Benjamin Wilke - MSDS 7349 Data and Network Security

Exploring Blockchain with BigchainDB

Import the BigChainDB Python driver, set the root URL to one of the nodes in our network, and instantiate BigChainDB.

In [1]:
```
from bigchaindb_driver import BigchainDB
bdb_root_url = 'ec2-34-213-113-185.us-west-2.compute.amazonaws.com:9984'
bdb = BigchainDB(bdb_root_url)
```

Our First Digital Asset

This is the digital asset definition. The data in this structure can be freeform (anything you want it to be), but is immutable once added to the blockchain. In this example we will be transacting a bicycle. Let's give it some attributes.

In [2]:
```
bicycle = {
    'data':{
        'bicycle':{
            'serial_number': 'gfkdfgsfd9878967',
            'manufacturer': 'schwinn',
            'color': 'blue',
            'style': 'mountain',
            'manufacture_date': 'June 1, 2016'
        }
    }
}
```

Create Identities via Public/Private Keys

User identities are represented by public/private key pairs. The private key is used to sign transactions, meanwhile the public key is used to verify that a signed transaction was indeed signed by the one who claims to be the signee. This code creates 2 identities (Alice and Bob) and displays their newly generated public and private keys.

In [3]:
```
from bigchaindb_driver.crypto import generate_keypair

alice, bob = generate_keypair(), generate_keypair()

print('Alice Public Key:', alice.public_key)
print('Alice Private Key:', alice.private_key)
print('------------------------------------------------------------------
-----')
print('Bob Public Key:', bob.public_key)
print('Bob Private Key:', bob.private_key)
```

```
Alice Public Key: B2Rvrz6J2JqpT68AcRPryVpRVzYUubAqasEUwtCDPjX6
Alice Private Key: 9NdQQrsPESgL7N1h5KPuZdW3hc2nGxR5Uu5NqbUsnf32
----------------------------------------------------------------------
Bob Public Key: A1PLkGLZp3Qea3XdzjJ4RDUX8WXhsrfQuCYWGjXFzz1b
Bob Private Key: LU4MbqWQgdderEdsVN29HnDDdKw8t7jyr8PmKUzaXdL
```

Asset Creation

The first transaction will be a CREATE operation to designate that the bike belongs to Alice. Let's create the digital asset (note the operation).

In [4]:
```
prepared_creation_tx = bdb.transactions.prepare(
    operation='CREATE',
    signers=alice.public_key,
    asset=bicycle
)
```

Let's take a look at what's inside our digital asset. You'll find our bicycle added to the asset as well as Alice's public key in both the 'inputs', 'owners_before', and the 'outputs' section (designating that she is marking herself as the owner).

In [5]:
```
prepared_creation_tx
```

Out[5]:
```
{'asset': {'data': {'bicycle': {'color': 'blue',
    'manufacture_date': 'June 1, 2016',
    'manufacturer': 'schwinn',
    'serial_number': 'gfkdfgsfd9878967',
    'style': 'mountain'}}},
 'id': None,
 'inputs': [{'fulfillment': {'public_key': 'B2Rvrz6J2JqpT68AcRPryVpRVzY
UubAqasEUwtCDPjX6',
    'type': 'ed25519-sha-256'},
   'fulfills': None,
   'owners_before': ['B2Rvrz6J2JqpT68AcRPryVpRVzYUubAqasEUwtCDPjX6']}],
 'metadata': None,
 'operation': 'CREATE',
 'outputs': [{'amount': '1',
   'condition': {'details': {'public_key': 'B2Rvrz6J2JqpT68AcRPryVpRVzY
UubAqasEUwtCDPjX6',
     'type': 'ed25519-sha-256'},
    'uri': 'ni:///sha-256;hbt48qFfQ-ZnqZ_MukxBGo6XeAL8fhxykyFAQGP5lJU?f
pt=ed25519-sha-256&cost=131072'},
   'public_keys': ['B2Rvrz6J2JqpT68AcRPryVpRVzYUubAqasEUwtCDPjX6']}],
 'version': '2.0'}
```

In order to be fulfilled the CREATE transaction must now be signed with Alice's private key.

In [6]:
```
fulfilled_creation_tx = bdb.transactions.fulfill(
    prepared_creation_tx,
    private_keys=alice.private_key
)
```

Let's also take a look at what's inside fullfilled transaction ready to be sent to BigchainDB.

```
In [7]: fulfilled_creation_tx
```

```
Out[7]: {'asset': {'data': {'bicycle': {'color': 'blue',
             'manufacture_date': 'June 1, 2016',
             'manufacturer': 'schwinn',
             'serial_number': 'gfkdfgsfd9878967',
             'style': 'mountain'}}},
         'id': '152d1710d16dee734081c0de0b2858d550a4c701ad83f04262db697977ded76
        a',
         'inputs': [{'fulfillment': 'pGSAIJTyoxG1ESpR4n6ML4X_0Z2oHwPIQqFzqUQ5zH
        zrQSuJgUBMNsFuri29zIvmaOFZWR0jXx5KILunFqk9NMD0XpUn8yxHUIrU9utcLlvDkYm0z
        OcpqwN5IBtsE2EZN8GzISUB',
           'fulfills': None,
           'owners_before': ['B2Rvrz6J2JqpT68AcRPryVpRVzYUubAqasEUwtCDPjX6']}],
         'metadata': None,
         'operation': 'CREATE',
         'outputs': [{'amount': '1',
           'condition': {'details': {'public_key': 'B2Rvrz6J2JqpT68AcRPryVpRVzY
        UubAqasEUwtCDPjX6',
             'type': 'ed25519-sha-256'},
            'uri': 'ni:///sha-256;hbt48qFfQ-ZnqZ_MukxBGo6XeAL8fhxykyFAQGP5lJU?f
        pt=ed25519-sha-256&cost=131072'},
           'public_keys': ['B2Rvrz6J2JqpT68AcRPryVpRVzYUubAqasEUwtCDPjX6']}],
         'version': '2.0'}
```

Next - we send the transaction over to a BigchainDB node in our network.

```
In [8]: sent_creation_tx = bdb.transactions.send_commit(fulfilled_creation_tx)
```

We can observe that the response from the node is the same as what was sent.

```
In [9]: sent_creation_tx == fulfilled_creation_tx
```

```
Out[9]: True
```

Let's now check to see if the transaction was included in the block. In order to do this we will look up the transaction by ID from the fulfillment dictionary (shown below). We'll also save this ID so we can transfer Alice's asset later.

```
In [10]: txid = fulfilled_creation_tx['id']
         txid
```

```
Out[10]: '152d1710d16dee734081c0de0b2858d550a4c701ad83f04262db697977ded76a'
```

NOTE: The block_height here doesn't start at 0 because of previous testing, afterall blockchain is immutable!

```
In [11]: block_height = bdb.blocks.get(txid=fulfilled_creation_tx['id'])
         block_height
```

```
Out[11]: 8
```

If we want to see the whole block, we can retrieve the block itself (using the block height). Since the blockchain height increases monotonically the height of block can be regarded as its identifier if it was the last block added.

```
In [12]: block = bdb.blocks.retrieve(str(block_height))
         block
```

```
Out[12]: {'height': 8,
          'transactions': [{'asset': {'data': {'bicycle': {'color': 'blue',
                 'manufacture_date': 'June 1, 2016',
                 'manufacturer': 'schwinn',
                 'serial_number': 'gfkdfgsfd9878967',
                 'style': 'mountain'}}},
             'id': '152d1710d16dee734081c0de0b2858d550a4c701ad83f04262db697977ded
         76a',
             'inputs': [{'fulfillment': 'pGSAIJTyoxG1ESpR4n6ML4X_0Z2oHwPIQqFzqUQ5
         zHzrQSuJgUBMNsFuri29zIvmaOFZWR0jXx5KILunFqk9NMD0XpUn8yxHUIrU9utcLlvDkYm
         0zOcpqwN5IBtsE2EZN8GzISUB',
               'fulfills': None,
               'owners_before': ['B2Rvrz6J2JqpT68AcRPryVpRVzYUubAqasEUwtCDPjX
         6']}],
             'metadata': None,
             'operation': 'CREATE',
             'outputs': [{'amount': '1',
               'condition': {'details': {'public_key': 'B2Rvrz6J2JqpT68AcRPryVpRV
         zYUubAqasEUwtCDPjX6',
                 'type': 'ed25519-sha-256'},
                'uri': 'ni:///sha-256;hbt48qFfQ-ZnqZ_MukxBGo6XeAL8fhxykyFAQGP5lJ
         U?fpt=ed25519-sha-256&cost=131072'},
               'public_keys': ['B2Rvrz6J2JqpT68AcRPryVpRVzYUubAqasEUwtCDPjX6']}],
             'version': '2.0'}]}
```

Now let's perform another CREATE transaction to see how the blockchain behaves. In this example we will assign the ownership of bike owned by Bob. We'll combine some of the steps here since we've outlined the procedure earlier.

```
In [13]: bicycle2 = {
             'data':{
                 'bicycle':{
                     'serial_number': 'nbghR32gCQsd1234',
                     'manufacturer': 'mongoose',
                     'color': 'black',
                     'style': 'bmx',
                     'manufacture_date': 'February 1, 2019'
                 }
             }
         }

         prepared_creation_tx1 = bdb.transactions.prepare(
             operation='CREATE',
             signers=bob.public_key,
             asset=bicycle2
         )

         fulfilled_creation_tx1 = bdb.transactions.fulfill(
             prepared_creation_tx1,
             private_keys=bob.private_key
         )

         sent_creation_tx1 = bdb.transactions.send_commit(fulfilled_creation_tx1)
```

Let's check out the new block height.

```
In [14]: block_height1 = bdb.blocks.get(txid=fulfilled_creation_tx1['id'])
         block_height1

Out[14]: 10
```

And finally take a peek at the block...

```
In [16]: block1 = bdb.blocks.retrieve(str(block_height1))
         block1
```

```
Out[16]: {'height': 10,
          'transactions': [{'asset': {'data': {'bicycle': {'color': 'black',
               'manufacture_date': 'February 1, 2019',
               'manufacturer': 'mongoose',
               'serial_number': 'nbghR32gCQsd1234',
               'style': 'bmx'}}},
            'id': 'bb6530a2e28d77feb8b737d850dc7a20bd01d069665e3b0a5171126f8299c
         891',
            'inputs': [{'fulfillment': 'pGSAIIXScVjPJ6Lxfy9Z0Iew0_zW0md9zljfMJeN
         Lse7oUKugUC2YsXZrov3QS3B3uPBX0_v25iKa-A3YB7DHPCtSF7Sa-8WsG9mkrFYivegnTO
         ln0_yIaWR4V1oenmJBKX0Zb0D',
              'fulfills': None,
              'owners_before': ['A1PLkGLZp3Qea3XdzjJ4RDUX8WXhsrfQuCYWGjXFzz1
         b']}],
            'metadata': None,
            'operation': 'CREATE',
            'outputs': [{'amount': '1',
              'condition': {'details': {'public_key': 'A1PLkGLZp3Qea3XdzjJ4RDUX8
         WXhsrfQuCYWGjXFzz1b',
                'type': 'ed25519-sha-256'},
               'uri': 'ni:///sha-256;9RCWWPOVZSwgsNYuaVph7I4Kh8PJZfCnCJ6t4ye_KX
         k?fpt=ed25519-sha-256&cost=131072'},
              'public_keys': ['A1PLkGLZp3Qea3XdzjJ4RDUX8WXhsrfQuCYWGjXFzz1b']}],
            'version': '2.0'}]}
```

You'll find in this output that Bob is indeed the owner of his black Mongoose BMX bicycle based on the presence of his public key in the 'outputs' section of the transtion.

Asset Transfer

Now let's transfer Alice's bicycle to Bob.

In order to do this Alice must consume the transaction in which the bicycle asset was created in order to have the correct information to make the transfer. That is - we need a reference of "what" to actually transfer.

We can achieve this by looking up the create transaction via the transaction ID we stored earlier (there are other ways to do this by querying the blockchain).

```
In [17]: creation_tx = bdb.transactions.retrieve(txid)
```

Let's take a look inside and verify it's Alice's asset creation from earlier.

```
In [18]: creation_tx
```

```
Out[18]: {'asset': {'data': {'bicycle': {'color': 'blue',
             'manufacture_date': 'June 1, 2016',
             'manufacturer': 'schwinn',
             'serial_number': 'gfkdfgsfd9878967',
             'style': 'mountain'}}},
          'id': '152d1710d16dee734081c0de0b2858d550a4c701ad83f04262db697977ded76
         a',
          'inputs': [{'fulfillment': 'pGSAIJTyoxG1ESpR4n6ML4X_0Z2oHwPIQqFzqUQ5zH
         zrQSuJgUBMNsFuri29zIvmaOFZWR0jXx5KILunFqk9NMD0XpUn8yxHUIrU9utcLlvDkYm0z
         OcpqwN5IBtsE2EZN8GzISUB',
            'fulfills': None,
            'owners_before': ['B2Rvrz6J2JqpT68AcRPryVpRVzYUubAqasEUwtCDPjX6']}],
          'metadata': None,
          'operation': 'CREATE',
          'outputs': [{'amount': '1',
            'condition': {'details': {'public_key': 'B2Rvrz6J2JqpT68AcRPryVpRVzY
         UubAqasEUwtCDPjX6',
              'type': 'ed25519-sha-256'},
             'uri': 'ni:///sha-256;hbt48qFfQ-ZnqZ_MukxBGo6XeAL8fhxykyFAQGP5lJU?f
         pt=ed25519-sha-256&cost=131072'},
            'public_keys': ['B2Rvrz6J2JqpT68AcRPryVpRVzYUubAqasEUwtCDPjX6']}],
          'version': '2.0'}
```

Sure enough - Alice is the owner as noted in the 'outputs' section of the dictionary above.

Let's prepare the transfer transaction. We first get the asset ID to determine what it is we're going to transfer (this is the ID from the create transaction). Next we prepare the transfer input dictionary.

```
In [19]: asset_id = creation_tx['id']
         transfer_asset = {
             'id': asset_id
         }
```

```
In [20]: output_index = 0

         output = creation_tx['outputs'][output_index]

         transfer_input = {
             'fulfillment': output['condition']['details'],
             'fulfills': {
                  'output_index': output_index,
                  'transaction_id': creation_tx['id'],
              },
              'owners_before': output['public_keys'],
         }

         prepared_transfer_tx = bdb.transactions.prepare(
             operation='TRANSFER',
             asset=transfer_asset,
             inputs=transfer_input,
             recipients=bob.public_key,
         )
```

Let's take a look at the prepared transfer. You will find the asset ID of Alice's bicycle and you will also notice that the current owner is Alice based on her public key in the 'inputs' and that Bob's public key has been added to the 'outputs', designating him as the desired new owner. Take a look at how this is done by inspecting how Alice's public key is added to transfer_input dictionary and Bob's public is added under recipients in the transactions.prepare method above.

```
In [21]: prepared_transfer_tx

Out[21]: {'asset': {'id': '152d1710d16dee734081c0de0b2858d550a4c701ad83f04262db6
         97977ded76a'},
          'id': None,
          'inputs': [{'fulfillment': {'public_key': 'B2Rvrz6J2JqpT68AcRPryVpRVzY
         UubAqasEUwtCDPjX6',
              'type': 'ed25519-sha-256'},
            'fulfills': {'output_index': 0,
              'transaction_id': '152d1710d16dee734081c0de0b2858d550a4c701ad83f042
         62db697977ded76a'},
            'owners_before': ['B2Rvrz6J2JqpT68AcRPryVpRVzYUubAqasEUwtCDPjX6']}],
          'metadata': None,
          'operation': 'TRANSFER',
          'outputs': [{'amount': '1',
            'condition': {'details': {'public_key': 'A1PLkGLZp3Qea3XdzjJ4RDUX8WX
         hsrfQuCYWGjXFzz1b',
               'type': 'ed25519-sha-256'},
              'uri': 'ni:///sha-256;9RCWWPOVZSwgsNYuaVph7I4Kh8PJZfCnCJ6t4ye_KXk?f
         pt=ed25519-sha-256&cost=131072'},
            'public_keys': ['A1PLkGLZp3Qea3XdzjJ4RDUX8WXhsrfQuCYWGjXFzz1b']}],
          'version': '2.0'}
```

Next let's sign the transaction as Alice using her private key and send it to the connected BigchainDB node. We will then verify that the BigchainDB response is the same as what was sent.

```
In [22]: fulfilled_transfer_tx = bdb.transactions.fulfill(
             prepared_transfer_tx,
             private_keys=alice.private_key,
         )

         sent_transfer_tx = bdb.transactions.send_commit(fulfilled_transfer_tx)
         sent_transfer_tx == fulfilled_transfer_tx

Out[22]: True
```

Finally - let's examine the current owner and previous owner of the transaction from the response from BigchainDB (which is as we've already tested, the same as our fulfill transfer dictionary).

```
In [23]: print("Is Bob the owner?",
             sent_transfer_tx['outputs'][0]['public_keys'][0] == bob.public_key)

         print("Was Alice the previous owner?",
             fulfilled_transfer_tx['inputs'][0]['owners_before'][0] == alice.publ
         ic_key)

         Is Bob the owner? True
         Was Alice the previous owner? True
```

Double Spends

One of the key benefits of blockchain technology and BigchainDB is the prevention of double spending - or transfering the same digital asset two or more times.

Let's create another another transfer transaction attempting to have Alice re-send the bike we just transferred to Bob back to her "secret" account.

Let's create Alice's "secret" account first.

```
In [26]: alice_secret_stash = generate_keypair()
```

Next let's create another transfer transaction with the same input (the bike we transferred to Bob).

```
In [27]: tx_transfer_2 = bdb.transactions.prepare(
             operation='TRANSFER',
             inputs=transfer_input,
             asset=transfer_asset,
             recipients=alice_secret_stash.public_key,
         )

         fulfilled_tx_transfer_2 = bdb.transactions.fulfill(
             tx_transfer_2,
             private_keys=alice.private_key,
         )
```

Now let's try and send our transfer transaction to our BigchainDB node (after importing the exception library so that we get a friendly exception).

```
In [28]: from bigchaindb_driver.exceptions import BigchaindbException
         try:
             bdb.transactions.send_commit(fulfilled_tx_transfer_2)
         except BigchaindbException as e:
             print(e.info)
```

```
{'message': 'Invalid transaction (DoubleSpend): input `152d1710d16dee73
4081c0de0b2858d550a4c701ad83f04262db697977ded76a` was already spent',
'status': 400}
```

You'll see that this transfer was not allowed as the bike already belongs to Bob.

Querying the Blockchain for Assets

As another demonstration we will show now how the blockchain can be queried via string search. Let's look up Alice's blue bike and Bob's black bike. You'll find multiple entries based on the series of testing we've done for this project.

```
In [25]: bdb.assets.get(search='blue')
```

```
Out[25]: [{'data': {'bicycle': {'color': 'blue',
             'manufacture_date': 'June 1, 2016',
             'manufacturer': 'schwinn',
             'serial_number': 'gfkdfgsfd9878967',
             'style': 'mountain'}},
           'id': 'bcda0cac29144845324a108f3061c4fb0346108eb0ed835f1112eb85b7c762
         c4'},
          {'data': {'bicycle': {'color': 'blue',
             'manufacture_date': 'June 1, 2016',
             'manufacturer': 'schwinn',
             'serial_number': 'gfkdfgsfd9878967',
             'style': 'mountain'}},
           'id': '152d1710d16dee734081c0de0b2858d550a4c701ad83f04262db697977ded7
         6a'}]
```

```
In [24]: bdb.assets.get(search='black')
```

```
Out[24]: [{'data': {'bicycle': {'color': 'black',
             'manufacture_date': 'February 1, 2019',
             'manufacturer': 'mongoose',
             'serial_number': 'nbghR32gCQsd1234',
             'style': 'bmx'}},
           'id': '71052a486dfe10ccde3461dccc0748d9cf18a6c5c884106dc720f78e5807ce
         49'},
          {'data': {'bicycle': {'color': 'black',
             'manufacture_date': 'February 1, 2019',
             'manufacturer': 'mongoose',
             'serial_number': 'nbghR32gCQsd1234',
             'style': 'bmx'}},
           'id': 'bb6530a2e28d77feb8b737d850dc7a20bd01d069665e3b0a5171126f8299c8
         91'}]
```