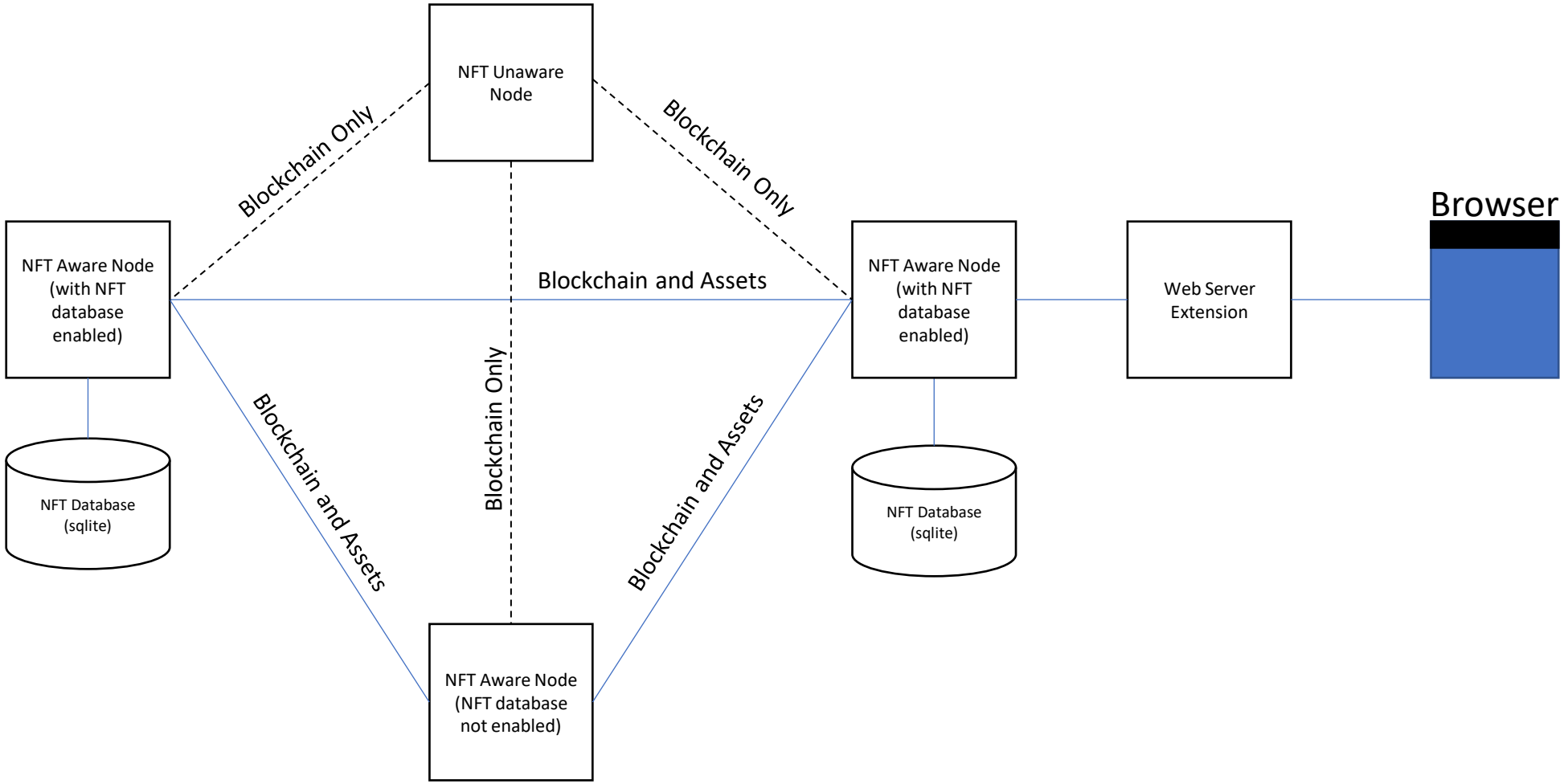
# Dynamo Coin NFT technical architecture



# Dynamo Coin NFT workflows

**Create Asset Class** – An asset class is a group of assets which are linked together by the asset class hash. Every asset must belong to an asset class.
Steps:

- Generate asset class hash and submit to blockchain with “sendtoaddress” RPC
- Wait for transaction to be mined and confirmed (approx. 6 blocks is usually ok)
- Load asset class data to NFT database using nft RPC

**Create Asset** – An asset is an individual, serialized, NFT. An asset includes an asset class hash, optional text metadata (up to 64K bytes) and optional binary data up to (16MB).

- Generate asset hash and submit to blockchain with “sendtoaddress” RPC
- Wait for transaction to be mined and confirmed (approx. 6 blocks is usually ok)
- Load asset data to NFT database using nft RPC

**Send Asset Class or Asset** – Owners of asset classes and assets can send them to another wallet address which will become the new owner.

- Generate send command and submit to blockchain.
- Wait for transaction to be mined and confirmed (approx. 6 blocks is usually ok)

Demonstration C# code on how to complete the above steps is located in the github repository.

https://github.com/dynamofoundation/dyn-nft-loader

Setting up a full node

**Non-NFT aware** – does not relay any NFT data – for users who are in a highly regulated environment or who want to conserve bandwidth (e.g. VMs on a hosting service). Install the non-nft aware full node from the github repo.

- Protocol version 70016 – this does not send or receive any NFT data at all (except the mined hash which is just random)

**NFT aware without database** – this will relay NFT data over the network but will not store NFT data in persistent storage. This is for users who want to use NFT data RPCs for something like web hosting or P2P marketplaces but don't want to commit storage. Install the nft aware full node from github but do not enable the database in the dynamo.conf file.

- Protocol version 70017 – this will relay NFT data and cache up to 100 asset classes and 100 assets in memory

**NFT aware with database** – this will relay NFT data over the network and will store the NFT data in persistent storage. This is for users looking to create asset classes and assets and for users who want to independently verify the content of NFTs. Install the nft aware full node from github and enable the nft database in the dynamo.conf file.

- Protocol version 70017 – this will relay NFT data and cache up to 100 asset classes and 100 assets in memory

dynamo.conf file NFT switches
nftnode (boolean) – true = NFT database enabled, false = NFT database disabled
nftdbkey (string) – required if NFT database is enabled – this is the encryption string for the NFT database

- The NFT database is being encrypted to support an upcoming implementation for NFT proof of storage reward

NOTE: NFT implementation is not a hard fork. The existing DYN blockchain and all consensus rules are preserved in the NFT implementation and users are free to run any version of the full node and wallet software.

# Creating NFT asset classes and assets

**Using dyn-nft-loader**

This command line tool is used to load NFT asset classes and NFT assets to the chain. It is written in C# .net core and is therefore able to run on Windows, Mac and Ubuntu. A 64 bit windows executable is available on the github repo.

dyn-nft-loader will look for a text file in the directory of the executable called settings.txt. This file contains information about the full node that you will be loading assets to. The full node must be NFT aware and must be storing the NFT database.

Sample contents (must be valid JSON and requires these three entries:
{
“FullNodeRPC” : “http://192.168.1.1:6433”,
“FullNodeUser” : “user”,
“FullNodePass” : “pass”
}

This information will be used by dyn-nft-loader to contact your full node. Note that your full node must have a wallet loaded with a balance of at least 0.0001 DYN. Each NFT command that mines a hash uses the sendtoaddress RPC call with a value of 0.0001. This is not the cost of the command, but there needs to be at least this much in the amount in order to make the minimum relay fee default so that the transaction will be propagated on the network. The actual cost of the NFT mining is determined by the current network fee and the size of the command. Most NFT commands are about 40 bytes plus overhead which results in a fee of around 0.00000150 DYN per command as of July 2021.

dyn-nft-loader can only create assets which are owned by an address that is located in the full node wallet that is loaded. The loader does not load a wallet, so you must explicitly load a wallet (either with the QT client or with the RPC “loadwallet” command) before creating any NFTs. In order to verify that the owner is valid the full node consensus checks the claimed owner against the transaction signature for authenticity.

Note that a full node must be run as NFT aware from genesis. An existing full node cannot be “converted” into an NFT aware node because the entire chain of NFT data and ownership transfer must be processed.

-continued-

dyn-nft-loader command usage

**create_asset_class**
dyn-nft-loader create_asset_class <owner address> “<metadata>” <max serial>
- Returns the hash of the asset class created or “error” if the create was not successful

**create_asset**
dyn-nft-loader create_asset <hash of asset class> <owner address> “<metadata>” <serial #> <binary data file name>
- Returns the hash of the asset created or “error” if the create was not successful

**send_asset_class**
dyn-nft-loader send_asset_class <hash of asset class>  <current owner address> <new owner address>
- Returns the hash of the transaction_id or “error” if the send was not successful

**send_asset**
dyn-nft-loader send_asset <hash of asset> <current owner address> <new owner address>
- Returns the hash of the transaction_id or “error” if the send was not successful

**get_asset_class**
dyn-nft-loader get_asset_class <hash of asset class>
> Returns hex encoded asset class data or “error” if the get was not successful

**get_asset**
dyn-nft-loader get_asset <hash of asset>
> Returns hex encoded asset data or “error” if the get was not successful

Owner address must be a valid bech32 address that can be received by the currently loaded wallet
Max length of meta data is 65536 bytes
Max serial must be between 0 and 2^64 - 1
Max size of binary data in file is 2^24 – 1  (16MB)
Currently loaded wallet must have at least 0.0001 DYN balance – actual transaction fee is determined by network, any overage will be refunded as change
Serial # must be between 0 and the max serial specified in the asset class
Asset class must be owned by the owner address when creating an asset
Asset class hash must be mined into blockchain before creating an asset (e.g. wait 6+ confirmations)

# dyn-nft-loader command examples

dyn-nft-loader create_asset_class dy123456789123456789zzz “This is a test asset class” 100

dyn-nft-loader create_asset 09be351109be351209ff351109be6641 dy123456789123456789zzz “This is a test asset class” 1 “test.jpg”

dyn-nft-loader send_asset_class 09be351109be351209ff351109be6641 dy123456789123456789zzz dyaaaanewguy1234567

dyn-nft-loader send_asset 09be351109be351209ff351109be6641 dy123456789123456789zzz dyaaaanewguy1234567

dyn-nft-loader get_asset_class 09be351109be351209ff351109be6641

dyn-nft-loader get_asset_class 09be351109be351209ff351109be6641

dyn-nft-loader get_asset 09be351109be351209ff351109be6641

# NFT technical design

NFT metadata and binary data is stored in a local sqlite database hosted by nft aware nodes.

Creating an asset class or asset consists of hashing together the binary of the metadata, serial# and binary data (if any). This hash is then combined with the owner address to form the submitted hash which gets mined into the blockchain using sendtoaddress. The full node verifies that the owner address matches the signature. The transaction has a txout with an OP_RETURN command appended which flags the NFT creation. Once mined, the newly created blocks are sent to NFT aware full nodes which decode them and request the NFT asset data from other NFT aware full nodes and replicate that asset data into their local databases. The hash of the NFT is the original hash created in the first step hashed together with the transaction ID, thus guaranteeing uniqueness. Each full node can then replay the same steps as part of consensus by hashing the original data and comparing it to the final NFT hash. If there is a mismatch the full node does not store the NFT in its local database.

NFT asset data is passed through the peer to peer network like all other block data. The NFT aware nodes use protocol version 70017 and only transmit NFT commands to other version 70017 nodes.

The sendtoaddress RPC endpoint has been modified to include an optional “nft_command” parameter which contains a hex string of the NFT operation. This endpoint will mine the NFT hash into the chain. Additional RPC endpoints have been added to support loading NFT asset data to the local full node database for replication.

# NFT data formats

## nft_data parameter of sendtoaddress

### create asset class and create asset:

| 1 byte | 16 bytes |
|---|---|
| command | hash of data (see below) |

Command:

| | |
|---|---|
| 0 | create asset class |
| 1 | create asset |

### send asset class and send asset:

| 1 byte | 1 byte | 16 bytes |
|---|---|---|
| command | sub-command | hash of data (see below) |

Command:

| | |
|---|---|
| 2 | send |

Sub-Command:

| | |
|---|---|
| 0 | send asset class |
| 1 | send asset |

# NFT hash calculation

## NFT asset class hash calculation

1. Calculate the SHA256 hash of the NFT data block – this is submitted to sendtoaddress to get a TXID
2. Append the bech32 format owner address to that hash and take the SHA256 hash
3. Append the TXID to the hash from step 2 and take the SHA256 hash of that – this is the NFT asset class hash

## NFT asset hash calculation

1. Calculate the SHA256 hash of the NFT data block appended with the asset class hash
2. Append the bech32 format owner address to that hash and take the SHA256 hash
3. Append the TXID to the hash from step 2 and take the SHA256 hash of that – this is the NFT asset hash

NFT data block format:

| | | |
|---|---|---|
| Metadata length | 2 bytes | Big Endian encoded, unsigned |
| Metadata | variable | UTF8/ASCII encoded text, not NULL terminated |
| Binary data length | 3 bytes | Big Endian encoded, unsigned |
| Binary data | variable | |
| Max serial or serial # | 8 bytes | Uint64 Big Endian encoded |