**tsmatz**

PROFESSIONAL DEVELOPMENT, DATA SCIENCE

MONDAY, JUNE 26TH, 2023

# Run ION (Sidetree), Decentralized Identifier (DID) Network

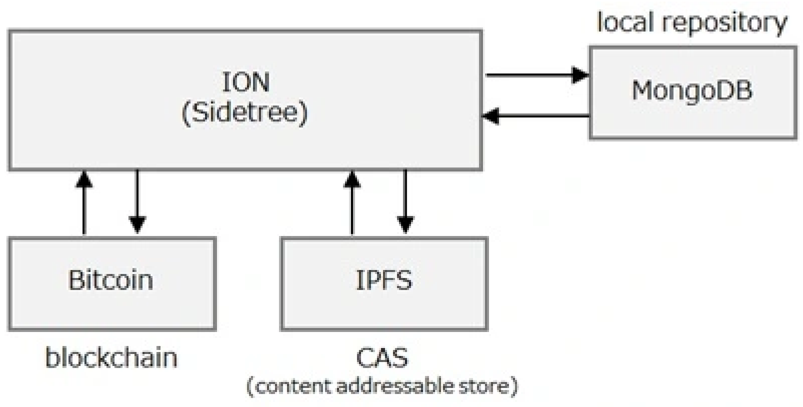
BY TSUYOSHI MATSUZAKI ON 2020-09-01 • ( 1 COMMENT )

ION is built on the top of Sidetree for decentralized identifier (DID) network.
Sidetree is a blockchain-agnostic protocol and has multiple reference implementations for a variety of blockchains, such as, Bitcoin, Ethereum, etc.
ION is one of such implementations, which runs on Bitcoin blockchain.

In this post, I show you how Sidetree DID network runs on decentralized technology stacks through ION implementation.

## Environment Settings

In this post, I installed ION on Ubuntu 18.04 in Microsoft Azure.
Before installing ION, first we should set up the underlying components – MongoDB and Bitcoin.



(https://tsmatz.files.wordpress.com/2020/09/20200904_ion_architecture.jpg)

Here I show you typical installation steps, but see ION installation guide (https://github.com/decentralized-identity/ion/blob/master/install-guide.md) on Github for details about ION installation.

## 1. Prerequisite Settings

The underlying blockchain, Bitcoin, will consume a large amount of disk for storing blocks. Before starting set-up, please make sure to expand disk.
In this post, we use Bitcoin testnet and it will consume around up-to 1 TB for blocks.
See here (https://netweblog.wordpress.com/2020/08/25/bitcoin-testnet-tutorial-for-developers/) for the steps to expand disk in Azure Virtual Machine.

Next install languages and dev tools for building ION.
ION is implemented by Node.js.

```
# install Node v10
sudo snap install node --classic --channel=10

# see whether node is correctly installed
node --version

# install build essential, including such as g++, etc
sudo apt install build-essential
```

Finally, the underlying IPFS (content addressable store) will use inbound port 4002 and 4003.
Open these ports in firewall settings.

## 2. Install and Set up Bitcoin

ION runs on Bitcoin protocol. For development purpose, here we set up Bitcoin testnet in virtual machine.

```
# download Bitcoin Core
# see https://bitcoincore.org/en/releases/
# (here I used Bitcoin Core 0.20.1.)
wget https://bitcoincore.org/bin/bitcoin-core-0.20.1/bitco

# extract tar
tar xvzf bitcoin-0.20.1-x86_64-linux-gnu.tar.gz

cd bitcoin-0.20.1
```

Now let’s start Bitcoin Core process.
For the first time to start, it will synchronize and verify all blocks in a chain, and generate transaction index (txindex) in mempool. Please wait patiently till transaction index will be generated. (As I show you later, the index is also

needed for synchronizing Sidetree operations.)
It will take 2 – 3 hours, and it will be much more when it's Bitcoin mainnet.

```
# start bitcoin core (testnet)
./bin/bitcoind -testnet \
  -rpcuser=myrpc01 \
  -rpcpassword=password01 \
  -fallbackfee=0.0002 \
  -txindex=1 \
  -server
```

> Note : The option "`-txindex=1`" is required in ION.

After you have run bitcoin core successfully, create a new wallet's address and get a private key for this generated address. (This private key will be needed for configuring ION later.)

```
# generate new address
./bin/bitcoin-cli -testnet \
  -rpcuser=myrpc01 \
  -rpcpassword=password01 \
  getnewaddress
```

```
tb1qwk2evuhspza74q6wc7l0zs3vln2pdykdk846qa
```

```
# dump private key
./bin/bitcoin-cli -testnet \
  -rpcuser=myrpc01 \
  -rpcpassword=password01 \
  dumpprivkey tb1qwk2evuhspza74q6wc7l0zs3vln2pdykdk846qa
```

```
cRAkycGAHeXCy2d9ezWK...
```

Here I don't go so far about Bitcoin itself, but please see here (https://netweblog.wordpress.com/2020/08/25/bitcoin-testnet-tutorial-for-developers/) for other commands in Bitcoin CLI.

## 3. Install and Set up MongoDB

In ION (Sidetree), MongoDB is used as local repository (operation store, transaction store, etc) by default.
Then, install and run MongoDB as follows.

```
cd ~

# import public key for mongodb
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.as

# create /etc/apt/sources.list.d/mongodb-org-4.4.list
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt

# reload local package
sudo apt-get update

# install mongodb (latest)
sudo apt-get install -y mongodb-org

# start mongodb daemon
sudo systemctl start mongod

# see whether mongodb is correctly started
sudo systemctl status mongod

# run as follows, when you stop or restart mongo
# sudo systemctl stop mongod
# sudo systemctl restart mongod
```

## 4. Configure and Build ION (Sidetree)

Now we can proceed to set up ION.
First, please download ION from Github.

```
# clone ion
git clone https://github.com/decentralized-identity/ion
cd ion
```

Next, configure ION to meet our Bitcoin and MongoDB settings.
Open `json/testnet-bitcoin-config.json` (configuration for sidetree bitcoin service) and edit the following values.

In this post, I use the default port – Bitcoin testnet 18332 and MongoDB 27017 – and then these settings are not needed to be changed.

- `bitcoinDataDirectory`
  Bitcoin data location (`datadir`). In my case, `/home/tsmatsuz/.bitcoin/testnet3`.
  Do not use "~" for specifying your home directory.
- `bitcoinPeerUri`
  The peer Uri of running Bitcoin Core. In my case, I'm using the default port 18332 for Bitcoin testnet.
- `bitcoinWalletOrImportString`
  Set a private key for Bitcoin address, which is extracted in above (`cRAkycGAHeXCy2d9ezWK...`).
- `mongoDbConnectionString`
  MongoDB connection string (Uri). In my case, I'm using the default port 27017.

I show you my entire configuration (`json/testnet-bitcoin-config.json`) as follows. Please replace `bitcoinDataDirectory` and `bitcoinWalletOrImportString` with your own settings.

```
{
  "bitcoinDataDirectory": "/home/tsmatsuz/.bitcoin/testnet3
  "bitcoinFeeSpendingCutoffPeriodInBlocks": 1,
  "bitcoinFeeSpendingCutoff": 0.001,
  "bitcoinPeerUri": "http://localhost:18332",
  "bitcoinRpcUsername": "myrpc01",
  "bitcoinRpcPassword": "password01",
  "bitcoinWalletOrImportString": "cRAkycGAHeXCy2d9ezWK...",
  "databaseName": "ion-testnet-bitcoin",
  "genesisBlockNumber": 1764000,
  "mongoDbConnectionString": "mongodb://localhost:27017/",
  "port": 3002,
  "sidetreeTransactionFeeMarkupPercentage": 1,
  "sidetreeTransactionPrefix": "ion:",
  "valueTimeLockAmountInBitcoins": 0
}
```

Same as above, open `json/testnet-core-config.json` (configuration for sidetree core service) and edit appropriately.

```
{
  "batchingIntervalInSeconds": 600,
  "blockchainServiceUri": "http://127.0.0.1:3002",
  "contentAddressableStoreServiceUri": "http://127.0.0.1:3
  "databaseName": "ion-testnet-core",
  "didMethodName": "ion:test",
  "maxConcurrentDownloads": 20,
  "mongoDbConnectionString": "mongodb://localhost:27017/",
  "observingIntervalInSeconds": 60,
  "port": 3000
}
```

Finally, set config path in the source code (.ts) in `ion/src` directory.
By default, the source code is configured for Bitcoin testnet, and there's nothing to do.

Now let's install and build ION project.
Please issue the following commands every time you change the configuration file.

```
npm install
npm run build
```

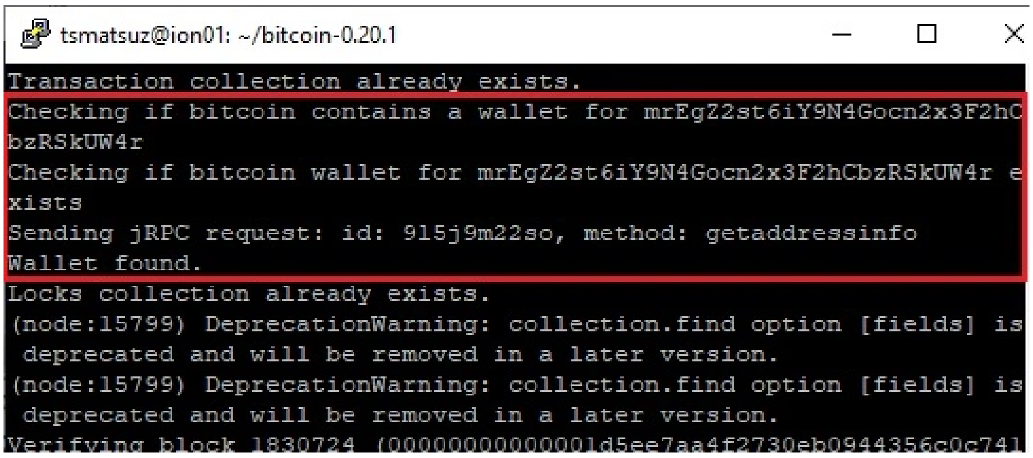
## 5. Run ION (Sidetree)

ION (Sidetree) will run 3 processes – bitcoin service, IPFS service, and core service.

First we run sidetree bitcoin service as follows.
As I show you later, this process (default port 3002) will extract all transactions from Bitcoin core and check whether it's a sidetree transaction for each transactions. Because of this, it will take a lot of time to start service for the first time. (Please wait patiently till it's ready. ION core will fail, if the bitcoin service is not started.)

```
# run sidetree bitcoin service
npm run bitcoin
```

When the bitcoin service starts, the address used in bitcoin service is shown in the console as follows. (In below output, `mrEgZ2st6iY9N4Gocn2x3F2hCbzRSkUW4r` is used.)
Please copy this address for the following instructions.



```
tsmatsuz@ion01: ~/bitcoin-0.20.1
Transaction collection already exists.
Checking if bitcoin contains a wallet for mrEgZ2st6iY9N4Gocn2x3F2hC
bzRSkUW4r
Checking if bitcoin wallet for mrEgZ2st6iY9N4Gocn2x3F2hCbzRSkUW4r e
xists
Sending jRPC request: id: 915j9m22so, method: getaddressinfo
Wallet found.
Locks collection already exists.
(node:15799) DeprecationWarning: collection.find option [fields] is
 deprecated and will be removed in a later version.
(node:15799) DeprecationWarning: collection.find option [fields] is
 deprecated and will be removed in a later version.
Verifying block 1830724 (00000000000001d5ee7aa4f2730eb0944356c0c741
```

(https://tsmatz.files.wordpress.com/2020/09/20200904_bitcoin_address.jpg)

> Note : When it starts, ION might generate wallet addresses (such as, compressed and not compressed) for the same private key.
> You can see all addresses in your wallet by issuing the following Bitcoin command (RPC).
>
> ```
> ./bin/bitcoin-cli -testnet \
>   -rpcuser=myrpc01 \
>   -rpcpassword=password01 \
>   listreceivedbyaddress 1 true
> ```

Next, run sidetree IPFS (content addressable store) service as follows.

```
# run sidetree IPFS service
npm run ipfs
```

After the above services (both bitcoin service and IPFS service) are successfully started, run core service finally as follows.

```
# run sidetree core
npm run core
```

As a result, the following 5 processes will be running in your ION node. Please ensure to launch these processes every time to start ION node.

- MongoDB daemon
- Bitcoin Core
- Sidetree Bitcoin Service
- Sidetree IPFS Service
- Sidetree Core

# Operations and Resolution

Now you are ready to use Sidetree DID network, ION !
In this post, we simply create and resolve a DID for tutorial, and see how it works in this implementation.

## 1. Generate New DID

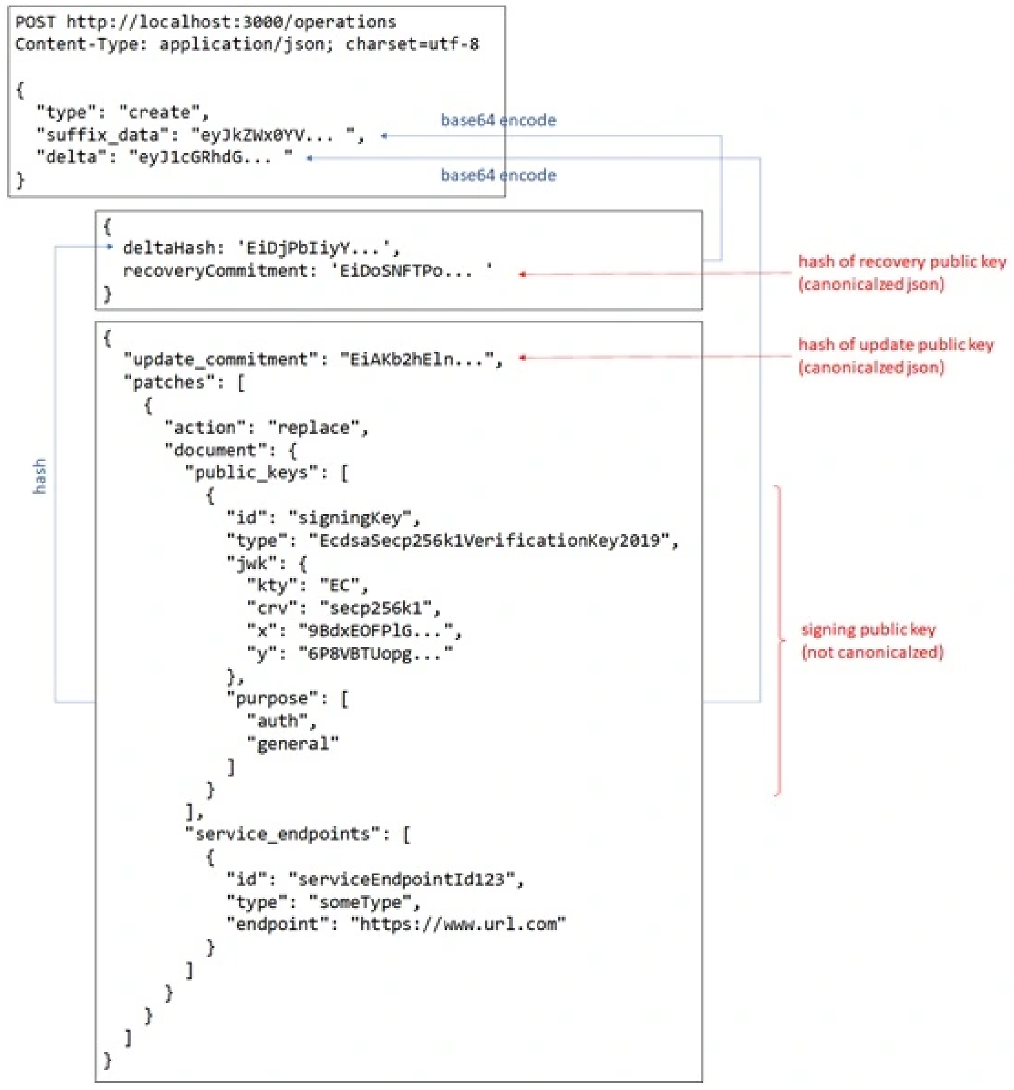
In order to issue a transaction in Bitcoin, ensure that your wallet address (used in sidetree bitcoin service) contains sufficient funds for write operations, else you will see the error. In this tutorial, it will approximately cost 0.0005 BTC for one create operation. (The real fee is calculated along with the size of transaction.)
If you are testing on testnet, you can use a Bitcoin testnet faucet for funding. Search a faucet site on the web, and charge BTCs.

> Note : Please pick up the right address for bitcoin service. (See above.) When the error occurs in write operation, sidetree bitcoin service will show the wallet address in a console.

For issuing a create operation, first you generate a create operation's payload, and then post this payload by HTTP.
There are multiple steps to generate a create operation's payload. Here I show you the outline of steps.

1. Generate key pairs (private key and public key) using SECP256K1 with ECDSA algorithm.
   3 pairs of keys – update key, recovery key, and signing key – should be generated in this phase. The signing key is the key that is used for DID user to generate signature. Other keys are used for managing this DID.
2. Compute hash from public key document.
   A public key document is a json value. Then it will be canonicalized by JCS and generate a hash with Multihash format. SHA2-256 (which code is 18 or 0x12) is used for the hash function.
3. Generate the base64-encoded entity including these hash values and document.

The following illustrates this procedure in the picture.
As a result, the following `suffix_data` and `delta` are submitted to Sidetree by HTTP POST method.



```
POST http://localhost:3000/operations
Content-Type: application/json; charset=utf-8

{
  "type": "create",
  "suffix_data": "eyJkZWx0YV... ",
  "delta": "eyJ1cGRhdG... "
}
```

```
{
  deltaHash: 'EiDjPbIiyY...',
  recoveryCommitment: 'EiDoSNFTPo... '
}
```

```
{
  "update_commitment": "EiAKb2hEln...",
  "patches": [
    {
      "action": "replace",
      "document": {
        "public_keys": [
          {
            "id": "signingKey",
            "type": "EcdsaSecp256k1VerificationKey2019",
            "jwk": {
              "kty": "EC",
              "crv": "secp256k1",
              "x": "9BdxEOFPlG...",
              "y": "6P8VBTUopg..."
            },
            "purpose": [
              "auth",
              "general"
            ]
          }
        ],
        "service_endpoints": [
          {
            "id": "serviceEndpointId123",
            "type": "someType",
            "endpoint": "https://www.url.com"
          }
        ]
      }
    }
  ]
}
```

base64 encode
base64 encode
hash
hash of recovery public key (canonicalzed json)
hash of update public key (canonicalzed json)
signing public key (not canonicalzed)

(https://tsmatz.files.wordpress.com/2020/09/20200904_sidetree_operation.jpg)

You might think that the process to generate this payload is so cumbersome. In ION, you can easily generate this payload by using ION CLI as follows. By running this command, corresponding key files (which includes a private key) are also generated and saved in your current (working) directory.

```
# install ION-cli on /usr/local/bin
cd ion
npm install
npm run build
sudo npm install -g .

# generate a create operation
ion operation create
```

Output :

```
DID: did:ion:test:EiCsnBO7XrB9hL96xvQ2R846j_Ebuyg3H05o4BOS

Recovery private key saved as: EiCsnBO7XrB9hL96xvQ2R846j_E
```

```
Siging private key saved as: EiCsnBO7XrB9hL96xvQ2R846j_Ebu

Create request body:
{
  "type": "create",
  "suffix_data": "eyJkZWx0YV...",
  "delta": "eyJ1cGRhdG..."
}
```

When you post this payload into ION (Sidetree), it soon returns HTTP 200 (success status) as follows. However this operation will be added in operation queue internally.

```
POST http://localhost:3000/operations
Content-Type: application/json; charset=utf-8

{
  "type": "create",
  "suffix_data": "eyJkZWx0YV...",
  "delta": "eyJ1cGRhdG..."
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
  "@context": "https://www.w3.org/ns/did-resolution/v1",
  "didDocument": {
    "id": "did:ion:test:EiCsnBO7XrB9hL96xvQ2R846j_Ebuyg3HO
    "@context": [
      "https://www.w3.org/ns/did/v1",
      {
        "@base": "did:ion:test:EiCsnBO7XrB9hL96xvQ2R846j_E
      }
    ],
    "service": [
      {
        "id": "#serviceEndpointId123",
        "type": "someType",
        "serviceEndpoint": "https://www.url.com"
      }
    ],
    "publicKey": [
      {
        "id": "#signingKey",
        "controller": "",
        "type": "EcdsaSecp256k1VerificationKey2019",
        "publicKeyJwk": {
          "kty": "EC",
          "crv": "secp256k1",
          "x": "IdNDV0Qv-Iw_dUYeF4gw29a0s-TRD3c84E7v2UlUqYI
          "y": "_6WvcTWyOvuAEXPdRB_IpJa7HkRUiSQ7FdXqeBDRj6M
        }
      }
    ],
    "authentication": [
      "#signingKey"
    ]
  },
  "methodMetadata": {
    "recoveryCommitment": "EiCWhklxteAU_Jr-l5Kh4fA2QRDGmrRe
    "updateCommitment": "EiBuejn9rZnQaQQB1o0J5vIAAHEPaFcYK
  }
}
```

After a while, the queued operation (write-batch operation) will be periodically extracted and executed by Sidetree scheduler called **Batch Scheduler**.

In this process, the operation (the above request) is written as 3 types of files – chunk file, map file, and anchor file – in content addressable store, IPFS. (These hash in content addressable store will be shown on the console in sidetree core service.) And the hash of anchor file is submitted into Bitcoin blockchain.
In the submission to blockchain, the hex string of the following string buffer is broadcasted (by calling `sendrawtransaction` RPC call) to Bitcoin blockchain as a transaction of `OP_RETURN` script. (Using unspendable `OP_RETURN`, you can post an arbitrary data in Bitcoin.)
By monitoring Bitcoin blocks, you can take this hex string in blockchain network and extract the hash of anchor content in IPFS.

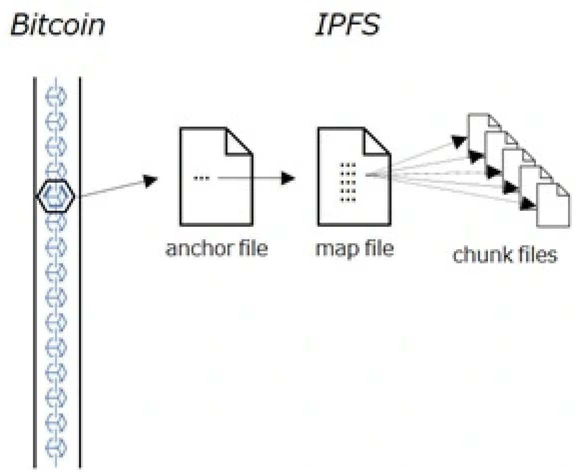
```
ion:{number of operations}.{IPFS hash of anchor file}
```

> Note : In Sidetree, you can also directly invoke the blockchain service as follows.
>
> ```
> POST http://localhost:3002/transactions
> Content-Type: application/json; charset=utf-8
>
> {
> ```

```
    "minimumFee":10,
    "anchorString":"1.EiDPEPJIzL..."
  }
```

> Note : This hash of IPFS content in ION is Base64-URL encoded MultiHash. Please convert this string to Base58 encoded hash – which is used for ordinary IPFS commands – as follows.
>
> ```
> // Node.js Example (Base64 Url Encode -> Base 58 Encoc
> const multihashes = require('multihashes');
> const base64url = require('base64url');
> base64url_hash = 'EiDEqEht-XMKdNjmRAi-T5fGD4PjmFrNAYJc
> bin_hash = base64url.toBuffer(base64url_hash);
> base58_hash = multihashes.toB58String(bin_hash);
> console.log(base58_hash); // QmbaKSyFm9znhgwbNMgQj7Tyr
> ```

To save blockchain transactions for large Sidetree operations, multiple operations can be wrapped as one anchor and one Bitcoin transaction. Other files (corresponding map file and multiple chunk files) are navigated from the path (URI) written in anchor file and ION can extract entire documents. (See the specification (https://identity.foundation/sidetree/spec/) for the format details about anchor, map, and chunk.)



The minimum fee (`minimumFee`) for sending transaction is 10 Satoshi = 0.00000010 BTC, however the real fee is calculated by the size of transaction. (See above.) Please ensure to charge sufficient funds in your wallet address used in sidetree bitcoin service, else you will see the error in the console of sidetree core service.

After the transaction is submitted to Bitcoin, please wait till the transaction is mined and confirmed in a network. (The transaction hash is shown in the console of sidetree bitcoin service, then please copy this value and monitor the status in Bitcoin Testnet Explorer (https://blockstream.info/testnet/).)

> Note : If some invalid writing operation remains in queue, please remove this item in MongoDB as follows. (The queued batch writing operations will be invoked periodically by Batch Scheduler.)
>
> ```
> mongo
> > show databases
> admin                 0.000GB
> config                0.000GB
> ion-testnet-bitcoin  0.007GB
> ion-testnet-core      0.000GB
> local                 0.000GB
> > use ion-testnet-core
> > show collections
> operations
> queued-operations
> transactions
> unresolvable-transactions
> > db["queued-operations"].count()
> 1
> > db["queued-operations"].remove({})
> > exit
> ```

## 2. Resolve

Sidetree service will periodically extract the confirmed blocks on Bitcoin network, and process sidetree's transactions in the following steps.
The component for running this process is called **Observer**.

1. Sidetree bitcoin service will extract the confirmed blocks (and these will be stored in local block store (MongoDB)), and expand all transactions in these blocks.
2. Sidetree bitcoin service will check whether it's a sidetree transaction or not for each transactions. As I mentioned above, the ION transaction will include `OP_RETURN` data with the prefix "`ion:`".
3. If a sidetree's transaction is found, sidetree bitcoin service will save the transaction in local transaction store (MongoDB).
4. Sidetree core service will periodically extract sidetree's transactions in local transaction store.
   Using `OP_RETURN` data (remember that this is a hash of the anchor file in IPFS), sidetree core service will download 3 files – chunk file, map file, and anchor file – from content addressable store (IPFS). The original operation (a create operation in this case) can be restored using these files.

5. Finally, sidetree core service will store this operation in local operation store (MongoDB).

After all, all ION nodes in Sidetree DID network will extract this operation and save in each local operation stores.

> Note : Please be sure to enable Bitcoin transaction indexer (txindex), since sidetree bitcoin service will query the transactions during this process.

After the operation is successfully restored, you can query (resolve) the original DID everywhere in ION network. (Invoke the following HTTP GET on your ION node.)
The component for running this process is called **Resolver**. The Resolver will just query operations into the local operation store in this case.

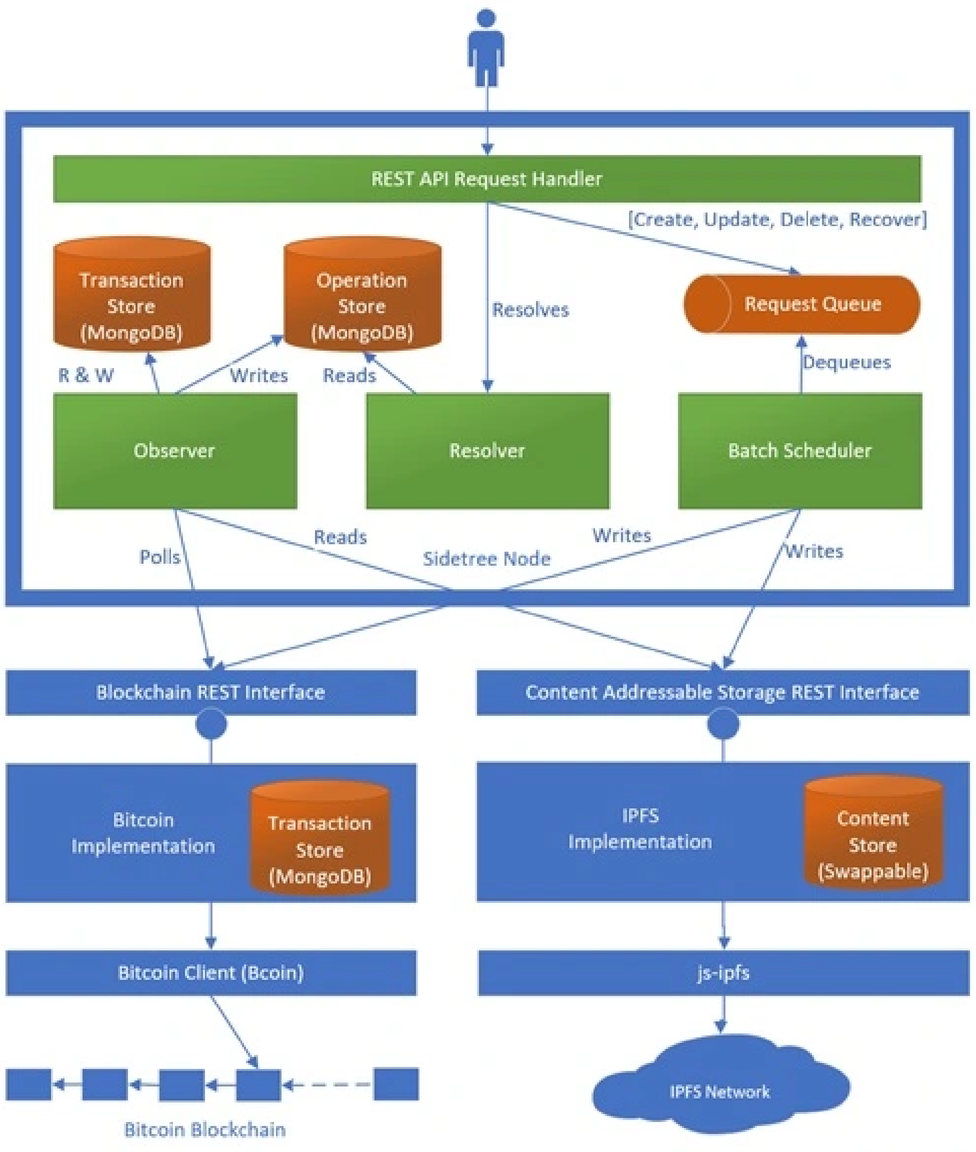
```
GET http://localhost:3000/identifiers/did:ion:test:EiCsnBO7
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
  "@context": "https://www.w3.org/ns/did-resolution/v1",
  "didDocument": {
    "id": "did:ion:test:EiCsnBO7XrB9hL96xvQ2R846j_Ebuyg3HO5
    "@context": [
      "https://www.w3.org/ns/did/v1",
      {
        "@base": "did:ion:test:EiCsnBO7XrB9hL96xvQ2R846j_Eb
      }
    ],
    "service": [
      {
        "id": "#serviceEndpointId123",
        "type": "someType",
        "serviceEndpoint": "https://www.url.com"
      }
    ],
    "publicKey": [
      {
        "id": "#signingKey",
        "controller": "",
        "type": "EcdsaSecp256k1VerificationKey2019",
        "publicKeyJwk": {
          "kty": "EC",
          "crv": "secp256k1",
          "x": "IdNDV0Qv-Iw_dUYeF4gw29a0s-TRD3c84E7v2UlUqYl
          "y": "_6WvcTWyOvuAEXPdRB_IpJa7HkRUiSQ7FdXqeBDRj6l
        }
      }
    ],
    "authentication": [
      "#signingKey"
    ]
  },
  "methodMetadata": {
    "recoveryCommitment": "EiCWhklxteAU_Jr-l5Kh4fA2QRDGmrRe
    "updateCommitment": "EiBuejn9rZnQaQQB1o0J5vIAAHEPaFcYK
  }
}
```

See Sidetree document (https://github.com/decentralized-identity/sidetree/blob/master/docs/core.md) for these implementation with **Batch Scheduler**, **Observer**, and **Resolver**. (See below.)
Here I summarize these process as follows.

- Batch Scheduler processes the queued request. The public key is stored in the decentralized store (IPFS) and the hash of this content is recorded in the decentralized ledger (Bitcoin).
- Observer retrieves the blocks in the decentralized ledger (Bitcoin) and restore document (including public key for signing) from the decentralized store (IPFS).
- Resolver knows how to resolve the requested DID, and retrieves document from Sidetree platform.



(https://tsmatz.files.wordpress.com/2020/09/20200904_sidetree_architecture.jpg)

(From "Sidetree Implementation Document (https://github.com/decentralized-identity/sidetree/blob/master/docs/core.md)" in Github)

## ION on Bitcoin Mainnet …

ION can also cover Bitcoin mainnet. Now Microsoft is providing Verifiable Credentials by Azure AD (https://aka.ms/opendid) for its preview. (See " Verifiable Credential Technical Overview (https://tsmatz.wordpress.com/2020/06/25/what-is-verifiable-credentials/)" for verifiable credentials.)
DID is now going-on and progressing for productions…

Reference :

Sidetree protocol document
https://identity.foundation/sidetree/spec/ (https://identity.foundation/sidetree/spec/)

Sidetree REST API reference
https://identity.foundation/sidetree/swagger/#/

(https://identity.foundation/sidetree/swagger/#/)

https://identity.foundation/sidetree/api/ (https://identity.foundation/sidetree/api/)

## 1 reply »

Dear sir,
I read your blog and deploy ION SIDETREE but I have one error.
I don't know why I deploy bitcoin testnet but when I use command "ion operation create" it generated "did:ion:EiA9VG44rwGopt_oZFlpuGvn2SsalJFTxsDOaiSURjU3yQ".
This DID belongs to the mainnet.
Please help me.

Reply

★(https://tsmatz.wordpress.com/2020/09/01/did-sidetree-ion/?like_comment=39060&_wpnonce=584021e64b)
Like