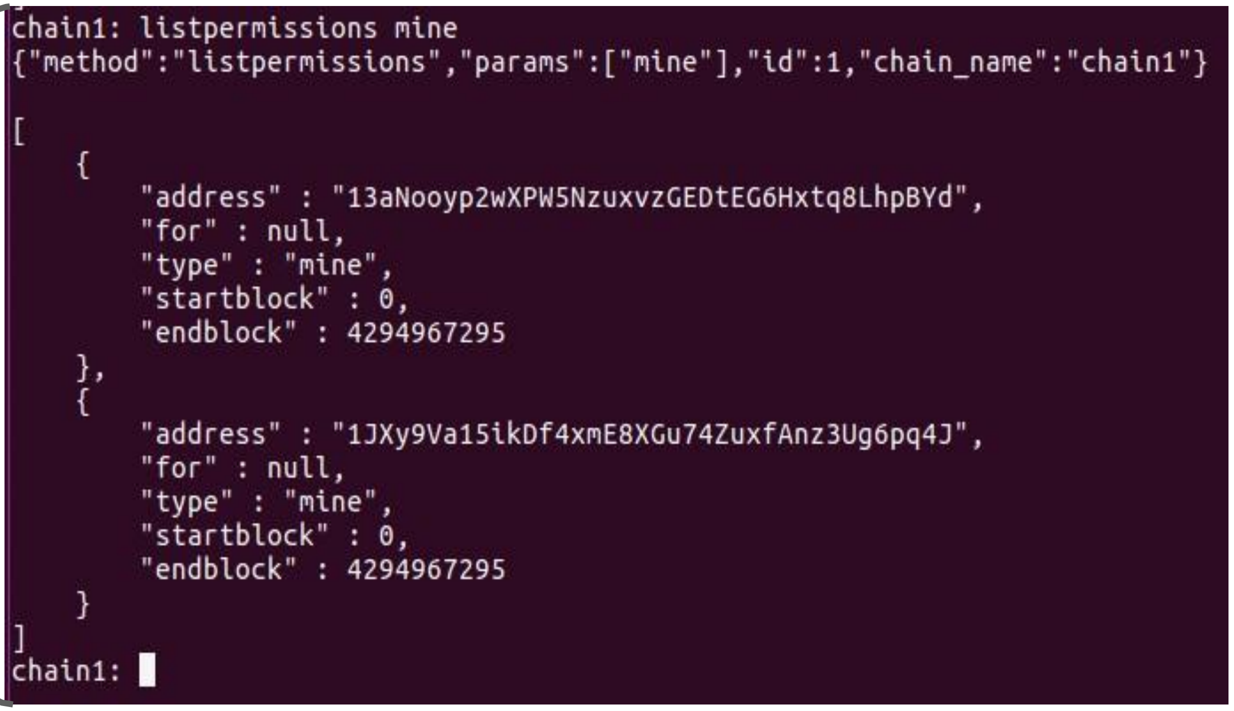
Note: *We have granted mining permission to both the nodes*

Check if the two permitted miners are listed

To check that two permitted miners are listed, execute the following command on the second

listpermissions mine

We can see that the mining consent is allowed to both the mineworkers

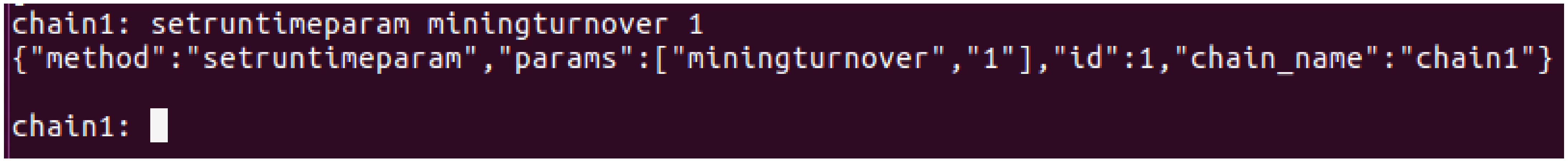


```
chain1: listpermissions mine
{"method":"listpermissions","params":["mine"],"id":1,"chain_name":"chain1"}

[
    {
        "address" : "13aNooyp2wXPW5NzuxvzGEDtEG6Hxtq8LhpBYd",
        "for" : null,
        "type" : "mine",
        "startblock" : 0,
        "endblock" : 4294967295
    },
    {
        "address" : "1JXy9Va15ikDf4xmE8XGu74ZuxfAnz3Ug6pq4J",
        "for" : null,
        "type" : "mine",
        "startblock" : 0,
        "endblock" : 4294967295
    }
]
chain1: 
```

Setting runtime parameters

To maximize the degree of miner randomness, execute the following command:

setruntimeparam miningturnover 1



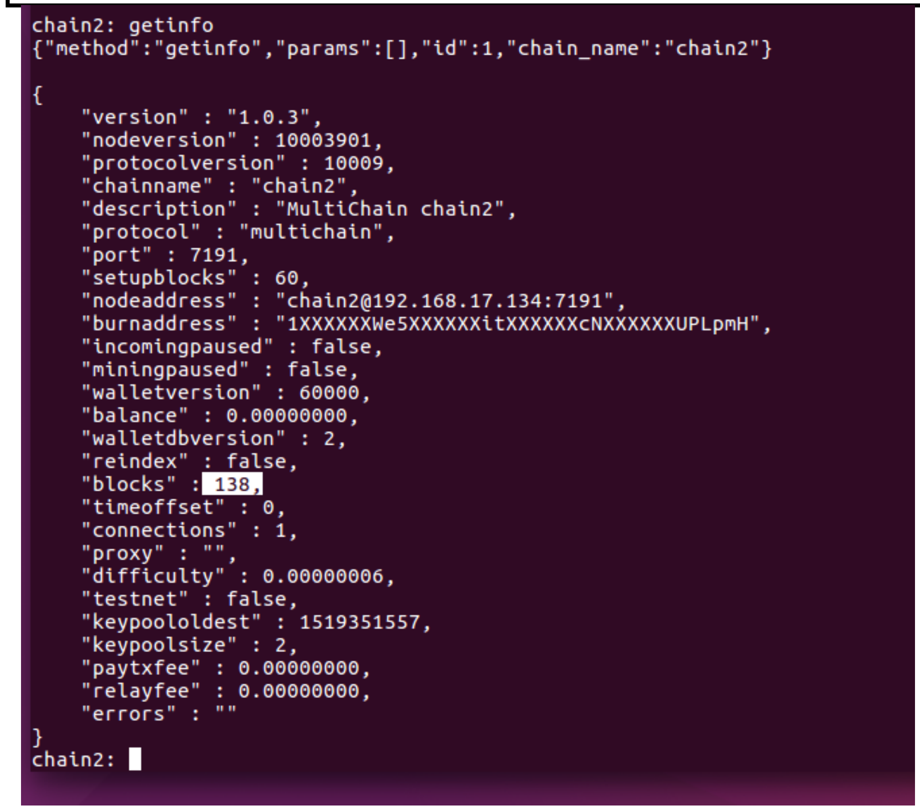
```
chain1: setruntimeparam miningturnover 1
{"method":"setruntimeparam","params":["miningturnover","1"],"id":1,"chain_name":"chain1"}

chain1: 
```

Note: *Presently sit tight for a few minutes, with the goal that a couple of blocks are mined*

Checking the current block height

To check the block height, on either server run the following command :

getinfo



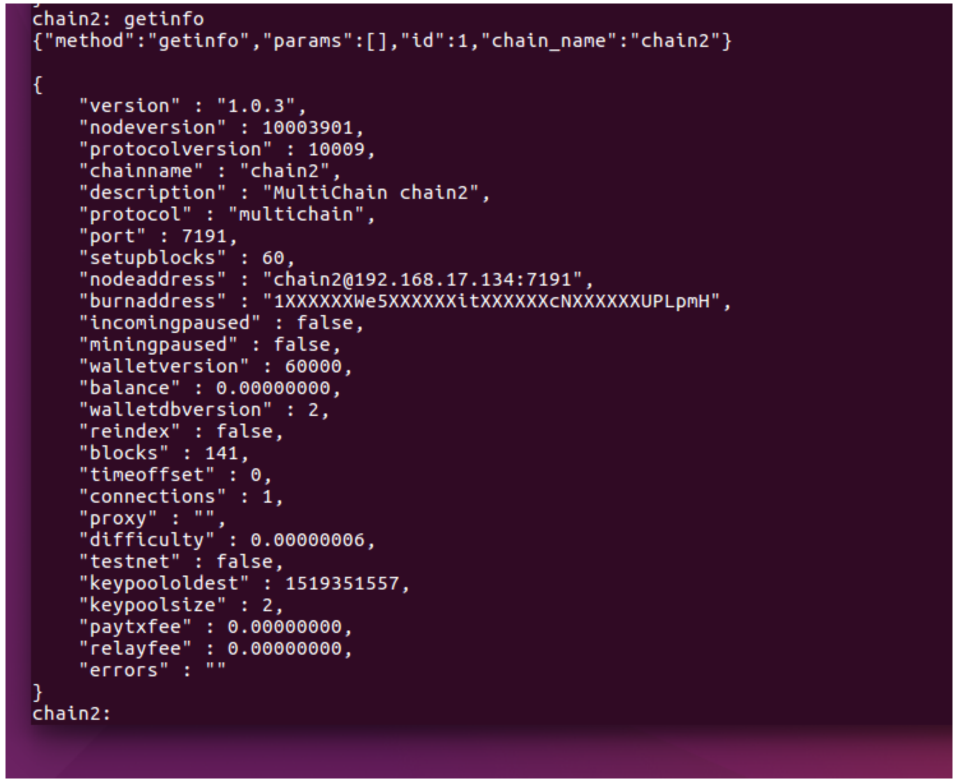
```
chain2: getinfo
{"method":"getinfo","params":[],"id":1,"chain_name":"chain2"}

{
    "version" : "1.0.3",
    "nodeversion" : 10003901,
    "protocolversion" : 10009,
    "chainname" : "chain2",
    "description" : "MultiChain chain2",
    "protocol" : "multichain",
    "port" : 7191,
    "setupblocks" : 60,
    "nodeaddress" : "chain2@192.168.17.134:7191",
    "burnaddress" : "1XXXXXXWe5XXXXXXitXXXXXXcNXXXXXXUPLpmH",
    "incomingpaused" : false,
    "miningpaused" : false,
    "walletversion" : 60000,
    "balance" : 0.00000000,
    "walletdbversion" : 2,
    "reindex" : false,
    "blocks" : 138,
    "timeoffset" : 0,
    "connections" : 1,
    "proxy" : "",
    "difficulty" : 0.00000006,
    "testnet" : false,
    "keypoololdest" : 1519351557,
    "keypoolsize" : 2,
    "paytxfee" : 0.00000000,
    "relayfee" : 0.00000000,
    "errors" : ""
}
chain2: 
```

***Note:** The block stature is in the blocks field of the reaction*

Getting information of last few blocks

Now let’s get information about the last few blocks, beginning with this (here, 197) one:

getblock [block-height]



```
chain2: getinfo
{"method":"getinfo","params":[],"id":1,"chain_name":"chain2"}

{
    "version" : "1.0.3",
    "nodeversion" : 10003901,
    "protocolversion" : 10009,
    "chainname" : "chain2",
    "description" : "MultiChain chain2",
    "protocol" : "multichain",
    "port" : 7191,
    "setupblocks" : 60,
    "nodeaddress" : "chain2@192.168.17.134:7191",
    "burnaddress" : "1XXXXXXWe5XXXXXXitXXXXXXcNXXXXXXUPLpmH",
    "incomingpaused" : false,
    "miningpaused" : false,
    "walletversion" : 60000,
    "balance" : 0.00000000,
    "walletdbversion" : 2,
    "reindex" : false,
    "blocks" : 141,
    "timeoffset" : 0,
    "connections" : 1,
    "proxy" : "",
    "difficulty" : 0.00000006,
    "testnet" : false,
    "keypoololdest" : 1519351557,
    "keypoolsize" : 2,
    "paytxfee" : 0.00000000,
    "relayfee" : 0.00000000,
    "errors" : ""
}
chain2:
```

The address of the digger of each block is in the mineworker field of the reaction. In various blocks you should see the two unique locations in this field

Thank You

Email us – support@intellipaat.com

Visit us - https://intellipaat.com