ION Installation Guide

ION is a decentralized Layer 2 network for Decentralized Identifiers that runs atop the Bitcoin blockchain. Running an minimizes trust in external parties for resolving ION DIDs, helps make the network more resilient and reliable, and pro operator with better DID resolution performance.

The ION node reference implementation a production-stable implementation of the v1 DIF Sidetree specification. Pre installation and operation is attuned for experienced developers who are able to invest the time in running, testing, ar to the codebase. This recommendation will change over time, which contributors will communicate to the community and communications from DIF and collaborating organizations.

The ION node implementation is composed of a collection of microservices. Of these components, the major depend Bitcoin Core, Kubo (IPFS), and MongoDB (for local persistence of data).

1. Preparing your development environment

Hardware

We recommend you run ION on a machine with the following minimum specs:

- i5 processor (2017+ models)
- 6GB of RAM
- 1TB of storage

Operating System

Both Linux and Windows are supported and tested. For Linux, the setup is verified on Ubuntu 18, so we recommend distros for Linux setup.

Linux Environment Setup

Snap

We use snap to simplify installation of certain services. Run the following command to install snap:

```
sudo apt install snapd
```

You may need to add the snap binaries directory to your path by adding the following line in ~/.bash_profile

```
PATH="$PATH:/snap/bin"
```

```
source ~/.bash_profile
```

Node.js

Services within ION rely on Node.js version 14. Run the following commands to install Node v14:

```
sudo snap install node --classic --channel=14
```

build-essential

Building ION requires your distro's equivalent of Ubuntu's 'build-essential', e.g. make, g++, etc.

```
sudo apt install build-essential
```

Windows Environment Setup

Go go https://nodejs.org, download and install the latest v14 of Node.js.

Inbound Ports to Open

If you wish to run a node that writes ION DID operations, you will need to enable uPnP on your router or open ports
`4003` so that the operation data files can be served to others via IPFS.

Testnet or Mainnet

> NOTE: This guide describes steps to setup an ION node targeting bitcoin testnet, but can be used to target the bitc
> by substituting testnet configs for mainnet.

Bitcoin and ION need to be configured to use either `testnet` (for development) or `mainnet` (for production). If you
service from `testnet` to `mainnet` or vice versa, the other services will also need to be rebuilt to match. Default con
`testnet` are not valid for `mainnet` and services will fail to start if they are mismatched.

2. Setting up Bitcoin Core

An ION node needs a trusted Bitcoin peer for fetching and writing ION transactions. We use Bitcoin Core for this.

Automated script for installing Bitcoin Core on Linux

If you would like to install and start Bitcoin Core automatically on Linux, you can review and run the automated script
the Sidetree repo.

> NOTE: Initial synchronization takes ~2 hours for testnet on a 2 core machine with an SSD.

Installing Bitcoin Core Manually

You can find Windows and Linux binaries for Bitcoin Core releases here.

On Linux:

Create a configuration file (`bitcoin.conf`) designating

1. the path you would like the Bitcoin data to be stored in (the `[datadir]`)
2. a username (e.g. `admin`)
3. a password (must match `ion-bitcoin` 's configuration later)

Testnet	Mainnet
`testnet=1` `server=1` `datadir=~/.bitcoin` `rpcuser=<your-rpc-username>` `rpcpassword=<your-rpc-password>` `txindex=1`	`server=1` `txindex=1` `datadir=~/.bitcoin` `rpcuser=<your-rpc-username>` `rpcpassword=<your-rpc-password>`

Start Bitcoin Core and let it sync:

```
./bin/bitcoind --config bitcoin.conf
```

> NOTE: You can add `--daemon` to run bitcoind as a daemon process.

On Windows:

Running Bitcoin Core with friendly UI after install:

```
bitcoin-qt.exe -testnet -datadir=<path-to-store-data> -server -rpcuser=<your-rpc-username> -rpcpassw
```

3. Installing Kubo (IPFS)

Follow the instruction found at IPFS website to install Kubo CLI (IPFS implementation), you can also use Docker imag
IPFS Desktop which internally installs Kubo daemon, it provides you with a user friendly UI.

4. Setting up MongoDB

On Linux:

The default persistence option for storing data locally is MongoDB, though it is possible to create adapters for other d
use the default MongoDB option, you'll need to install MongoDB community build:

- Download as a `deb` package: https://www.mongodb.com/download-center/community.
- Installation doc: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/

> NOTE: You may not have all the dependencies required to run MongoDB, if so you can run `sudo apt-get instal`
> them in.

You'll probably want to store the data from the Mongo instance in the same drive you chose to store the blockchain da
large amount of storage required. Set the directory for this by creating a `db` folder in the location you chose and run
`dbpath ~/YOUR_LOCATION/db`

On Windows:

Download and install MongoDB from https://www.mongodb.com/download-center/community.

> NOTE: To view MongoDB files with a more approachable GUI, download and install MongoDB Compass:
> https://docs.mongodb.com/compass/master/install/

5. Configure & Build ION Microservices

Clone https://github.com/decentralized-identity/ion:

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

Example configuration files for both `testnet-` and `mainnet-` can be found under the top-level `config/` directory.

> NOTE: If not specified, `json/testnet-*-*.json` **files are used as default configuration values**. Be sure to sta
> whichever config template (`testnet-` or `mainnet-`) is right for your use case.

Create your configuration files from templates

Copy the configuration files `<testnet-or-mainnet>-bitcoin-config.json` and `<testnet-or-mainnet>-bitcoin-versioning.json` to another directory, (e.g. `/etc/ion/`)

Update configuration files

Update the ION Bitcoin microservice (e.g. `/etc/ion/testnet-bitcoin-config.json`):

- `bitcoinPeerUri`
  - Ensure it points to the RPC endpoint of the Bitcoin Core client you setup earlier in this guide

  - testnet: `http://localhost:18332`
  - mainnet: `http://localhost:8332` (assuming default Bitcoin Core configuration from Step 2)
- `bitcoinDataDirectory`
  - This is an ***optional*** config value. By configuring this value, instead of using rpc call to initialize Bitcoin micros node will read from the block binary files. This is useful in speeding up init time if you have fast access to the SSD is optimal). If the files are stored and retrieved across network, such as on the cloud in AWS S3 Bucket Storage, then this will be slower than using RPC as it has to download GB worth of files.
  - Leave it blank if you do not wish to init from file. If you want to init from files, it needs to point to the block file
  - testnet: `[datadir]/testnet3` .
  - mainnet: exactly the same as the `[datadir]` value configured for Bitcoin Core in Step 2.
- `bitcoinWalletImportString`
  - if you intend to write DID operations, populate it with your private key, else use any generated import string **bitcoin**
  - testnet: (a valid `testnet` example wallet will be generated each time `ion-bitcoin` fails to load a valid WIF startup as part of its error message. You can use one of those values for testing as well
  - mainnet: (must be a mainnet-compatible WIF)
- `bitcoinRpcUsername` & `bitcoinRpcPassword`
  - must match what was set in `bitcoin.conf` from step 2.

Update the configuration for the ION core service under `json/testnet-core-config.json` :

- `didMethodName`
  - testnet: `ion:test`
  - mainnet: `ion`

NOTE: You can set a few config variables via environment variables for simplicity. The following env variables, if set, values listed in the config files.

Environment Variable	Config Mapping
`BITCOIN_DATA_DIR`	**bitcoinDataDirectory**
`BITCOIN_RPC_PASSWORD`	**bitcoinRpcPassword**
`BITCOIN_WALLET`	**bitcoinWalletImportString**
`BITCOIN_ENDPOINT`	**bitcoinPeerUri**
`MONGO_ENDPOINT`	**mongoDbConnectionString**

Run the following commands to build ION:

```
npm i
npm run build
```

> NOTE: You may nee to run `npm install tsc` before running `npm run build` to install TypeScript in Linux/Mac e

> NOTE: You **must rerun** `npm run build` every time a configuration JSON file is modified.

6. Run ION Bitcoin microservice

Update the paths below to where you edited and saved the config files from the previous step.

```
ION_BITCOIN_CONFIG_FILE_PATH=/etc/ion/testnet-bitcoin-config.json
ION_BITCOIN_VERSIONING_CONFIG_FILE_PATH=/etc/ion/testnet-bitcoin-versioning.json
npm run bitcoin
```

> NOTE: This service will fail to start until your Bitcoin Core client has blocks past the ION genesis block. Please wa again later if this happens.

7. Run ION core service

(Optional) Create your configuration files from templates

> NOTE: This is not required when using `testnet` because the defaults are sufficient

Copy the configuration files `<testnet or mainnet>-core-config.json` and `<testnet or mainnet>-core-version:`
another directory, (e.g. `/etc/ion/` or `~`)

Start a new console and run the following command to start the core service.

NOTE: You can set a few config variables via environment variables for simplicity. The following env variables, if set,
values listed in the config files.

Environment Variable	Config Mapping
`IPFS_ENDPOINT`	**ipfsHttpApiEndpointUri**
`BLOCKCHAIN_SERVICE_ENDPOINT`	**blockchainServiceUri**
`MONGO_ENDPOINT`	**mongoDbConnectionString**

```
ION_CORE_CONFIG_FILE_PATH=/usr/local/src/ion/config/testnet-core-config.json
ION_CORE_VERSIONING_CONFIG_FILE_PATH=/usr/local/src/ion/config/testnet-core-versioning.json
npm run core
```

> NOTE: This service will fail to start until your ION Bitcoin service has started successfully.

Give it some time to synchronize ION transactions.

8. Verify ION is working properly

Check the following DID resolution in your browser:

- testnet: http://localhost:3000/identifiers/did:ion:test:EiClWZ1MnE8PHjH6y4e4nCKgtKnI1DK1foZiP61I86b6pw
- mainnet: http://localhost:3000/identifiers/did:ion:EiClkZMDxPKqC9c-umQfTkR8vvZ9JPhl_xLDI9Nfk38w5w